БИБЛИОТЕКА ПРОГРАММИСТА

Пол Дейтел Харви Дейтел Александер Уолд

Android для разработчиков

3-е издание

- Android Studio
- Материальный дизайн
- Новая модель разрешений
- Android Wear и Android TV

PRENTICE HALL

ПИТЕР®

ANDROID™ 6 FOR PROGRAMMERS

AN APP-DRIVEN APPROACH, 3/E

DEITEL® DEVELOPER SERIES

PAUL DEITEL • HARVEY DEITEL • ALEXANDER WALD
DEITEL & ASSOCIATES, INC.

PRENTICE HALL

DEITEL®

Boston • Columbus • Indianapolis • New York • San Francisco
Amsterdam • Capetown • Dubai • London • Madrid • Milan • Munich
Paris • Montreal • Toronto • Delhi • Mexico City • Sao Paulo • Sidney
Hong Kong • Seoul • Singapore • Taipei • Tokyo

БИБЛИОТЕКА ПРОГРАММИСТА

Пол Дейтел Харви Дейтел Александер Уолд

Android для разработчиков

3-е издание

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2016

П. Дейтел, Х. Дейтел, А. Уолд

Android для разработчиков. 3-е издание

Серия «Библиотека программиста»

Перевел с английского *Е. Матвеев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Художник	*С. Заматевская*
Корректоры	*С. Беляева, М. Молчанова*
Верстка	*Л. Егорова*

ББК 32.973.2-018.2
УДК 004.451

Дейтел П., Дейтел Х., Уолд А.

Д27 Android для разработчиков. 3-е изд. — СПб.: Питер, 2016. — 512 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-02371-9

Добро пожаловать в динамичный мир разработки приложений для смартфонов и планшетов Android с использованием Android Software Development Kit (SDK), языка программирования Java™, а также новой и стремительно развивающейся среды разработки Android Studio. В основе книги лежит принцип разработки, ориентированной на приложения, — концепции показаны на примере полностью работоспособных приложений Android, а не фрагментов кода.

Более миллиона человек уже воспользовались книгами Дейтелов, чтобы освоить Java, C#, C++, C, JavaScript, XML, Visual Basic, Visual C++, Perl, Python и другие языки программирования. Третье издание этой книги позволит вам не только приступить к разработке приложений для Android, но и быстро опубликовать их в Google Play. Третье издание книги было полностью обновлено и познакомит вас с возможностями Android 6 и Android Studio.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ISBN 978-0134289366 англ.
ISBN 978-5-496-02371-9

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

Подписано в печать 22.04.16. Формат 70×100/16. Бумага писчая. Усл. п. л. 41,280. Тираж 1000. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87

Оглавление

Предисловие

Добро пожаловать в динамичный мир разработки приложений для смартфонов и планшетов Android с использованием Android Software Development Kit (SDK), языка программирования Java™, а также новой и стремительно развивающейся среды разработки Android Studio. Многие навыки программирования для Android, представленные в книге, также применимы к разработке приложений для Android Wear и Android TV, так что книга подготовит вас к разработке и для этих платформ.

В книге представлены передовые технологии разработки мобильных приложений для профессиональных программистов. В основу книги заложен принцип *разработки, ориентированной на приложения*, — концепции разработки продемонстрированы на примере *полностью работоспособных приложений* Android, а не фрагментов кода. Каждая из глав 2–9 начинается с вводной части, в которой вкратце описано разрабатываемое приложение. Затем приводятся результаты тестирования приложения и обзор технологий, применяемых в процессе его разработки. Далее выполняется подробный анализ исходного кода приложения. Исходный код всех приложений доступен на сайте *www.deitel.com/books/AndroidFP3*. Во время чтения книги мы рекомендуем держать исходный код открытым в среде разработки.

Перед разработчиками Android-приложений открываются огромные возможности.

Объемы продаж устройств Android и количество загрузок Android-приложений стремительно растут. Мобильные телефоны Android первого поколения появились на рынке в октябре 2008 года. По данным отчета IDC, к концу первого квартала 2015 года Android принадлежало 78% глобального рынка смартфонов, по сравнению с 18,3% у Apple, 2,7% у Microsoft и 0,3% у Blackberry[1]. Только в 2014 году в продажу поступило около миллиарда устройств на базе Android[2]. На конференции Google I/O в 2015 году компания Google объявила, что за предшествующие 12 месяцев в магазине Google Play™ — магазине для

[1] *http://www.idc.com/prodserv/smartphone-os-market-share.jsp*.

[2] *http://www.businessinsider.com/android-1-billion-shipments-2014-strategy-analytics-2015-2*.

приложений Android — количество установок приложений достигло 50 миллиардов[1]. Ожесточенная конкуренция среди разработчиков популярных мобильных платформ и мобильных сервисов приводит к быстрому внедрению инноваций и стремительному обвалу цен. Благодаря соперничеству между десятками производителей устройств Android ускоряется внедрение аппаратных и программных инноваций в сообществе Android.

Авторские права и лицензии на код

Целевая аудитория

Предполагается, что читатели этой книги знают язык Java и имеют опыт объектно-ориентированного программирования. Также предполагается, что читатель знаком с XML — как вы увидите, проекты Android содержат много файлов XML, хотя программист часто работает с XML в редакторе, в основном скрывающем значительную часть XML. Мы используем только завершенные рабочие приложения, поэтому, даже не зная Java, но имея опыт объектно-ориентированного программирования на C++, C#, Swift или Objective-C, вы сможете быстро освоить излагаемый в книге материал, а заодно узнать много полезного о Java и объектно-ориентированном программировании.

Эта книга *не является* учебником по Java. Если вы хотите глубже освоить Java, обратите внимание на следующие книги:

[1] *http://bit.ly/2015GoogleIOKeynote*.

Java for Programmers, 3/e (*http://www.deitel.com/books/javafp3*)

Java Fundamentals, 2/e: Parts I and II LiveLessons videos (*www.deitel.com/books/LiveLessons*)

Java How to Program, 10/e (*www.deitel.com/books/jhtp10*)

Если вы не знакомы с XML, обращайтесь к бесплатным онлайновым учебникам:

- *http://www.ibm.com/developerworks/xml/newto*
- *http://www.w3schools.com/xml/default.asp*
- *http://bit.ly/DeitelXMLBasics*
- *http://bit.ly/StructureXMLData*

Особенности книги

Разработка, ориентированная на приложения. В каждой из глав 2–9 представлено одно полное приложение — рассмотрены функции приложения, приведены снимки экрана выполняющегося приложения, результаты тестовых запусков и обзор технологий и архитектуры, используемых при создании приложения. Затем мы строим графический интерфейс приложения, представляем его полный исходный код и проводим подробный анализ этого кода; обсуждаем концепции, применяемые в программировании, и демонстрируем функциональность Android API, используемую при создании приложения.

Android 6 SDK. В книге рассматриваются новые возможности Android 6 SDK (Software Development Kit).

Android Studio IDE. Бесплатная среда Android Studio (созданная на базе IntelliJ IDEA Community Edition) в настоящее время является основной интегрированной средой, рекомендуемой для разработки приложений Android (исходные средства разработки Android работали на базе Eclipse IDE). Среда Android Studio в сочетании с бесплатным пакетом Android Software Development Kit (SDK) и бесплатным пакетом Java Development Kit (JDK) предоставляет все необходимое для создания, запуска и отладки приложений Android, поддержки их распространения (например, отправки в магазин Google Play™) и т. д. Инструкция по поводу загрузки и установки этих продуктов приведена в разделе «Подготовка».

Материальный дизайн. В Android 5 компания Google представила новый стиль приложений, основанный на спецификации материального дизайна:

http://www.google.com/design/spec/material-design/introduction.html

В этой спецификации Google приводит обзор целей и принципов материального дизайна, а также подробную информацию по методам анимации, стилевому оформлению экранных элементов, позиционированию элементов, использованию конкретных компонентов интерфейса пользователя, стандартным схемам взаимодействия с пользователем, доступности, интернационализации и т. д. В настоящее время компания Google использует принципы материального дизайна как в своих мобильных приложениях, так и в приложениях для браузеров.

Материальный дизайн — весьма обширная тема. В этой книге мы сосредоточимся на следующих его аспектах:

Использование встроенных материальных тем Android — темы гарантируют, что внешний вид встроенных компонентов Android будет соответствовать принципам материального дизайна.

Использование встроенных шаблонов приложений Android Studio — эти шаблоны были разработаны компанией Google в соответствии с принципами материального дизайна.

Использование компонентов интерфейса пользователя, рекомендованных в спецификации Google для конкретных целей, таких как `FloatingActionButton`, `TextInputLayout` и `RecyclerView`.

Кроме спецификации материального дизайна Google, также стоит прочитать книгу «Android User Interface Design: Implementing Material Design for Developers, 2nd Edition»:

http://www.google.com/design/spec/material-design/introduction.html

http://bit.ly/IanCliftonMaterialDesign

нашего коллеги и рецензента прошлых изданий книги «Android for Programmers» Иэна Клифтона (Ian Clifton). Несколько слов от Иэна: «Компания Google опубликовала свои рекомендации материального дизайна в 2014 году. Так была создана система дизайна, которая определяла желательный внешний вид и поведение приложений. Целью было формирование инфраструктуры дизайна, которая бы делала приложения более привлекательными и создавала целостный стиль поведения, отсутствовавший ранее. В книге подробно рассматривается концепция материального дизайна, благодаря которой дизайн, ориентированный на пользователя, теория применения цветов, работа со шрифтами, стандартные схемы взаимодействия и другие аспекты оформления станут доступны для всех разработчиков».

Поддержка и библиотеки совместимости. При использовании новых возможностей Android разработчики часто сталкиваются с проблемой обеспечения

обратной совместимости с более ранними платформами Android. Многие новые возможности Android теперь вводятся через библиотеки поддержки. Это позволяет разработчику использовать новые возможности в приложениях, ориентированных как на современные, так и на старые платформы Android. К числу таких библиотек принадлежит и AppCompat. Шаблоны приложений в Android Studio были обновлены; теперь в них используется библиотека AppCompat и ее темы, что позволяет новым приложениям, которые вы создаете, работать на большинстве устройств на базе Android. Если вы создаете приложения, изначально ориентированные на библиотеку AppCompat, вам не придется изменять реализацию кода, если вы захотите поддерживать старые версии Android в более широкой аудитории пользователей.

Кроме того, на конференции разработчиков Google I/O в 2015 году компания Google представила библиотеку Android Design Support Library:

http://android-developers.blogspot.com/2015/05/android-design-support-library.html

для использования материального дизайна в Android 2.1 и выше. Поддержка материального дизайна также встроена в большинство шаблонов Android Studio.

REST-совместимые веб-сервисы и JSON. В главе 7 представлено приложение `Weather Viewer`, демонстрирующее использование веб-сервисов с поддержкой архитектурного стиля REST (Representational State Transfer) — в данном случае сервиса получения 16-дневного прогноза погоды с сайта *OpenWeatherMap.org*. Этот веб-сервис возвращает прогноз в формате JSON (JavaScript Object Notation) — популярном текстовом формате обмена данными, используемом для представления объектов в виде пар «ключ—значение». Приложение также использует классы из пакета *org.json* для обработки ответа веб-сервиса.

Разрешения Android 6.0. В Android 6.0 используется новая модель разрешений, разработанная для удобства пользователя. До выхода Android 6.0 пользователь обязан был во время установки заранее предоставить все разрешения, которые могли когда-либо понадобиться приложению. Нередко это отпугивало пользователей от установки приложений. В новой модели приложение устанавливается, не запрашивая никаких разрешений. Вместо этого пользователю предлагается запросить разрешение только при первом использовании соответствующей возможности. Глава 5 знакомит читателя с новой моделью разрешений и демонстрирует, как в ней запросить у пользователя разрешение на сохранение изображения на внешнем носителе.

Фрагменты. Начиная с главы 4 мы будем использовать фрагменты для создания и управления частями графического интерфейса каждого фрагмента. Объединяя несколько фрагментов, можно создавать интерфейсы, эффективно использующие пространство экрана планшетов. Разработчик может легко заменять фрагменты, что делает графический интерфейс более динамичным; пример переключения фрагментов рассматривается в главе 9.

Паттерн View-Holder, компоненты ListView и RecyclerView. Каждое из приложений в главах 7–9 отображает прокручиваемый список данных. В главе 7 данные отображаются в списке `ListView`; также в ней описан паттерн View-Holder, повышающий скорость прокрутки за счет повторного использования компонентов графического интерфейса, выходящих за пределы экрана. При работе с `ListView` применение паттерна View-Holder желательно, но не обязательно. В главах 8 и 9 данные выводятся в более гибком и эффективном компоненте `RecyclerView`, для которого паттерн View-Holder обязателен.

Печать. Возможности печати из приложений продемонстрированы на примере класса `PrintHelper` (глава 5), входящего в инфраструктуру печати Android. Класс `PrintHelper` предоставляет пользовательский интерфейс для выбора принтера, метод для проверки того, поддерживает ли заданное устройство печать, а также метод для печати объектов `Bitmap`. Класс `PrintHelper` является частью библиотеки Android Support Library.

Режим погружения. Панель состояния в верхней части экрана и кнопки меню в нижней части можно скрыть, чтобы ваши приложения могли использовать большую часть экрана. Чтобы получить доступ к панели состояния, пользователь проводит пальцем от верхнего края экрана, а к системной панели с кнопками Back, Home и Recent Apps — от нижнего края.

Тестирование на смартфонах Android, планшетах и в эмуляторе. Для достижения оптимального результата приложения следует тестировать на физических смартфонах и планшетах Android. Полезную информацию также можно получить при тестировании в эмуляторе Android (см. раздел «Подготовка»), однако эмуляция создает существенную нагрузку на процессор и может работать медленно, особенно в играх с большим количеством подвижных объектов. В главе 1 перечислены некоторые функции Android, не поддерживаемые эмулятором.

Cloud Test Lab. Google работает над новым сервисом Cloud Test Lab — сайтом для тестирования приложений на широком спектре устройств, ориентаций устройства, локальных контекстов, языков и состояний сети. Вы сможете проводить автоматизированные тесты и получать подробные отчеты со снимками экранов и видеороликами, а также протоколами ошибок, которые помогут найти проблемы и улучшить приложения. Чтобы получить дополнительную информацию и подписаться на уведомления о доступности Cloud Test Lab, посетите страницу:

http://developers.google.com/cloud-test-lab/

Android Wear и Android TV. Android Wear работает на «умных часах»; Android TV работает на некоторых умных телевизорах и медиаплеерах, подключаемых к телевизору (обычно кабелем HDMI). Многие приемы программирования для Android, представленные в книге, также относятся и к разработке приложений для Android Wear и Android TV. Android SDK предоставляет эмуляторы для

Android Wear и Android TV, поэтому вы сможете тестировать свои приложения для этих платформ, даже если у вас нет самих устройств.

За дополнительной информацией об этих технологиях с точки зрения разработчика обращайтесь по адресу

http://developer.android.com/wear/index.html

для Android Wear и

http://developer.android.com/tv/index.html

для Android TV.

Мультимедиа. В приложениях используются разнообразные мультимедийные возможности Android, включая графику, изображения, покадровую анимацию, анимацию и работу с аудио.

Отправка приложений в Google Play. В главе 10 описан процесс регистрации в Google Play и настройки учетной записи для продажи приложений. Вы узнаете, как подготовить приложение к отправке в Google Play, как установить цену на приложение, и познакомитесь с возможностями монетизации приложений через размещение рекламы и внутренние продажи. Также будут представлены ресурсы, которые могут использоваться для маркетинга приложений. Главу 10 можно читать после главы 1.

Как связаться с авторами книги

Мы ждем ваши комментарии, критические замечания, исправления и предложения по улучшению книги. С вопросами и предложениями обращайтесь по адресу *deitel@deitel.com*.

Исправления и уточнения к материалу книги публикуются по адресу

www.deitel.com/books/AndroidFP3

а также в Facebook, Twitter, Google+, LinkedIn и Deitel® Buzz Online.

Посетите сайт *www.deitel.com*, здесь вы сможете:

- загрузить примеры кода;
- ознакомиться с постоянно расширяющимся списком ресурсов по программированию;
- получить обновления к материалу книги;
- подписаться на бесплатный бюллетень Deitel® Buzz Online, распространяемый по электронной почте, по адресу *www.deitel.com/newsletter/subscribe.html*;

- получить информацию об учебных курсах серии Dive Into® Series, проводимых на территории клиента по всему миру.

Благодарности

Спасибо Барбаре Дейтел (Barbara Deitel) за многочасовую работу над проектом — она создавала наши ресурсные центры Android, терпеливо разбираясь во всех технических подробностях.

Мы высоко ценим экстраординарные усилия и 20-летнее наставничество нашего друга и профессионала Марка Л. Тауба (Mark L. Taub), главного редактора издательской группы Pearson Technology Group. Марк со своей группой работает над всеми нашими профессиональными книгами и видеоуроками. Мишель Хаусли (Michelle Housley) привлекла заслуженных участников сообщества Android к рецензированию. Мы выбрали иллюстрацию для обложки, а Чати Презертсит (Chuti Prasertsith) разработал ее дизайн. Джон Фуллер (John Fuller) руководил процессом публикации всех наших книг из серии Deitel Developer Series.

Мы благодарим Майкла Моргано (Michael Morgano), нашего бывшего коллегу по Deitel & Associates, Inc., а теперь Android-разработчика в компании PHHHOTO, соавтора первых изданий этой книги, а также книги «iPhone for Programmers: An App-Driven Approach». Майкл — исключительно одаренный программист.

Наконец, мы благодарим Эбби Дейтел (Abbey Deitel), бывшего президента компании Deitel & Associates, Inc. Эбби закончила школу менеджмента Tepper при Университете Карнеги-Мелон и получила степень бакалавра в области промышленного менеджмента. Она курировала коммерческие операции в компании Deitel & Associates, Inc. на протяжении 17 лет, а также была соавтором некоторых наших публикаций, включая версии глав 1 и 10 из предыдущих изданий.

Рецензенты книг «Android for Programmers: An App-Driven Approach» и «Android How to Program»

Мы хотим поблагодарить рецензентов этого и двух предыдущих изданий книги. Они тщательно проверили текст и предоставили множество рекомендаций по его улучшению: Пол Бойстерьен (Paul Beusterien), главный специалист компании Mobile Developer Solutions; Эрик Дж. Боуден (Eric J. Bowden), главный управляющий компании Safe Driving Systems, LLC; Тони Кантрелл (Tony Cantrell) (Северо-западный технический колледж штата Джорджия); Иэн Дж. Клифтон (Ian G. Clifton), независимый подрядчик, разработчик приложений Android и автор книги «Android User Interface Design: Implementing Material Design for Developers, 2nd Edition»; Даниэль Гэлпин (Daniel Galpin), энтузиаст Android

и автор книги «Intro to Android Application Development»; Джим Хэзевэй (Jim Hathaway), разработчик из компании Kellogg; Дуглас Джонс (Douglas Jones), старший инженер-программист, компания Fullpower Technologies; Чарльз Ласки (Charles Lasky), муниципальный колледж Нагаутук; Энрике Лопес-Манас (Enrique Lopez-Manas), старший специалист по архитектуре Android и преподаватель информатики в Университете Алькала, Мадрид; Себастиан Никопп (Sebastian Nykopp), главный архитектор, компания Reaktor; Майкл Пардо (Michael Pardo), разработчик Android, компания Mobiata; Ронан «Зеро» Шварц (Ronan «Zero» Schwarz), директор по информационным технологиям, компания OpenIntents; Ариджит Сенгупта (Arijit Sengupta), Государственный университет Райта; Дональд Смит (Donald Smith), Колумбийский колледж; Хесус Убальдо (Jesus Ubaldo), Кеведо Торреро, Университет штата Висконсин, Парксайд; Дон Уик (Dawn Wick), Юго-Западный муниципальный колледж; Фрэнк Сю (Frank Xu), Университет Гэннон.

Итак, свершилось! Эта книга поможет вам быстро освоить разработку приложений Android с использованием Android 6 и Android Studio. Мы надеемся, что от чтения этой книги вы получите не меньше удовольствия, чем мы — от ее написания!

Пол Дейтел
Харви Дейтел

Об авторах

Пол Дж. Дейтел (Paul J. Deitel), генеральный и технический директор компании Deitel & Associates, Inc., окончил Массачусетский технологический институт (MIT) по специальности «Информационные технологии» (Information Technology). Обладатель сертификатов Java Certified Programmer, Java Certified Developer и Oracle Java Champion. Пол также получил премию Microsoft® Most Valuable Professional (MVP) по C# в 2012–2014 годах. В Deitel & Associates, Inc. он провел сотни занятий по всему миру для корпоративных клиентов, включая Cisco, IBM, Siemens, Sun Microsystems, Dell, Fidelity, NASA (Космический центр имени Кеннеди), Национальный центр прогнозирования сильных штормов, ракетный полигон Уайт-Сэндз, Rogue Wave Software, Boeing, SunGard Higher Education, Stratus, Cambridge Technology Partners, One Wave, Hyperion Software, Adra Systems, Entergy, CableData Systems, Nortel Networks, Puma, iRobot, Invensys и многих других. Пол и его соавтор, д-р Харви М. Дейтел, являются авторами всемирно известных бестселлеров — учебников по языкам программирования, предназначенных для начинающих и для профессионалов, а также видеокурсов.

Харви М. Дейтел (Dr. Harvey M. Deitel), председатель и главный стратег компании Deitel & Associates, Inc., имеет 50-летний опыт работы в области информационных технологий. Он получил степени бакалавра и магистра Массачусетского технологического института и степень доктора философии Бостонского университета. В 1960-е годы он работал в группах, занимавшихся созданием различных операционных систем IBM, в Advanced Computer Techniques и Computer Usage Corporation, а в 1970-е годы занимался разработкой коммерческих программных систем. Харви имеет огромный опыт преподавания в колледже и занимал должность председателя отделения информационных технологий Бостонского колледжа. В 1991 году вместе с сыном — Полом Дж. Дейтелом — он основал компанию Deitel & Associates, Inc. Харви с Полом написали несколько десятков книг и выпустили десятки видеокурсов LiveLessons. Написанные ими книги получили международное признание и были изданы на китайском, корейском, японском, немецком, русском, испанском, французском, польском, итальянском, португальском, греческом, турецком языках и на языке урду. Дейтел провел сотни семинаров по программированию в крупных корпорациях, академических институтах, правительственных и военных организациях.

Александер Уолд (Alexander Wald), практикант в компании Deitel, помог нам преобразовать книгу и приложения для Android 4.3 и 4.4 с использованием Eclipse на Android 6 с использованием Android Studio. В настоящее время Александер собирается получить ученую степень бакалавра в области «Информационные технологии» в Уорчестерском политехническом институте с непрофильным образованием в области электротехники. Он заинтересовался математикой и наукой в раннем возрасте и пишет программный код около 9 лет. Его вдохновляет страсть к творчеству и новшествам, а также желание поделиться своими знаниями с другими.

О компании Deitel & Associates, Inc.

Компания Deitel & Associates, Inc., основанная Полом Дейтелом и Харви Дейтелом, получила международное признание в области авторских разработок и корпоративного обучения. Компания специализируется на разработке приложений для Android и iOS, языках программирования, объектных технологиях, Интернете и веб-программировании. В число клиентов компании входят многие ведущие корпорации, правительственные агентства, военные и образовательные учреждения. Компания предоставляет учебные курсы, проводимые на территории клиента по всему миру для многих языков программирования и платформ, включая разработку приложений для Android и iOS, Swift™, Java™, C++, C, Visual C#®, Visual Basic® и веб-программирования, с постоянно расширяющимся списком новых языков программирования и тем.

Подготовка

Благодаря этому разделу ваш компьютер будет правильно настроен и подготовлен к выполнению упражнений, приведенных в книге. Средства разработчика Android часто обновляются. Прежде чем читать этот раздел, посетите сайт книги

http://www.deitel.com/books/AndroidFP3

и проверьте, не был ли опубликован обновленный вариант этого раздела.

Требования к аппаратному и программному обеспечению

Для разработки приложений Android необходима операционная система Windows®, Linux® либо Mac® OS X®. Новейшие требования к операционной системе доступны по адресу

http://developer.android.com/sdk/index.html#Requirements

(прокрутите до раздела System Requirements). Для разработки приложений, представленных в книге, используется следующее программное обеспечение:

- Java SE 7 Software Development Kit;
- Интегрированная среда разработки (IDE) Android Studio 1.4;
- Android 6 SDK (API 23).

О том, как установить все эти компоненты, подробно рассказано ниже.

Установка Java Development Kit (JDK)

Для разработки Android-приложений требуется пакет *Java Development Kit* (JDK) версии 7 (JDK 7). Все средства языка Java стандарта JDK 7 поддерживаются в Android Studio, но конструкция `try` с ресурсами поддерживается только

для версий платформы Android уровней API 19 и выше. Чтобы загрузить JDK 7 для Windows, OS X или Linux, перейдите на сайт

http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html

Выберите 32- или 64-разрядную версию в зависимости от типа процессора вашего компьютера и операционной системы. Выполните инструкции по установке по адресу

http://docs.oracle.com/javase/7/docs/webnotes/install/index.html

Android еще не поддерживает некоторые возможности языка Java 8, например лямбда-выражения, новые возможности интерфейса и потоковые API. Вы можете использовать JDK 8 (как поступили мы при разработке приложений этой книги) при условии, что в коде не будут использоваться специфические возможности Java 8.

Установка пакета Android Studio

Android Studio включает последнюю версию Android Software Development Kit (SDK) и базируется на популярной интегрированной Java-среде IntelliJ® IDEA (разработчик — компания JetBrains). Чтобы загрузить Android Studio, откройте страницу

http://developer.android.com/sdk/index.html

и нажмите кнопку Download Android Studio. Когда загрузка завершится, запустите программу установки и выполните инструкции для ее завершения. Если ранее вы устанавливали предыдущую версию Android Studio, в конце процесса установки откроется окно Complete Installation с предложением импортировать старые настройки. На момент написания книги версия Android Studio 1.4 считалась последней официально выпущенной версией, а версия Android Studio 1.5 была доступна как ознакомительная.

Ознакомительные версии

При построении приложений, предназначенных для публикации в Google Play и других магазинах приложений, желательно использовать текущую версию Android Studio. Если вы хотите пользоваться новыми возможностями ознакомительных версий Android Studio, Google публикует эти версии в каналах Canary Channel и Beta Channel. Android Studio можно настроить для загрузки обновлений

на этих каналах. Чтобы обновить Android Studio до последней ознакомительной или бета-версии:

1. Откройте Android Studio.
2. В окне Welcome to Android Studio щелкните на кнопке Configure.
3. Щелкните на кнопке Check for update.
4. В диалоговом окне Platform and Plugin Updates щелкните на ссылке Updates link.
5. В диалоговом окне Updates выберите в раскрывающемся списке справа от флажка Automatically check updates for канал Canary Channel или Beta Channel.
6. Щелкните на кнопке OK, затем на кнопке Close.
7. Снова щелкните на кнопке Check for update.
8. Среда разработки выполняет проверку и сообщает, имеются ли возможные обновления.
9. Щелкните на кнопке Update and Restart, чтобы установить новейшую версию Android Studio.

Если ранее в Android Studio был открыт проект, среда разработки пропускает окно Welcome to Android Studio и открывает последний проект. В этом случае диалоговое окно Updates вызывается командой Android Studio ▸ Check for Updates... (Mac) или командой Help ▸ Check for Update... (Windows/Linux); далее продолжайте с шага 4. Список полезных советов по работе с Android Studio доступен по адресу

http://developer.android.com/sdk/installing/studio-tips.html

Настройка Android Studio для вывода номеров строк

По умолчанию Android Studio не выводит номера строк в коде, с которым вы работаете. Включите режим нумерации строк, чтобы вам было проще следить за кодом в приведенных примерах:

1. Откройте Android Studio ().
2. Когда на экране появится окно Welcome to Android Studio, щелкните на кнопке Configure, а затем на Settings. Открывается окно Default Settings. Если заставка Welcome to Android Studio не отображается, воспользуйтесь командой меню Android Studio ▸ Preferences... (Mac) или File ▸ Other Settings ▸ Default Settings... (Windows/Linux).
3. Откройте раздел Editor ▸ General и выберите категорию Appearance. Установите флажок Show line numbers и щелкните на кнопке OK.

Отключение свертки кода в Android Studio

По умолчанию в Android Studio включен режим свертки кода: фрагменты кода сокращаются в одну строку, чтобы разработчик мог сосредоточиться на других аспектах кода. Например, все директивы `import` в исходном кода Java могут быть скрыты в одной строке или целый метод прячется за одной строкой. Если вам потребуется просмотреть скрытый код, свернутые фрагменты можно развернуть. Мы отключили этот режим в своей среде разработки. Если вы захотите поступить так же, выполните действия из предыдущего раздела, а затем в категории Editor ▸ General ▸ Code Folding снимите флажок Show code folding outline.

Android 6 SDK

Все примеры книги рассчитаны на Android 6. На момент написания этой главы версия 6 включалась в комплект поставки Android Studio. С выпуском новых версий Android в комплект поставки IDE может быть включена обновленная версия, что может привести к нарушению компиляции приложений. При работе с примерами мы рекомендуем использовать Android 6. Чтобы установить другие версии платформы Android, выполните следующие действия.

1. Откройте Android Studio ().
2. В окне Welcome to Android Studio щелкните на кнопке Configure, а затем на SDK Manager. Если вместо окна Welcome to Android Studio открывается окно проекта, Android SDK Manager можно вызвать командой Tools ▸ Android ▸ SDK Manager.
3. На вкладке SDK Platforms выберите версии Android, которые нужно установить, после чего щелкните на кнопках Apply и OK. Среда разработки загружает и устанавливает дополнительные версии платформы. Также с ее помощью вы сможете своевременно обновлять установленные версии.

Создание виртуальных устройств Android (AVD)

Эмулятор Android, включенный в состав Android SDK, позволяет тестировать приложения Android на компьютере, а не на реальном устройстве Android. Такая возможность может быть полезна, если физическое устройство недоступно. При этом эмулятор может работать очень медленно, так что физическое устройство все же предпочтительнее. Кроме того, эмулятор не поддерживает некоторые возможности физических устройств, включая телефонные звонки, подключения USB, использование гарнитур и связь Bluetooth. Информация

о возможностях и ограничениях последней версии эмулятора доступна по адресу

http://developer.android.com/tools/devices/emulator.html

В разделе Using Hardware Acceleration этой страницы описаны некоторые возможности повышения быстродействия эмулятора, например использование графического процессора для ускорения вывода графики или применение Intel HAXM (Hardware Accelerated eXecution Manager) для повышения общей производительности AVD. Также существуют более производительные сторонние эмуляторы — например, Genymotion.

Прежде чем запускать приложение в эмуляторе, создайте хотя бы одно *виртуальное устройство Android* (Android Virtual Device, AVD) для Android 6. Оно определяет характеристики реального устройства, для которого должно тестироваться приложение: размер экрана (в пикселах), плотность пикселов, физический размер экрана, объем карты памяти SD для хранения данных и ряд других параметров. Чтобы протестировать приложения для нескольких устройств Android, создайте отдельное устройство AVD для каждого конкретного физического устройства. Также можно воспользоваться сервисом Google Cloud Test Lab:

https://developers.google.com/cloud-test-lab/

Этот сайт позволяет вам передать свое приложение и протестировать его на многих популярных современных Android-устройствах. По умолчанию Android Studio создает одно виртуальное устройство AVD для версии Android, включенной в поставку среды разработки. В этой книге мы используем AVD для двух эталонных устройств Android (телефон Nexus 6 и планшет Nexus 9) со стандартными версиями Android без модификаций, вносимых многими производителями оборудования. AVD удобнее создавать в Android Studio для уже открытого проекта, поэтому процесс создания AVD для Android 6 будет описан в разделе 1.9.

Настройка устройства Android для разработки

Как упоминалось ранее, на физическом устройстве приложения работают быстрее, чем в AVD. Кроме того, некоторые функции могут тестироваться только на физических устройствах. За информацией о том, как выполнить приложение на устройстве Android, обращайтесь по адресу

http://developer.android.com/tools/device.html

Если разработка ведется на платформе Microsoft Windows, вам также понадобится драйвер Windows USB для устройств Android. В некоторых случаях

могут также понадобиться USB-драйверы, предназначенные для конкретного устройства. Со списком USB-драйверов, предназначенных для различных устройств, можно ознакомиться по адресу

http://developer.android.com/tools/extras/oem-usb.html

Как получить примеры кода

Все примеры кода, рассматриваемые в книге, доступны по адресу

http://www.deitel.com/books/AndroidFP3/

Щелкните на ссылке Download Code Examples, чтобы загрузить zip-архив с примерами на ваш компьютер. В зависимости от операционной системы сделайте двойной щелчок на файле, чтобы распаковать его, или щелкните правой кнопкой мыши и выберите команду распаковки. Запомните, где именно будет сохранено распакованное содержимое архива.

О средствах разработчика Android

Если вы импортировали приложение в Android Studio и оно не компилируется, это может быть связано с обновлением Android Studio или инструментария Android. Информацию о таких проблемах можно найти в разделе вопросов/ответов Android на сайте StackOverflow:

http://stackoverflow.com/questions/tagged/android

и в сообществе Google+ Android Development:

http://bit.ly/GoogleAndroidDevelopment

Также вы можете связаться с нами по электронной почте:

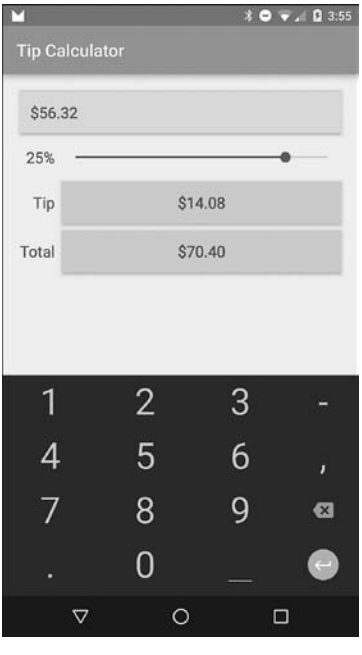
deitel@deitel.com

Итак, вы установили все необходимые программы и загрузили примеры кода, которые вам понадобятся для изучения разработки Android-приложений и написания собственных программ. Успехов!

1 Введение в Android

В этой главе…

- История Android и Android SDK
- Загрузка приложений из магазина Google Play Store
- Пакеты Android, используемые в книге для создания приложений Android
- Основные концепции объектной технологии
- Ключевые программные продукты, применяемые для разработки приложений Android, в том числе Android SDK, Java SDK и интегрированная среда разработки Android Studio
- Основная документация по Android
- Тестирование приложения Android в Android Studio
- Характеристики профессиональных приложений Android



1.1. Введение

Добро пожаловать в мир разработки приложений Android! Мы надеемся, что книга «Android для разработчиков» покажется вам познавательной и интересной и вы не пожалеете о потраченном времени.

Материал книги ориентирован на *программистов Java*. В книге используются только завершенные рабочие приложения, поэтому, даже не зная Java, но имея опыт объектно-ориентированного программирования на другом языке (C#/, Objective-C/Cocoa либо C++ с библиотеками классов), вы сможете быстро освоить излагаемый в книге материал, а также изучить Java и объектно-ориентированное программирование в стиле Java в процессе изучения разработки приложений Android.

Все новые технологии в книге рассматриваются в контексте разработки завершенных рабочих приложений Android, каждое из которых описано в отдельной главе. Мы описываем приложение и *тестируем* его. Затем вкратце описываются ключевые технологии *Android Studio* (интегрированной среды разработки), Java и *Android SDK* (Software Development Kit), используемые для создания приложения. Если разрабатываемые приложения используют графический интерфейс, описывается процесс его *визуального* проектирования в Android Studio. Далее приводятся листинги исходного кода с нумерацией строк и выделением ключевых фрагментов кода. Результаты выполнения кода проиллюстрированы одним-двумя экранными снимками. Код подробно анализируется, причем особое внимание уделяется новым концепциям программирования, использованным в приложении. Исходный код всех приложений, рассматриваемых в книге, может быть загружен с веб-сайта *http://www.deitel.com/books/AndroidFP3/*.

1.2. Android — мировой лидер в области мобильных операционных систем

Стремительный рост продаж устройств на базе Android открывает выдающиеся возможности перед разработчиками приложений Android.

- Первое поколение телефонов на базе Android было выпущено в октябре 2008 года. По данным за июнь 2015 года Android принадлежало 82,8% глобального рынка смартфонов — по сравнению с 13,9% у Apple и 2,6% у Microsoft[1].

[1] *http://www.idc.com/prodserv/smartphone-os-market-share.jsp*.

- Загрузки приложений из Google Play исчислялись миллиардами, а в 2014 году было куплено более миллиарда устройств на базе Android[1].
- По данным PC World, в 2014 году было куплено около 230 миллионов планшетных устройств, из которых 67,3% составляли устройства на базе Android — по сравнению с 27,6% у iOS и 5,1% у планшетов на базе Microsoft Windows[2].
- На платформе Android сейчас работают смартфоны, планшеты, электронные книги, роботы, реактивные двигатели, спутники NASA, игровые приставки, холодильники, телевизоры, камеры, медицинские устройства, «умные часы», автомобильные информационные системы (для управления радио, GPS, телефонами, термостатами и т. д.) и многие другие устройства[3].
- По последним прогнозам, доходы от мобильных приложений (по всем мобильным платформам) к 2019 году достигнут 99 миллиардов долларов[4].

1.3. Особенности Android

Одно из главных преимуществ платформы Android — ее открытость. Операционная система Android построена на основе *открытого исходного кода* и находится в свободном распространении. Это позволяет разработчикам получить доступ к исходному коду Android и понять, каким образом реализованы свойства и функции приложений. Любой пользователь может принять участие в совершенствовании операционной системы Android. Для этого достаточно отправить отчет об обнаруженных ошибках (*http://source.android.com/source/report-bugs.html*) либо принять участие в одной из дискуссионных групп Open Source Project (*http://source.android.com/ community/index.html*). В Интернете доступны различные приложения Android с открытым исходным кодом, предлагаемые компанией Google и рядом других производителей (табл. 1.1). В табл. 1.2 показано, где можно получить исходный код Android, узнать об идеологии, заложенной в основу операционной системы с открытым кодом, и получить лицензионную информацию.

Открытость платформы способствует быстрому обновлению. В отличие от закрытой системы iOS компании Apple, доступной только на устройствах Apple,

---

[1] *http://www.cnet.com/news/android-shipments-exceed-1-billion-for-first-time-in-2014/.*

[2] *http://www.pcworld.com/article/2896196/windows-forecast-to-gradually-grab-tablet-market-share-from-ios-and-android.html.*

[3] *http://www.businessweek.com/articles/2013-05-29/behind-the-internet-of-things-is-android-and-its-everywhere.*

[4] *http://www.telecompetitor.com/mobile-app-forecast-calls-for-revenue-of-99-billion-by-2019/.*

система Android доступна на устройствах десятков производителей оборудования (OEM, Original Equipment Manufacturer) и телекоммуникационных компаний по всему миру. Все они конкурируют между собой, что идет на пользу конечному потребителю.

Таблица 1.1. Ресурсы приложений и библиотек Android с открытым кодом

URL	Описание
http://en.wikipedia.org/wiki/List_of_open_source_Android_applications	Обширный список приложений с открытым исходным кодом, разбитых по категориям (игры, коммуникации, эмуляторы и пр.)
http://developer.android.com/tools/samples/index.html	Примеры приложений Google для платформы Android; включает свыше 100 приложений и игр, демонстрирующих различные возможности Android
http://github.com	Репозиторий GitHub позволяет распространять ваши приложения и исходный код, а также участвовать в проектах с открытым кодом других разработчиков
http://f-droid.org	Сотни бесплатных и распространяемых с открытым кодом приложений Android
http://www.openintents.org	Библиотеки с открытым кодом, расширяющие функциональность приложений
http://www.stackoverflow.com	Сайт Stack Overflow предназначен для публикации вопросов и ответов для программистов. Пользователи могут голосовать, и лучшие ответы занимают первые места

Таблица 1.2. Ресурсы с исходным кодом операционной системы Android

Название	URL
Исходный код Android	http://source.android.com/source/downloading.html
Лицензии	http://source.android.com/source/licenses.html
Списки FAQ	http://source.android.com/source/faqs.html

Java

При разработке приложений Android используется Java — один из наиболее распространенных языков программирования. Использование Java стало логичным выбором для платформы Android, потому что это мощный, свободный

и открытый язык, известный миллионам разработчиков. Опытные программисты Java могут быстро освоить Android-программирование, используя интерфейсы Google Android API (Application Programming Interface) и другие разработки независимых фирм.

Язык Java является объектно-ориентированным, предоставляет разработчикам доступ к мощным библиотекам классов, ускоряющих разработку приложений. Программирование графического интерфейса пользователя *управляется событиями* — в этой книге приведены примеры приложений, которые реагируют на инициируемые пользователями *события*, такие как *касания экрана*. Помимо непосредственного написания кода приложений можно воспользоваться средами разработки Eclipse и Android Studio, позволяющими собирать графический интерфейс из готовых объектов, таких как кнопки и текстовые поля, перетаскивая их в определенные места экрана, добавляя подписи и изменяя их размеры. Эти среды разработки позволяют быстро и удобно создавать, тестировать и отлаживать приложения Android.

Мультисенсорный экран

Многие современные смартфоны Android сочетают в себе функции мобильных телефонов, интернет-клиентов, MP3-плееров, игровых консолей, цифровых фотоаппаратов и многого другого. Эти портативные устройства оборудованы полноцветными *мультисенсорными экранами*. Простые прикосновения пальцев позволяют легко переключаться между использованием телефона, запуском приложений, воспроизведением музыки, просмотром веб-страниц и т. д. На экране может отображаться клавиатура для ввода электронной почты и текстовых сообщений, а также ввода данных в приложениях (некоторые устройства Android также оснащаются физическими клавиатурами).

Жесты

Мультисенсорный экран позволяет управлять устройством с помощью касаний и *жестов*, как показано в табл. 1.3.

Таблица 1.3. Жесты, применяемые на устройствах Android

Название	Физическое действие	Применение
Касание	Коснитесь один раз экрана	Открытие приложения, «нажатие» кнопки или элемента меню
Двойное касание	Дважды коснитесь экрана	Увеличение или уменьшение масштаба просмотра изображений, карт Google Maps и веб-страниц. Повторное двойное касание возвращает к прежнему масштабу

Название	Физическое действие	Применение
Длинное нажатие	Нажмите выбранную область экрана и удерживайте палец в этой позиции	Выбор элементов в списке
Смахивание	Нажмите, быстро проведите пальцем вдоль экрана в нужном направлении, после чего отпустите палец	Прокрутка серии объектов (например, галереи фотографий). Смахивание автоматически останавливается у следующего объекта
Перетаскивание	Нажмите пальцем и перетащите его вдоль экрана	Перемещение объектов или значков либо точная прокрутка веб-страницы или списка
Масштабирование двумя пальцами	Коснитесь экрана двумя пальцами, а потом сведите или разведите их для изменения масштаба	Увеличение или уменьшение масштаба просмотра экрана (увеличение или уменьшение текста и рисунков)

Встроенные приложения

В комплект поставки устройств Android входят различные стандартные приложения, набор которых зависит от устройства, производителя или оператора мобильной связи. Обычно это приложения `Телефон` (Phone), `Контакты` (People), `Сообщения` (Messenger), `Браузер` (Browser), `Калькулятор` (Calculator), `Календарь` (Calendar), `Часы` (Clock) и `Фото` (Photos).

Веб-сервисы

Веб-сервисы (web services) представляют собой программные компоненты, хранящиеся на одном компьютере, к которым могут обращаться приложения (или другие программные компоненты) с другого компьютера по Интернету. На базе веб-сервисов могут создаваться *мэшапы* (mashups), ускоряющие разработку приложений путем комбинирования веб-сервисов, используемых в различных организациях, с различными типами вводимой информации. Например, сайт 100 Destinations (*www.100destinations.co.uk*) объединяет фотографии и публикации в «Твиттере» с картами Google Maps, чтобы вы могли познакомиться с разными странами по фотографиям других пользователей.

На сайте Programmableweb (*http://www.programmableweb.com/*) размещен каталог 14 000 API и мэшапов, а также руководства и примеры кода по самостоятельному созданию мэшапов. В табл. 1.4 перечислены некоторые популярные веб-сервисы. В главе 7 мы воспользуемся веб-сервисами OpenWeatherMap.org.

Таблица 1.4. Некоторые популярные веб-сервисы (www.programmableweb.com/category/all/apis)

Веб-сервисы	Применение
Google Maps	Картографические службы
Twitter	Микроблоги
YouTube	Поиск видео
Facebook	Социальные сети
Instagram	Публикация фотографий в Интернете
Foursquare	Мобильная регистрация местоположения
LinkedIn	Социальные бизнес-сети
Netflix	Аренда фильмов
eBay	Интернет-аукционы
Wikipedia	Коллективная энциклопедия
PayPal	Платежи
Amazon eCommerce	Покупка книг и множества других продуктов
Salesforce.com	Управление отношениями с клиентами и партнерами
Skype	Интернет-телефония
Microsoft Bing	Поисковая система Bing
Flickr	Публикация фотографий в Интернете
Zillow	Цены на недвижимость
Yahoo Search	Поиск
WeatherBug	Прогноз погоды

1.4. Операционная система Android

Операционная система Android была разработана компанией Android, Inc., которая в 2005 году была приобретена компанией Google. В ноябре 2007-го был сформирован консорциум Open Handset Alliance™ (*http://www.openhandsetalliance.com/oha_members.html*). В задачи этого консорциума входит разработка, сопровождение и развитие Android, внедрение инноваций в мобильных технологиях, а также повышение удобства работы с устройствами Android при одновременном снижении затрат.

В этом разделе рассматривается эволюция операционной системы Android, представлены ее основные версии и их ключевая функциональность. На многих устройствах продолжают использоваться старые версии Android, поэтому разработчику будет полезно знать о том, в какой версии появлялись те или иные новые возможности.

Имена версий Android

Каждой новой версии Android присваивалось название, соответствующее названию какого-либо десерта (табл. 1.5).

Таблица 1.5. Номера версий Android и их названия

Версия Android	Название
Android 1.5	Cupcake
Android 1.6	Donut
Android 2.0–2.1	Eclair
Android 2.2	Froyo
Android 2.3	Gingerbread
Android 3.0–3.2	Honeycomb
Android 4.0	Ice Cream Sandwich
Android 4.1–4.3	Jelly Bean
Android 4.4	KitKat
Android 5.0–5.1	Lollipop
Android 6.0	Marshmallow

1.4.1. Android 2.2 (Froyo)

Версия Android 2.2, которая также называется Froyo (замороженный йогурт), вышла в мае 2010 года. В ней появилась поддержка внешней памяти, позволявшая хранить приложения на внешнем устройстве (вместо внутренней памяти устройства Android). С помощью службы *C2DM* (Android Cloud to Device Messaging, обмен сообщениями с устройствами через облако) разработчики приложений могут использовать программы и данные, хранящиеся в «облаке» — то есть на удаленных компьютерах (серверах) в Интернете, вместо того чтобы хранить их на настольном компьютере, ноутбуке или мобильном устройстве. Облачные технологии предоставляют гибкие средства

наращивания или сокращения вычислительных ресурсов под конкретные потребности в любой момент времени. Это делает их более экономически эффективными по сравнению с приобретением дорогостоящего оборудования, гарантирующего достаточную вычислительную мощность и емкость памяти для периодических пиковых нагрузок. Android C2DM позволяет разработчикам приложений пересылать данные со своих серверов на приложения, установленные на устройствах Android, даже если эти приложения в данный момент *не активны*. Сервер оповещает приложения, предлагая им подключиться к нему для приема обновления или пользовательских данных[1]. Сейчас технология C2DM считается устаревшей — на смену ей пришла технология Google Cloud Messaging.

Информация о других новых возможностях Android 2.2 — графических средствах OpenGL ES 2.0, медиафреймворках и т. д. — доступна по адресу

http://developer.android.com/about/versions/android-2.2-highlights.html

1.4.2. Android 2.3 (Gingerbread)

Версия Android 2.3 Gingerbread («имбирный пряник»), появившаяся в 2010 году, предлагает новые возможности для пользователя — более удобную клавиатуру, улучшенные навигационные функции, более эффективное использование батареи и ряд других преимуществ. Также в ней добавились новые средства разработчика для передачи данных (например, технологии, упрощающие телефонные звонки и ответы на них в приложениях), мультимедийные технологии (новые API для работы со звуком и графикой) и игровые средства (повышение быстродействия и новые датчики — например, гироскоп для обработки перемещений в пространстве).

Одним из самых значительных нововведений Android 2.3 стала поддержка *NFC* (Near-Field Communication) — технологии беспроводной высокочастотной связи малого радиуса действия для обмена данными между устройствами, находящимися на расстоянии нескольких сантиметров. Уровень поддержки и функциональность NFC зависит от конкретного устройства. NFC может использоваться для электронных платежей (например, когда вы прикасаетесь устройством Android с поддержкой NFS к платежному блоку на торговом автомате), передачи данных (контактов, фотографий и т. д.), сопряжении устройств и аксессуаров и т. д. Полный список средств разработчика Android 2.3 доступен по адресу

http://developer.android.com/about/versions/android-2.3-highlights.html

[1] *http://code.google.com/android/c2dm/*.

1.4.3. Android 3.0–3.2 (Honeycomb)

В версии Android 3.0 Honeycomb («медовые соты») появились усовершенствования пользовательского интерфейса, предназначенные для устройств с большим экраном (то есть планшетов): усовершенствованная клавиатура для более эффективного ввода данных, трехмерный пользовательский интерфейс, упрощение переходов между экранами в приложении и ряд других улучшений. Некоторые новые средства разработчика Android 3.0:

- фрагменты, описывающие части пользовательского интерфейса приложения, которые могут объединяться в один экран или использоваться на разных экранах;
- постоянная панель действий (action bar) в верхней части экрана, предоставляющая дополнительные средства для взаимодействия с приложениями;
- возможность добавления макетов для большого экрана к существующим приложениям, рассчитанным на малые экраны, чтобы оптимизировать приложение для разных вариантов размера экрана;
- привлекательный и более функциональный пользовательский интерфейс «Holo», имитирующий «голографический» эффект;
- новые средства анимации;
- усовершенствованные графические и мультимедийные возможности;
- возможность использования многоядерных процессоров для повышения быстродействия;
- улучшенная поддержка Bluetooth (например, возможность проверки подключенных устройств — гарнитуры, клавиатуры и т. д.);
- фреймворк для анимации пользовательского интерфейса или графических объектов.

Список средств Android 3.0 для пользователей и разработчиков, а также платформенных технологий доступен по адресу

http://developer.android.com/about/versions/android-3.0-highlights.html

1.4.4. Android Ice Cream Sandwich

Версия Android 4.0 Ice Cream Sandwich («мороженое с вафлями»), вышедшая в 2011 году, объединила Android 2.3 (Gingerbread) и Android 3.0 (Honeycomb) в одну операционную систему, предназначенную для использования на всех устройствах Android. Эта версия Android позволяет включить функции,

поддерживаемые в версии Honeycomb и ранее доступные только для планшетов («голографический» интерфейс пользователя, новый лаунчер и т. д.), в приложения для смартфонов. Так обеспечивается простое масштабирование приложений, позволяющее использовать их на различных устройствах. В версии Ice Cream Sandwich также появился ряд новых API для улучшения обмена данными между устройствами, технологий для пользователей с ограниченными возможностями (например, с ослабленным зрением), поддержки социальных сетей и т. д. (табл. 1.6). За полным списком API для Android 4.0 обращайтесь по адресу

http://developer.android.com/about/versions/android-4.0.html

Таблица 1.6. Некоторые средства разработчика Android Ice Cream Sandwich (http://developer.android.com/about/versions/android-4.0.html)

Функция	Описание
Face detection (Распознавание лиц)	С помощью камеры совместимые устройства могут определять положение глаз, носа и рта пользователя. Камера может также отслеживать направление взгляда пользователя, позволяя создавать приложения, которые изменяют перспективу в зависимости от направления взгляда пользователя
Virtual camera operator (Виртуальный оператор камеры)	В процессе видеосъемки камера автоматически фокусируется на говорящем человеке
Android Beam	Технология Android Beam позволяет передавать контент (контакты, фото, видео) между двумя соприкасающимися устройствами Android
Wi-Fi Direct	Wi-Fi P2P (Peer-to-Peer) API позволяет связать несколько устройств Android по Wi-Fi. В этом случае беспроводная связь между устройствами может осуществляться на большем расстоянии, чем при использовании Bluetooth
Social API	Получение доступа и обмен контактными данными между социальными сетями и приложениями (с разрешения пользователя)
Calendar API	Добавление событий, обмен событиями между приложениями, управление сигналами, участниками и т. д.
Accessibility API	Новые вспомогательные API повышают доступность ваших приложений для людей с ограниченными возможностями (дефектами зрения и т. д.). Режим «Explore-by-touch» позволяет слабовидящим пользователям прикоснуться к любой точке экрана и прослушать голосовое описание контента

Функция	Описание
Android@Home framework (Фреймворк Android@Home)	Обеспечивает создание приложений Android, управляющих бытовыми устройствами в доме пользователя: термостатами, системами полива, освещения и т. д.
Bluetooth Health Devices	Создание приложений, взаимодействующих с медицинскими устройствами с поддержкой Bluetooth: весами, пульсометрами и т. д.

1.4.5. Android 4.1–4.3 (Jelly Bean)

В версии Android Jelly Bean («жевательная конфета»), выпущенной в 2012 году, центральное место занимают внутренние усовершенствования: улучшенная производительность, доступность, поддержка региональных стандартов и т. д. Также следует отметить расширенную поддержку Bluetooth (поддержка Bluetooth LE появилась в Android 4.3), поддержку внешних экранов, усовершенствованные средства безопасности, улучшенное оформление (например, виджеты приложений с изменяемыми размерами и крупные оповещения), оптимизированные средства позиционирования и работы с датчиками, улучшенные мультимедийные возможности (аудио/видео) и функции более плавного переключения между приложениями и экранами (табл. 1.7). Кроме того, компания Google представила новые API, которые разрабатываются отдельно от версий платформы Android:

- Google Cloud Messaging — кроссплатформенное решение для доставки сообщений на устройства;
- Google Play Services — набор API для интеграции функциональности Google в приложения.

За полным списком возможностей Jelly Bean обращайтесь по адресу

http://developer.android.com/about/versions/jelly-bean.html

Таблица 1.7. Некоторые возможности Android Jelly Bean (http://developer.android.com/about/versions/jelly-bean.html)

Функция	Описание
Android Beam	Технология Android Beam была расширена для передачи данных Bluetooth (помимо NFC)
Виджеты блокировки экрана	Создание виджетов, которые отображаются на экране пользователя во время блокировки устройства, или изменение существующих виджетов главного экрана, чтобы они тоже были видны во время блокировки

Таблица 1.7 (*окончание*)

Функция	Описание
Photo Sphere	API для работы с новыми панорамными фотографиями позволяют создавать обзорные фото, сходные с теми, которые используются в режиме Google Maps Street View
Daydreams	Интерактивные заставки, которые активизируются при зарядке или нахождении устройства в док-станции. Заставки Daydreams способны воспроизводить аудио и видео, а также реагировать на действия пользователя
Поддержка языков	Новые функции, ориентированные на пользователей из других стран (двусторонний вывод текста — слева направо и справа налево, клавиатуры для других языков, дополнительные раскладки и т. д.)
Средства разработчика	Новые средства диагностики и отладки, например возможности отправки отчетов об ошибках, содержащие экранные снимки и информацию о состоянии устройства

1.4.6. Android 4.4 (KitKat)

Версия Android 4.4 KitKat, выпущенная в октябре 2013 года, включает ряд усовершенствований, обеспечивающих работу операционной системы на любых устройствах Android, включая старые устройства с ограниченной памятью, особенно популярные в развивающихся странах[1].

Переход большего количества пользователей на KitKat сократит «фрагментацию» версий Android на рынке. Фрагментация создавала проблемы для разработчиков, которые были вынуждены проектировать приложения для разных версий операционной системы или ограничивать круг потенциальных пользователей, разрабатывая приложение для конкретной версии операционной системы.

Android KitKat также включает усовершенствования безопасности и доступности, улучшенные графические и мультимедийные возможности, средства анализа памяти и т. д. В табл. 1.8 перечислены некоторые из ключевых новых возможностей KitKat. За полным списком обращайтесь по адресу

http://developer.android.com/about/versions/kitkat.html

[1] *http://techcrunch.com/2013/10/31/android-4-4-kitkat-google/*.

Таблица 1.8. Некоторые возможности Android KitKat (http://developer.android.com/about/versions/kitkat.html)

Функция	Описание
Режим погружения (Immersive mode)	Панель состояния у верхнего края экрана и кнопки меню у нижнего края можно скрыть, чтобы ваши приложения могли занимать большую часть экрана. Пользователь может вызвать панель состояния, смахнув вниз от верхнего края экрана, а системную панель (с кнопками Back, Home и Recent Apps) — смахиванием вверх от нижнего края
Инфраструктура печати	Встраивание функциональности печати в приложения, включая поиск доступных принтеров по Wi-Fi или в облаке, выбор размера бумаги и печатаемых страниц
Доступ к документам	Создание провайдеров хранения документов, позволяющих просматривать, создавать и редактировать файлы (например, документы и изображения) в разных приложениях
Сервис SMS	Создание приложений SMS (Short Message Service) и MMS (Multimedia Messaging Service) с использованием нового поставщика SMS и API. Теперь пользователи могут выбирать приложение передачи сообщений по умолчанию
Переходы	Новая инфраструктура упрощает создание переходных анимаций
Запись с экрана	Запись видеороликов работающих приложений для создания интерактивных руководств и маркетинговых материалов
Улучшения доступности	Captioning Manager API позволяет приложению проверить пользовательские настройки субтитров (язык, оформление текста и т. д.)
Chromium WebView	Поддержка новейших стандартов вывода веб-контента, включая HTML5, CSS3 и ускоренную версию JavaScript
Датчик и счетчик шагов	Создание приложений, которые определяют, идет ли пользователь, бежит или поднимается по лестнице, и подсчитывают количество сделанных шагов
Host Card Emulator (HCE)	Технология HCE позволяет любому приложению выполнять безопасные транзакции NFC (например, мобильные платежи) без обязательного присутствия элемента безопасности на SIM-карте под управлением оператора мобильной связи

1.4.7. Android 5.0 и 5.1 (Lollipop)

Версия Android Lollipop, выпущенная в ноябре 2014 года, стала крупным обновлением с тысячами усовершенствований API для телефонов и планшетов. Новые возможности этой версии позволяют разработчикам создавать

приложения для носимых устройств (например, «умных часов»), телевизоров и автомобилей. Одним из самых серьезных изменений стал *материальный дизайн* — полностью переработанная концепция пользовательского интерфейса (также используемая в веб-приложениях Google). Среди других новшеств также стоит выделить новую исполнительную среду Android, улучшения в системе оповещений (теперь пользователь может взаимодействовать с оповещением, не покидая приложения), улучшения в работе с сетями (Bluetooth, Wi-Fi, сотовые сети и NFC), высокопроизводительную графику (OpenGL ES 3.1 и Android Extension Pack), расширенную поддержку аудио (запись, многоканальное микширование, воспроизведение и поддержка периферийных устройств USB), расширенные средства работы с камерой, поддержку новых датчиков, расширенные средства доступности, поддержку использования нескольких SIM-карт и т. д. В табл. 1.9 перечислены некоторые ключевые возможности Lollipop. За более полной информацией обращайтесь по адресам:

http://developer.android.com/about/versions/lollipop.html

http://developer.android.com/about/versions/android-5.0.html

http://developer.android.com/about/versions/android-5.1.html

Таблица 1.9. Некоторые возможности Android Lollipop (http://developer.android.com/about/versions/lollipop.html)

Функция	Описание
Материальный дизайн	Новый вариант дизайна приложений Android и веб-приложений от компании Google стал ключевым нововведением Lollipop. Материальный дизайн позволяет создавать приложения с переходами, тенями и имитацией глубины, расширенными возможностями настройки и т. д. За дополнительной информацией обращайтесь по адресу https://www.google.com/design/spec/material-design/introduction.html
Исполнительная среда ART	Компания Google заменила исходную исполнительную среду Android новой исполнительной средой ART, использующей сочетание интерпретации, опережающей компиляции AOT (Ahead-Of-Time) и компиляции JIT (Just-In-Time) для повышения быстродействия
Параллельные документы и активности на экране Recent Apps	Теперь приложение может выбрать способ отображения множественных активностей и документов на экране Recent Apps. Например, если пользователь открыл несколько вкладок в браузере или несколько документов в текстовом редакторе, при нажатии кнопки Recent Apps (▣) каждая вкладка может отображаться в виде отдельного элемента, который может быть выбран пользователем

Функция	Описание
Сохранение и передача содержимого экрана	Теперь приложение может сохранить содержимое экрана устройства и передать его другим пользователям по сети
Project Volta	Функции энергосбережения, включая новый планировщик заданий, способный выполнять асинхронные задачи во время зарядки устройства, при неограниченном подключении к сети (например, Wi-Fi вместо сотовой связи) или во время бездействия

1.4.8. Android 6 (Marshmallow)

Версия Android Marshmallow, выпущенная в сентябре 2015 года, является последней версией Android на момент написания книги. В нее были включены такие новые возможности, как Now on Tap (для получения информации Google Now в контексте приложения), Doze и App Standby (для экономии заряда батареи), новая модель разрешений, упрощающая установку приложений, аутентификация по отпечатку пальца, улучшенная защита данных, улучшенная поддержка выделения текста, поддержка экранов 4K, новые возможности аудио и видео, новые средства работы с камерой (API режима фонарика и повторной переработки изображений) и другие функции.

Некоторые ключевые возможности Lollipop перечислены в табл. 1.10. За полным списком обращайтесь по адресу

http://developer.android.com/about/versions/marshmallow/android-6.0-changes.html

Таблица 1.10. Некоторые возможности Android Marshmallow (http://developer.android.com/about/versions/marshmallow/android-6.0-changes.html)

Функция	Описание
Doze	При помощи датчиков и программных компонентов Android определяет, когда устройство остается неподвижным в течение долгого времени (например, если вы положили его на столик на ночь), и откладывает выполнение фоновых процессов, разряжающих батарею
App Standby	Если пользователь открыл приложение, но не работал с ним в последнее время, Android откладывает выполнение фоновых сетевых операций
Now on Tap	Если нажать и удерживать кнопку Home в любом приложении, функция Google Now анализирует содержимое экрана и представляет актуальную информацию в форме карточек. Например, в текстовом сообщении, в котором обсуждается фильм,

Таблица 1.10 (*окончание*)

Функция	Описание
	появляется карточка с информацией о фильме. В сообщении, в котором упоминается название ресторана, появится карточка с рейтингом, картой и номером телефона, и т. д.
Новая модель разрешений	До выхода Android 6.0 пользователь должен был задать все разрешения, которые могли понадобиться приложению, во время установки. Это приводило к тому, что многие пользователи отказывались от установки некоторых приложений. С новой моделью приложение устанавливается, не запрашивая никаких разрешений. Вместо этого пользователь должен предоставить необходимое разрешение только при первом использовании соответствующей функции
Аутентификация по отпечатку пальца	Для устройств, оснащенных функцией считывания отпечатков, приложения теперь могут проводить проверку пользователей по отпечатку пальца
Связывание приложений	Разработчик может связывать приложения с веб-доменами и создавать веб-ссылки, запускающие конкретные приложения того же разработчика
Автоматическое резервное копирование	Android может автоматически архивировать и восстанавливать данные приложения
Direct Share	В приложении можно определять объекты Direct Share, при помощи которых пользователи могут обмениваться данными с другими приложениями, не покидая вашего приложения
Voice Interaction API	Поддержка голосового управления
Поддержка стилуса Bluetooth	Приложение может реагировать на операции, чувствительные к давлению, от стилуса Bluetooth, например сильное нажатие стилусом на экран может создать более толстую линию

1.5. Загрузка приложений из Google Play

На время написания этой книги в Google Play было доступно более 1,6 миллиона приложений, и это число быстро росло[1]. В табл. 1.11 перечислены некоторые популярные платные и бесплатные приложения из разных категорий. Приложения можно загрузить из приложения `Play Store`, установленного на устройстве. Вы также можете войти в учетную запись Google Play по адресу *http://play.google.com*, а затем выбрать устройство Android, на котором должно быть установлено

[1] *http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/*.

приложение. Приложение загружается на устройство через подключение Wi-Fi или 3G/4G. В главе 10 будут рассмотрены другие магазины, предоставляющие приложения (бесплатно или за деньги), цены и т. д.

Таблица 1.11. Некоторые популярные приложения Android в Google Play

Категория	Некоторые популярные приложения в этой категории
Книги и справочники	WolframAlpha, Dictionary.com, Audible for Android, Kindle
Бизнес	Polaris Office, OfficeSuite 8, QuickBooks Online, PayPal Here
Связь	Snapchat, LinkedIn, Pinterest, Instagram, WeChat, Line
Образование	Google Classroom, Star Tracker, Sight Words, Math Tricks
Развлечения	Showtime Anytime, History Channel, Discovery Channel
Финансы	PayPal, Credit Karma, Google Wallet, Chase Mobile
Игры	Pac-Man 256, Angry Birds 2, Fruit Ninja, Tetris, Solitaire
Здоровье и фитнес	RunKeeper, ViewRanger GPS, Calorie Counter
Стиль жизни	Assistant, Horoscope, Food Network, Starbucks
Живые обои	Facebook, Next Launcher 3D Shell, Weather Live
Видео и мультимедиа	VHS Camera Recorder, VivaVideo Pro, musical.ly, GIF Keyboard
Медицина	Feed Baby Pro, CareZone, FollowMyHealth, Essential Anatomy
Музыка и аудио	SoundCloud, Spotify, Beats Music, Pandora, iHeartRadio
Новости и журналы	BBC News, CBS News, NPR News, Reuters, NBC News
Фотография	Google Camera, Instagram, Retrica, GoPro App, Pencil Sketch
Производительность	Pocket, Wunderlist, Microsoft Word, Google Docs, SwiftKey
Покупки	Zappos, Groupon, JackThreads, Fancy, Etsy, Home Depot
Социальные сети	Snapchat, Instagram, Meetup, textPlus, Pinterest, Tumblr
Спорт	Fox Sports, theScore, NBA 2015–16, ESPN, CBS Sports
Инструменты	CM Security Antivirus, Clean Master, Google Translate
Транспорт	Uber, Lyft, MarrineTraffic, BringGo, DigiHUD Speedometer
Путешествия	Priceline, Google Earth, Eat24, GasBuddy, Hotels.com
Погода	AccuWeather, Weather Underground, Yahoo Weather
Виджеты	Facebook, Pandora, Pocket Casts, Tasker, Weather Timeline

1.6. Пакеты

В Android используется целая коллекция *пакетов* — групп взаимосвязанных готовых классов. Некоторые из пакетов относятся к специфике Android, другие относятся к специфике Java и Google. Пакеты обеспечивают удобный доступ к функциям операционной системы Android, а также включение этих функций в приложения. Они упрощают создание Android-приложений, использующих особенности дизайна и соответствующих стилевым рекомендациям Android (*http://developer.android.com/design/index.html*). В табл. 1.12 перечислены пакеты, которые будут рассмотрены в книге. Полный перечень пакетов Android доступен по адресу *developer.android.com/reference/packages.html*.

Некоторые пакеты, используемые нами в работе, предоставляются *библиотеками поддержки* Android для использования новых возможностей Android в приложениях, работающих на более старых платформах. Сводка ключевых возможностей библиотек поддержки Android доступна по адресу

https://developer.android.com/tools/support-library/features.html

Таблица 1.12. Пакеты Android и Java, используемые в книге (с указанием главы, в которой они впервые встречаются)

Пакет	Описание
android.animation	Классы анимации свойств (приложение Flag Quiz — в главе 4, приложение Doodlz — в главе 5)
android.app	Включает классы высокого уровня в модели приложения Android (приложение Flag Quiz — в главе 4, приложение Doodlz — в главе 5)
android.content	Добавление и публикация данных на устройстве (приложение Cannon Game — в главе 6)
android.content.res	Классы, предназначенные для обеспечения доступа к ресурсам приложения (например, к медиафайлам, цветам, рисункам и прочим ресурсам), а также к информации о конфигурации устройства, влияющей на поведение приложения (приложение Flag Quiz — в главе 4)
android.database	Обработка данных, возвращаемых провайдером контента (приложение Address Book — в главе 9)
android.database.sqlite	Управление базами данных SQLite для частных баз данных (приложение Address Book — в главе 9)
android.graphics	Графические инструменты, применяемые для рисования на экране (приложение Flag Quiz в главе 4, приложение Doodlz — в главе 5)

Пакет	Описание
android.graphics.drawable	Классы для элементов, используемых только для отображения на экране: градиенты и т. д. (приложение Flag Quiz — в главе 4)
android.hardware	Поддержка аппаратного обеспечения устройств (приложение Doodlz — в главе 5)
android.media	Классы, предназначенные для работы с медиаинтерфейсами аудио и видео (приложение Cannon Game — в главе 6)
android.net	Классы для работы с сетью (приложение Twitter® Searches — в главе 8)
android.os	Сервис операционной системы (приложение Tip Calculator — в главе 3)
android.preference	Работа с пользовательскими настройками приложения (приложение Flag Quiz — в главе 4)
android.provider	Работа с провайдерами контента Android (приложение Doodlz — в главе 5)
android.support.design.widget	Классы библиотеки Android Design Support Library, обеспечивающие использование новых возможностей графического интерфейса на старых платформах Android (приложение Weather Viewer — в главе 7)
android.support.v4.print	Часть библиотеки Android Support Library v4 для платформ с уровнями API 4 и выше. Включает функциональность использования инфраструктуры печати Android 4.4 (приложение Doodlz — в главе 5)
android.support.v7.app	Часть библиотеки Android Support Library v7 для платформ с уровнями API 7 и выше. Включает такие компоненты совместимости, как панели приложений (ранее — панели действий) (приложение Doodlz — в главе 5)
android.support.v7.widget	Часть библиотеки Android Support Library v7 для платформ с уровнями API 7 и выше. Включает макеты и компоненты графического интерфейса (приложение Weather Viewer — в главе 7)
android.text	Вывод текста и отслеживание изменений (приложение Tip Calculator — в главе 3)
android.util	Вспомогательные методы и средства работы с XML (приложение Flag Quiz — в главе 4)
android.widget	Классы пользовательского интерфейса, предназначенные для работы с виджетами (приложение Tip Calculator — в главе 3)
android.view	Классы пользовательского интерфейса, предназначенные для взаимодействия с пользователем и построения макетов (приложение Flag Quiz — в главе 4)

1.7. Android Software Development Kit (SDK)

В состав Android SDK включены инструменты, необходимые для создания Android-приложений. Android SDK устанавливается вместе с Android Studio. За подробностями о загрузке программ, необходимых для разработки Android-приложений, включая Java SE 7 и Android Studio, обращайтесь к разделу «Подготовка».

Android Studio

Среда Android Studio[1], анонсированная в 2013 году на конференции разработчиков Google, в настоящее время считается основной средой разработки Android-приложений. В эту среду разработки входят:

- конструктор графических интерфейсов;
- редактор кода с поддержкой цветового выделения синтаксиса и нумерации строк;
- автоматические отступы и автозавершение (то есть подсказки при вводе кода);
- отладчик;
- система контроля версий;
- поддержка рефакторинга

и многое другое.

Эмулятор Android

Эмулятор Android, включенный в состав Android SDK, позволяет смоделировать среду для запуска приложений Android под управлением Windows, Mac OS X либо Linux без физического устройства. Эмулятор отображает вполне реалистичное окно интерфейса пользователя Android. Он особенно полезен, если у разработчика нет доступа к устройствам Android для прямого тестирования. Безусловно, перед отправкой в Google Play приложения желательно протестировать на различных Android-устройствах.

Перед запуском приложения на выполнение следует создать *AVD* (Android Virtual Device, виртуальное устройство Android). Это устройство определяет характеристики реального устройства Android, на котором должно тестироваться приложение: аппаратное обеспечение, системный образ, размер экрана,

[1] Среда Android Studio создана на базе Java-среды JetBrains IntelliJ IDEA (*http://www.jetbrains.com/idea/*).

хранилище данных и ряд других характеристик. Если вы намереваетесь тестировать приложения для нескольких устройств Android, необходимо создать отдельные AVD для каждого уникального физического устройства или воспользоваться сервисом Google Cloud Test Lab (*https://developers.google.com/cloud-test-lab*), позволяющим провести тестирование на многих устройствах.

В эмуляторе можно имитировать большинство жестов Android (табл. 1.13) и элементов управления (табл. 1.14), используя клавиатуру и мышь компьютера. Конечно, возможности воспроизведения жестов с помощью эмулятора несколько ограничены, поскольку компьютер не в состоянии смоделировать все аппаратные функции Android. Например, чтобы протестировать GPS-приложения в эмуляторе, нужно создать файлы, имитирующие данные GPS. Хотя можно смоделировать изменения ориентации (переключение в *портретный/альбомный* режим), для моделирования показаний *акселерометра* (это устройство оценивает ориентацию и наклон устройства) необходимы функции, отсутствующие в эмуляторе. Впрочем, эмулятор может использовать данные датчиков с физического устройства Android, подключенного к компьютеру; описание этой возможности доступно по адресу

http://tools.android.com/tips/hardware-emulation

В табл. 1.13 перечислены функциональные аспекты Android, *недоступные* в эмуляторе. Впрочем, вы всегда можете загрузить свое приложение на устройство Android для их тестирования. Мы начнем создавать виртуальные устройства AVD и использовать эмулятор для разработки Android-приложений в приложении `Welcome` главы 2.

Таблица 1.13. Имитация жестов Android на эмуляторе

Жест	Действие в эмуляторе
Касание	Щелкните кнопкой мыши (приложение Tip Calculator — в главе 3)
Двойное касание	Дважды щелкните кнопкой мыши
Длинное нажатие	Щелкните кнопкой мыши и удерживайте ее (приложение Twitter® Searches — в главе 8)
Перетаскивание	Щелкните левой кнопкой мыши, удерживайте ее и переместите мышь (приложение Cannon Game — в главе 6)
Смахивание	Щелкните кнопкой мыши и, удерживая ее, переместите указатель мыши в направлении смахивания, а потом отпустите кнопку мыши (приложение Weather Viewer — в главе 7)
Масштабирование двумя пальцами	Нажмите и удерживайте Ctrl. Появятся две окружности, имитирующие касание экрана двумя пальцами. Переместите окружности в начальную позицию, щелкните кнопкой мыши и, удерживая ее, переместите окружности в конечную позицию

Таблица 1.14. Имитация органов управления Android в эмуляторе (имитация других органов управления описана на сайте http://developer.android.com/tools/help/emulator.html)

Орган управления устройства Android	Действие в эмуляторе
Back (Назад)	Esc
Вызов/набор номера	F3
Камера	Ctrl-KEYPAD_5, Ctrl-F3
Завершить вызов	F4
Home (Домой)	Кнопка Home
Меню (левая программная кнопка)	F2 или Page Up
Питание	F7
Поиск	F5
* (правая программная кнопка)	Shift-F2 или Page Down
Поворот в предыдущую ориентацию	KEYPAD_7, Ctrl-F11
Поворот в следующую ориентацию	KEYPAD_9, Ctrl-F12
Включение/выключение сети мобильной связи	F8
Кнопка увеличения громкости	KEYPAD_PLUS, Ctrl-F5
Кнопка уменьшения громкости	KEYPAD_MINUS, Ctrl-F6

Таблица 1.15. Функциональность Android, недоступная в эмуляторе (http://developer.android.com/tools/devices/emulator.html#limitations)

Функциональность Android, недоступная в эмуляторе
Реальные телефонные звонки (эмулятор поддерживает только имитацию звонков)
Bluetooth
Подключения USB
Гарнитуры, подключенные к устройству
Определение состояния подключения к сети
Определение заряда батареи или состояния зарядки
Определение вставки/извлечения SD-карты
Прямая поддержка датчиков (акселерометр, барометр, компас, датчик освещения, дистанционный датчик) — впрочем, существует возможность получения данных с устройства, подключенного через USB

1.8. Краткий обзор объектно-ориентированного программирования

В программировании для Android используются объектно-ориентированные технологии, поэтому в данном разделе мы приведем обзор объектных технологий. Все эти концепции будут часто встречаться в книге.

В условиях постоянно растущего спроса на новые мощные программные продукты достаточно трудно сочетать такие требования, как быстрота разработки, правильность работы и экономичность. *Объекты* (а точнее, как мы увидим в главе 3, *классы*, на основе которых создаются объекты) по сути представляют собой *повторно используемые* программные компоненты. В качестве объектов могут использоваться дата, время, видео, человек, автомобиль и другие предметы материального мира. Практически каждое *существительное* может быть адекватно представлено программным объектом в понятиях *атрибутов* (например, имя, цвет и размер) и *поведений* (например, вычисление, перемещение и передача данных). Разработчики программ видят, что использование модульной структуры и объектно-ориентированного проектирования при разработке приложений повышает продуктивность работы. Этот подход пришел на смену применявшемуся ранее структурному программированию — объектно-ориентированный код проще понять и изменить.

1.8.1. Автомобиль как объект

Чтобы лучше понять суть объектов и их содержимого, воспользуемся простой аналогией. Представьте себе, что вы *находитесь за рулем автомобиля и нажимаете педаль газа, чтобы набрать скорость*. Что должно произойти до того, как вы получите такую возможность? Прежде чем вы поведете автомобиль, кто-то должен его *спроектировать*. Изготовление любого автомобиля начинается с инженерных чертежей, которые подробно описывают устройство автомобиля. В частности, на этих чертежах показано устройство педали акселератора. За этой педалью *скрываются* сложные механизмы, которые непосредственно ускоряют автомобиль, подобно тому как педаль тормоза *скрывает* механизмы, тормозящие автомобиль, а руль *скрывает* механизмы поворота. Благодаря этому люди, не имеющие понятия о внутреннем устройстве автомобиля, могут легко им управлять.

Подобно тому как невозможно готовить пищу на кухне, которая лишь изображена на листе бумаги, нельзя водить автомобиль, существующий лишь в чертежах. Прежде чем вы сядете за руль машины, ее нужно *построить* на основе инженерных чертежей. Воплощенный в металле автомобиль имеет *реальную* педаль газа, с помощью которой он может ускоряться, но и это не все — он не может это делать самостоятельно (к счастью!), а только после того, как водитель *нажмет* на педаль.

1.8.2. Методы и классы

Воспользуемся примером с автомобилем для демонстрации некоторых ключевых концепций объектно-ориентированного программирования. Для выполнения операции в программе требуется *метод*, в котором «скрываются» инструкции программы, непосредственно выполняющие операцию. Метод скрывает эти инструкции от пользователя подобно тому, как педаль газа автомобиля скрывает от водителя механизмы, вызывающие ускорение автомобиля. Программная единица, именуемая классом, включает методы, выполняющие задачи класса. Например, класс, представляющий банковский счет, может включать три метода, один из которых пополняет счет, второй снимает средства со счета, а третий запрашивает текущий баланс. С концептуальной точки зрения класс подобен инженерному чертежу, в котором изображено устройство педали газа, рулевого колеса и других механизмов.

1.8.3. Создание экземпляра класса

Водитель не сядет за руль автомобиля, пока его не *построят* по чертежам, так и *построение объекта* класса необходимо для выполнения задач, определяемых методами класса. Этот процесс называется *созданием экземпляра*. Полученный при этом объект называется *экземпляром класса*.

1.8.4. Повторное использование

На основе одних и тех же чертежей можно создать много автомобилей, а на основе одного класса можно создать много объектов. Использование существующих классов для создания новых классов экономит время и силы разработчика. *Повторное использование* также облегчает создание более надежных и эффективных систем, поскольку ранее созданные классы и компоненты обычно проходят тщательное тестирование, отладку и оптимизацию. Подобно тому как концепция использования взаимозаменяемых частей легла в основу промышленной революции, повторно используемые классы играют ключевую роль в программной революции, которая была инициирована внедрением объектных технологий.

1.8.5. Сообщения и вызовы методов

Когда вы ведете машину, нажатие педали газа отправляет автомобилю *сообщение* с запросом на выполнение определенной задачи (ускорение автомобиля). Подобным же образом *отправляются сообщения объекту*. Каждое сообщение

представляется вызовом метода, который «сообщает» методу объекта о необходимости выполнения некоторой задачи. Например, программа может вызвать метод `deposit` объекта банковского счета, чтобы пополнить банковский счет.

1.8.6. Атрибуты и переменные экземпляра класса

Любой автомобиль помимо возможности выполнять определенные операции также обладает *атрибутами*, такими как цвет, количество дверей, запас топлива в баке, показания спидометра и одометра. По аналогии с операциями атрибуты автомобиля представляются на инженерных диаграммах (в качестве атрибутов автомобиля могут выступать одометр и указатель уровня бензина). При вождении автомобиля его атрибуты перемещаются вместе с ним. Каждый автомобиль содержит *собственный* набор атрибутов. Например, каждый автомобиль «знает» о том, сколько бензина осталось в его баке, но ему ничего не известно о запасах горючего в баках *других* автомобилей.

Объект, как и автомобиль, имеет собственный набор атрибутов, которые он «переносит» с собой при использовании этого объекта в программах. Эти атрибуты определяются как часть объекта класса. Например, объект `bankaccount` имеет атрибут баланса, представляющий количество средств на банковском счете. Каждый объект `bankaccount` «знает» о количестве средств на собственном счете, но ничего не «знает» о размерах *других* банковских счетов. Атрибуты определяются с помощью других *переменных экземпляра* класса.

1.8.7. Инкапсуляция

Классы *инкапсулируют* атрибуты и методы в объекты (атрибуты и методы объекта между собой тесно связаны). Объекты могут обмениваться информацией между собой, но обычно они не «знают» о деталях реализации других объектов, которые скрыты внутри самих объектов. Подобное сокрытие информации жизненно важно в практике хороших программных архитектур.

1.8.8. Наследование

С помощью наследования можно быстро и просто создать новый класс объектов. При этом новый класс наследует характеристики существующего класса, которые при этом могут частично изменяться. Также в новый класс добавляются уникальные характеристики, присущие только этому классу. Если вспомнить аналогию с автомобилем, «трансформер» является объектом более обобщенного класса «автомобиль», у которого может подниматься или опускаться крыша.

1.8.9. Объектно-ориентированный анализ и проектирование

А теперь ответьте на вопрос, каким образом вы собираетесь программировать? Скорее всего, таким же образом, как и большинство других программистов, — включите компьютер и начнете вводить исходный код программы. Подобный подход годится при создании маленьких программ, но что делать в том случае, когда приходится создавать крупный программный комплекс, который, например, управляет тысячами банкоматов крупного банка? Либо если вам приходится возглавлять команду из 1000 программистов, занятых разработкой системы управления воздушным движением следующего поколения? В таких крупных и сложных проектах нельзя просто сесть за компьютер и начать набирать код.

Чтобы выработать наилучшее решение, следует провести детальный анализ *требований* к программному проекту (то есть определить, *что* должна делать система) и разработать архитектуру, которая будет соответствовать этим требованиям (то есть определить, *как* система будет выполнять свои задачи). В идеале перед началом создания кода следует выполнить эту процедуру и тщательно проанализировать проект (либо поручить выполнение этой задачи другим профессионалам). Если в ходе выполнения этого процесса происходит анализ и проектирование системы с применением объектно-ориентированного подхода, значит, мы имеем дело с процессом *объектно-ориентированного анализа и проектирования* (ООАП). Языки программирования, подобные Java, называются объектно-ориентированными. Программирование на таких языках, называемое *объектно-ориентированным программированием* (ООП), реализует объектно-ориентированные проекты в виде работоспособных систем.

1.9. Тестирование приложения Tip Calculator на виртуальном устройстве AVD

В этом разделе вы запустите на выполнение и будете «общаться» со своим первым приложением Android на виртуальном устройстве — или на физическом, если оно у вас есть. С помощью приложения `Tip Calculator` (рис. 1.1, *а*), которое мы построим в главе 3, вычисляется и отображается сумма ресторанных чаевых и общая сумма. В процессе ввода пользователем выставленного счета с цифровой клавиатуры приложение вычисляет и отображает величину итогового счета и размер чаевых (по умолчанию 15%). Пользователь может указать собственную ставку в диапазоне от 0% до 30% (по умолчанию 18%). Для этого нужно переместить ползунок компонента `Seekbar`, после чего обновляется величина процентной ставки, а обновленная сумма чаевых и общий счет отображаются в текстовых полях под компонентом `Seekbar` (рис. 1.1, *б*).

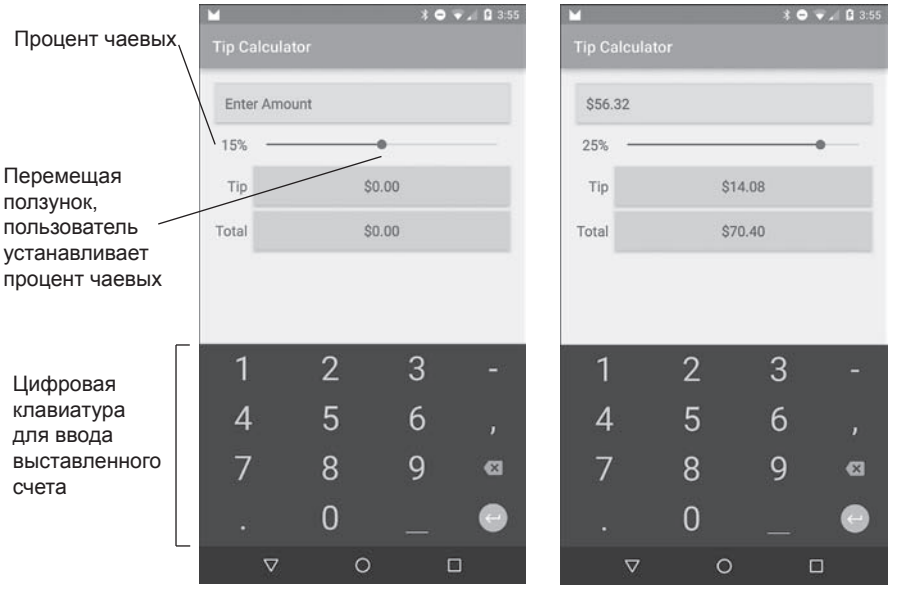


Рис. 1.1. Приложение Tip Calculator при загрузке и после ввода суммы и изменения процента чаевых: *а* — приложение Tip Calculator после загрузки; *б* — приложение Tip Calculator после того, как пользователь введет сумму счета и установит 25% чаевых

1.9.1. Запуск приложения Tip Calculator в Android Studio

Чтобы открыть проект приложения `Tip Calculator`, выполните следующие действия.

1. **Проверка конфигурации системы.** Выполните действия, описанные в разделе «Подготовка» (если это не было сделано ранее).
2. **Запуск Android Studio.** Щелкните на значке Android Studio, чтобы открыть среду разработки. В системе Windows значок находится в меню `Пуск` (Start) или на основном экране. В OS X он находится в папке `Applications`. В Linux местонахождение значка зависит от того, в какую папку был распакован ZIP-файл с файлами Android Studio. После первого запуска Android Studio на экране появляется окно Welcome to Android Studio (рис. 1.2).
3. **Открытие проекта приложения Tip Calculator**. Если в Android Studio уже открыт другой проект, выполните команду `File` ▸ `Open`..., перейдите в папку проекта и откройте ее. Также проект можно открыть из окна Welcome to Android Studio (рис. 1.2) — щелкните на ссылке `Open an existing Android Studio Project`, чтобы открыть диалоговое окно `Open File or Project` (рис. 1.3).

Рис. 1.2. Окно Welcome to Android Studio

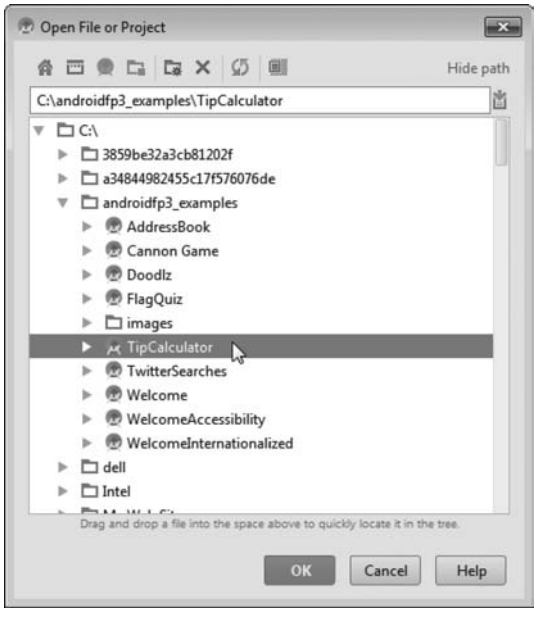


Рис. 1.3. Диалоговое окно Open File or Project

Перейдите в папку примеров книги, выберите папку `TipCalculator` и щелкните на кнопке `Choose` (Mac) или `OK` (Windows/Linux). Android Studio сохраняет путь к Android SDK в настройках каждого созданного вами проекта. Открыв наш проект в своей системе, вы получите сообщение об ошибке, если путь к SDK в вашей системе отличается от нашего. Просто щелкните на кнопке `OK` в окне ошибки, и Android Studio обновит настройки проекта в соответствии с местонахождением SDK в вашей системе. IDE открывает проект и отображает его содержимое на окне `Project` в левой части окна (рис. 1.4). Если окно `Project` не отображается, выполните команду `View` ▸ `Tool Windows` ▸ `Project`.

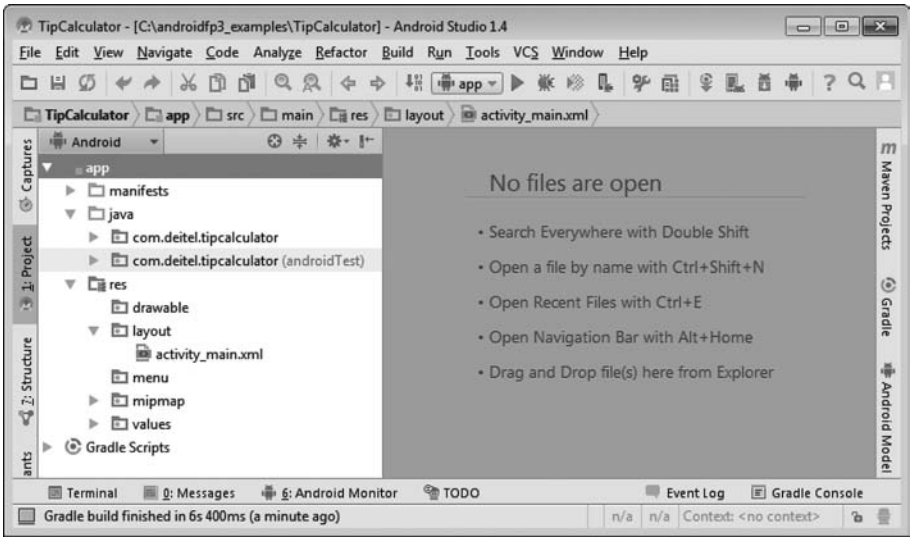


Рис. 1.4. Окно Project в проекте Tip Calculator

1.9.2. Создание виртуальных устройств Android (AVD)

Как упоминалось в разделе «Подготовка», приложения можно тестировать на разных устройствах Android; для этого нужно создать виртуальные устройства Android (AVD), эмулирующие каждое тестовое устройство[1].

В этом разделе мы создадим Android 6 AVD для устройств, которые будут использоваться для тестирования приложений в книге, — телефона Google Nexus 6 и планшета Nexus 9. Чтобы создать эти AVD, выполните следующие действия:

[1] На момент написания книги при установке Android Studio создается устройство AVD, эмулирующее телефон Google Nexus 5 с Android 6.0 (Marshmallow). Чтобы создать дополнительные AVD, необходимые для тестирования, выполните действия из раздела 1.9.2.

1. В Android Studio выполните команду `Tools` ▸ `Android` ▸ `AVD Manager`. На экране появляется окно `Android Virtual Device Manager` (рис. 1.5).

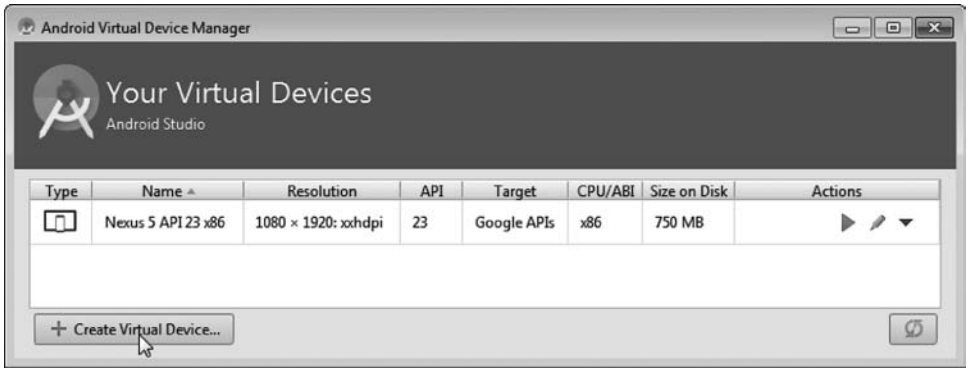


Рис. 1.5. Окно Android Virtual Device Manager

2. Щелкните на кнопке `Create Virtual Device...`, чтобы открыть окно `Virtual Device Configuration` (рис. 1.6). По умолчанию в столбце `Category` выбрана категория `Phone`, но вы также можете создать AVD для планшета (`Tablet`), носимого устройства (`Wear`) или телевизора (`TV`). Для удобства Google предоставляет много заранее подготовленных устройств, которые могут использоваться для быстрого создания AVD. Выберите Nexus 6 и щелкните на кнопке `Next`.

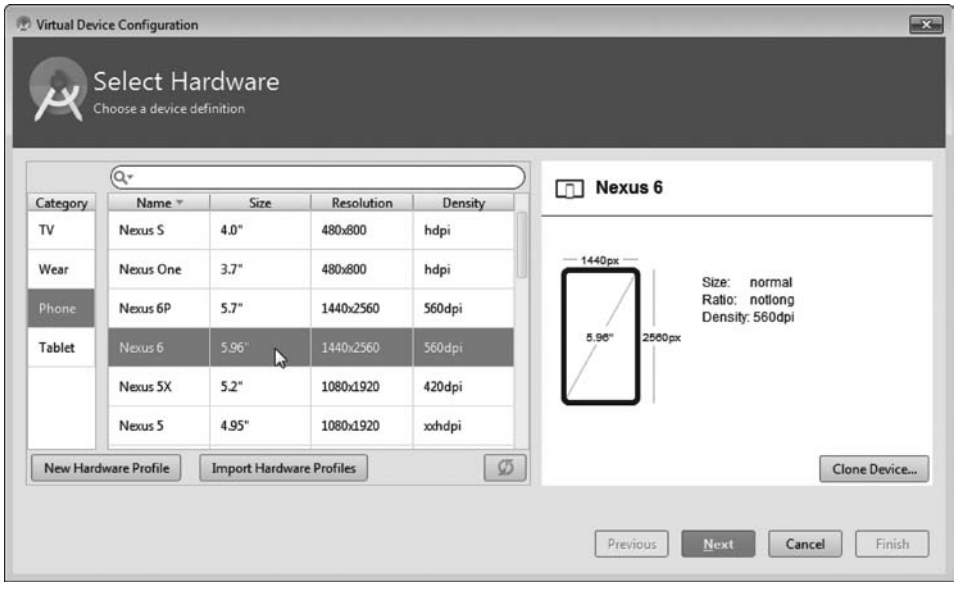


Рис. 1.6. Окно Virtual Device Configuration

3. Выберите образ системы для виртуального устройства, которое вы собираетесь создать, — в данном случае для платформы `Marshmallow` уровня API 23, ABI (Application Binary Interface) x86 и целью Android 6.0 (с Google API). Щелкните на кнопке `Next`. Android Studio создает устройство Android AVD для Android 6, также включающее поддержку Google Play Services API.
4. В поле `AVD Name` введите имя `Nexus 6 API 23`.
5. Щелкните на кнопке `Show Advanced Settings` в левом нижнем углу окна `Virtual Device Configuration`, прокрутите панель расширенных настроек до конца, *снимите* флажок `Enable Keyboard Input` и щелкните на кнопке `Finish`.
6. Повторите шаги 1–6, создайте AVD для планшета Nexus 9 и присвойте ему имя `Nexus 9 API 23`. Это устройство AVD будет использоваться в главе 2.

Если оставить установленным флажок `Enable Keyboard Input` (шаг 5), вы сможете использовать клавиатуру своего компьютера для ввода данных в приложениях, выполняемых в AVD. Однако в этом случае не будет отображаться экранная клавиатура, показанная на снимках экрана.

Для каждого созданного вами устройства AVD определяется множество других параметров конфигурации, которые хранятся в его файле *config.ini*. Чтобы задать более точное описание аппаратной конфигурации устройства, отредактируйте содержимое `config.ini`:

http://developer.android.com/tools/devices/managing-avds.html

1.9.3. Запуск приложения Tip Calculator на AVD смартфона Nexus 6

Чтобы протестировать приложение `Tip Calculator`, выполните следующие действия.

1. **Проверка конфигурации.** Выполните действия, описанные в разделе «Подготовка» (если это не было сделано ранее).
2. **Запуск Nexus 6 AVD.** Для тестового запуска мы используем устройство AVD смартфона Nexus 6, созданное в разделе 1.9.2. Чтобы запустить AVD Nexus 6, выполните команду `Tools ▸ Android ▸ AVD Manager`; на экране появляется диалоговое окно Android Virtual Device Manager (рис. 1.7). Щелкните на кнопке запуска AVD в эмуляторе (▶) в строке Nexus 6 API 23 AVD. Загрузка AVD потребует времени — не пытайтесь запускать приложение до завершения загрузки. Когда все будет готово, AVD отображает экран блокировки. На физическом устройстве блокировка снимается жестом смахивания. Чтобы имитировать этот жест в AVD, поместите указатель мыши над «экраном» AVD и протащите. На рис. 1.8 показано, как выглядит AVD после снятия блокировки.

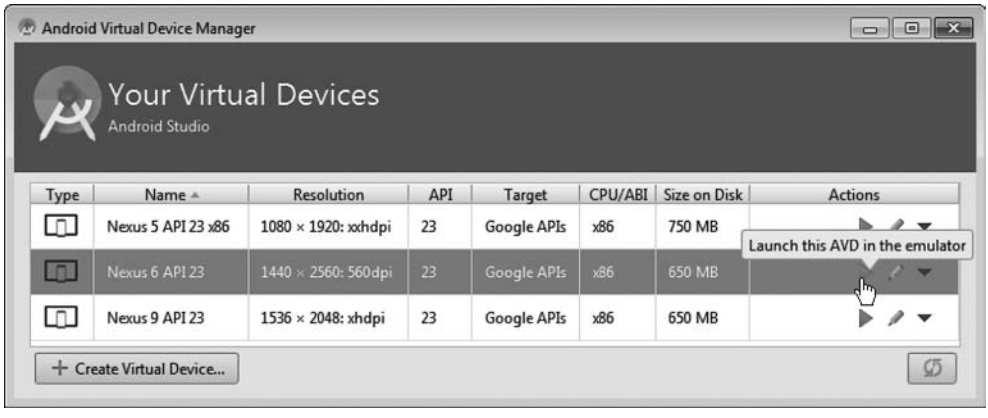


Рис. 1.7. Диалоговое окно Android Virtual Device Manager

Рис. 1.8. Главный экран Nexus 6 AVD после снятия блокировки

3. **Запуск приложения Tip Calculator.** В Android Studio выполните команду `Run ▸ Run 'app'` или щелкните на кнопке `Run 'app'` на панели инструментов Android Studio (▶). На экране появляется диалоговое окно `Device Chooser`

(рис. 1.9), в котором уже выбрано текущее устройство AVD. Щелкните на кнопке `OK`, чтобы запустить приложение `Tip Calculator` на устройстве AVD, выбранном на шаге 2 (рис. 1.10)[1]. Также существует другой способ: вместо открытия Android Virtual Device Manager на шаге 2 щелкните на кнопке `Run 'app'` (▶) на панели инструментов Android Studio toolbar; появляется диалоговое окно `Device Chooser`. После этого воспользуйтесь переключателем `Launch emulator` в нижней части диалогового окна и выберите устройство AVD для запуска приложения.

4. **Структура AVD.** В нижней части экрана AVD расположены различные программные кнопки, отображаемые на сенсорном экране устройства. Пользователь касается этих кнопок, чтобы взаимодействовать с приложением и ОС Android. В AVD касания имитируются щелчками мышью. Кнопка «вниз» (▽) закрывает клавиатуру. Если на экране нет клавиатуры, то вместо нее

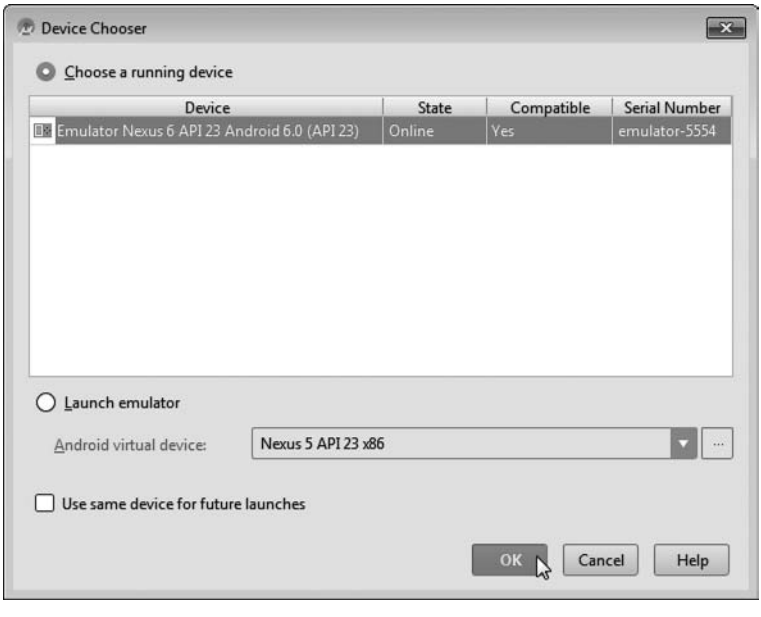


Рис. 1.9. Окно Device Chooser для выбора AVD или устройства

[1] Клавиатура на рис. 1.10 может выглядеть иначе в зависимости от AVD, версии Android устройства и т. д. Мы настроили AVD для отображения темной клавиатуры (чтобы улучшить контраст на снимках экрана). Для этого коснитесь значка Home (○) на AVD или устройстве. На домашнем экране коснитесь значка лаунчера (⋯) и откройте приложение Settings. В разделе Personal выберите раздел Language and Input. В AVD выберите Android Keyboard (AOSP), на устройстве выберите Google Keyboard (стандартная клавиатура Android). Выберите Appearance & layouts, затем Theme и выберите вариант Material Dark, чтобы переключиться на клавиатуру с темным фоном.

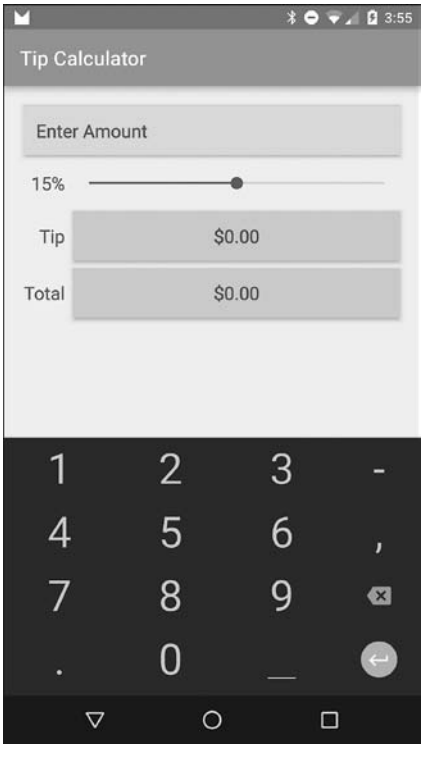


Рис. 1.10. Приложение Tip Calculator в AVD

отображается кнопка «назад» (◁). Эта кнопка выполняет возврат к предыдущему экрану приложения или предыдущему приложению, если текущим является начальный экран приложения. Кнопка ○ возвращает пользователя к главному экрану устройства, а кнопка □ позволяет просмотреть список недавно использовавшихся приложений для быстрого возврата к ним. В верхней части экрана располагается панель приложения; на ней выводится имя приложения и могут выводиться другие программные кнопки — часть кнопок может размещаться прямо на панели приложения, другие кнопки вынесены в *меню команд*, представленное кнопкой ⋮. Количество командных элементов на панели приложения зависит от размеров устройства — эта тема рассматривается в главе 5.

5. **Ввод суммы счета.** Введите сумму `56.32` с экранной клавиатуры. Если в процессе ввода будет допущена ошибка, нажмите кнопку удаления (⌫) в правом нижнем углу клавиатуры, чтобы стереть последнюю введенную цифру. Хотя на клавиатуре имеется клавиша с точкой, приложение настроено так, что пользователь может вводить только цифры 0–9. Каждый раз, когда вы вводите или удаляете цифру, приложение читает текущий введенный текст

и преобразует его в число (если удалить все цифры, приложение снова выводит подсказку `Enter Amount` в компоненте `TextView` в верхней части). Приложение делит значение на 100 и выводит результат в синем поле, после чего вычисляет и обновляет величину чаевых и общую сумму. Мы используем средства форматирования Android для вывода денежных сумм, отформатированных по правилам текущего локального контекста устройства. Для локального контекста США при вводе четырех цифр — 5, 6, 3 и 2 — сумма счета выводится последовательно в форматах $0.05, $0.56, $5.63 и $56.32.

6. **Выбор процента чаевых.** Компонент `SeekBar` позволяет выбрать процент чаевых, а в компонентах `TextView` в правом столбце под `SeekBar` отображаются соответствующие значения чаевых и общей суммы. Перетащите ползунок `Seekbar` до отметки 25%. В процессе перетаскивания происходит непрерывное обновление `SeekBar`. Приложение обновляет процент чаевых, величину чаевых и общий счет для текущего значения `Seekbar`, пока не будет отпущен ползунок. На рис. 1.11 показано, как выглядит приложение после ввода суммы счета и выбора процента чаевых.

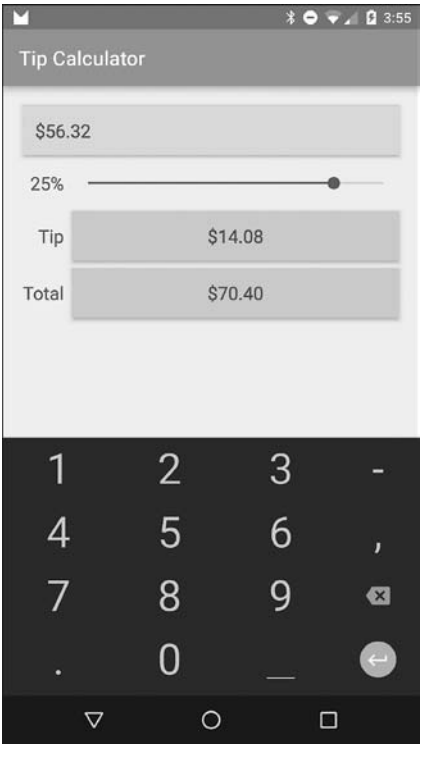


Рис. 1.11. Tip Calculator после ввода суммы и выбора 25% чаевых

7. **Возврат к главному экрану.** Чтобы вернуться к главному экрану AVD, коснитесь кнопки ● на экране AVD.

Возможные проблемы при запуске AVD

Если у вас возникли проблемы с выполнением AVD, возможно, AVD выделено слишком много памяти. Чтобы сократить объем памяти AVD, выполните следующие действия.

1. В Android Studio выполните команду `Tools ▸ Android ▸ AVD Manager`. На экране появляется окно `Android Virtual Device Manager`.
2. В окне выводится список существующих AVD. Для того устройства, конфигурацию которого вы хотите изменить, щелкните на кнопке ✎ в столбце `Actions`.
3. В окне `Virtual Device Configuration` щелкните на кнопке `Show Advanced Settings` и прокрутите список до раздела `Memory and Storage`.
4. Уменьшите значение `RAM` с используемых по умолчанию 1536 Мбайт (1,5 Гбайт) до 1 Гбайт.
5. Щелкните на кнопке `Finish` и закройте окно `Android Virtual Device Manager`.

Если AVD по-прежнему не запускается, повторите эти действия и сократите объем памяти до 768 Мбайт.

1.9.4. Выполнение приложения Tip Calculator на устройстве Android

Если у вас имеется устройство Android, вы можете легко протестировать приложение на этом устройстве.

1. **Активизируйте средства разработчика на устройстве.** Прежде всего нужно включить для устройства режим отладки. Запустите приложение `Settings`, установленное на устройстве, затем выберите раздел `About Phone` (или `About Tablet`), найдите пункт `Build Number` (в конце списка) и последовательно нажимайте на него, пока на экране не появится сообщение `You are now a developer`. При этом в приложении `Settings` появляется раздел `Developer Options`.
2. **Включите режим отладки на устройстве.** Вернитесь в приложение `Settings`, выберите раздел `Developer Options` и убедитесь в том, что флажок `USB debugging` установлен — это происходит по умолчанию при первом включении средств разработчика на устройстве.

3. **Подключите устройство.** Подключите устройство к компьютеру с помощью кабеля USB, входящего в комплект поставки вашего устройства. Если вы пользователь Windows, вспомните, о чем говорилось в разделе «Подготовка», — возможно, вам потребуется установить драйвер USB для вашего устройства. Подробную информацию можно найти на следующих двух страницах:

   developer.android.com/tools/device.html
   developer.android.com/tools/extras/oem-usb.html

4. **Запустите Tip Calculator на устройстве Android.** В Android Studio выполните команду `Run ▸ Run 'app'` или щелкните на кнопке `Run 'app'` на панели инструментов Android Studio (▶). На экране появляется диалоговое окно `Device Chooser` (рис. 1.9). Выберите свое устройство в списке работающих AVD и устройств. Щелкните на кнопке `OK`, чтобы запустить приложение `Tip Calculator` на AVD или выбранном вами устройстве.

Тестирование приложений

Чтобы получить общее представление о возможностях, которые будут представлены в книге, просмотрите описания пробных запусков приложений в главах 2–9.

Подготовка к распространению приложений

Если вы создаете приложения, предназначенные для распространения через Google Play или другие магазины, протестируйте их на возможно большем числе реальных устройств. Не забывайте о том, что некоторые функции могут быть протестированы *только* на реальных устройствах. Если вы не можете найти реальные устройства в достаточном количестве, создайте устройства AVD, имитирующие различные устройства, на которых должно выполняться созданное вами приложение, — AVD Manager предоставляет множество готовых шаблонов AVD. Когда вы настраиваете AVD для имитации конкретного устройства, просмотрите в Интернете спецификации и настройте соответствующим образом каждое виртуальное устройство AVD. Также можно изменить файл *config.ini* виртуального устройства AVD, как описано в разделе «Setting hardware emulation options» на сайте

http://developer.android.com/tools/devices/managing-avds-cmdline.html#hardwareopts

Файл *config.ini* включает параметры, которые не настраиваются в `Android Virtual Device Manager`. Настройка этих параметров позволяет добиться более точного соответствия выбранного виртуального устройства Android с реальным.

1.10. Создание успешных Android-приложений

Сейчас в Google Play доступно свыше 1,6 миллиона приложений[1]. Как создать приложение Android, которое пользователи найдут, загрузят, будут использовать и рекомендовать другим? Подумайте, что сделает ваше приложение интересным, полезным, привлекательным и долгоживущим. Эффектное название, привлекательный значок и заманчивое описание могут привлечь людей к вашему приложению в Google Play или одном из многих других магазинов приложений Android. Но когда пользователь загрузит ваше приложение, что заставит его регулярно работать с приложением и порекомендовать его другим? В табл. 1.16 перечислены некоторые характеристики успешных приложений.

Таблица 1.16. Характеристики успешных приложений

Характеристики успешных приложений
Успешные игры
Интересны и увлекательны
Требуют усилий со стороны игрока
Предоставляют разные уровни сложности
Выводят счет и ведут таблицы рекордов
Предоставляют звуковую и визуальную обратную связь
Реализуют качественную анимацию
Выделяют операции ввода/вывода и интенсивные вычисления в отдельные программные потоки для повышения скорости отклика интерфейса и быстродействия приложения
Используют инновационную технологию дополненной реальности, дополняя реальное окружение виртуальными компонентами; данная возможность особенно популярна в приложениях на базе видео
Утилиты
Предоставляют полезную функциональность и точную информацию
Повышают персональную производительность и эффективность бизнеса
Упрощают выполнение операций (ведение списков задач, управление расходами и т. д.)
Повышают информированность пользователя

[1] *http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.*

Характеристики успешных приложений
Предоставляют тематическую информацию (последние биржевые котировки, новости, предупреждения о штормах, оперативная информация о потоке транспорта и т. д.)
Используют географическую привязку для предоставления локального сервиса (купоны для местных магазинов, лучшие цены на бензин, доставка еды и т. д.)
Общие характеристики
Используют новейшие возможности Android, но совместимы с разными версиями Android для поддержки максимально широкой аудитории
Работают корректно
Ошибки быстро исправляются
Следуют стандартным правилам графического интерфейса приложений Android
Быстро запускаются
Быстро реагируют на действия пользователя
Не требуют слишком большого объема памяти, передачи больших объемов данных или заряда батареи
Оригинальны и необычны
Выдерживают испытание временем (то есть пользователи периодически возвращаются к ним)
Используют значки профессионального уровня, которые отображаются в Google Play и на устройстве пользователя
Используют качественную графику, анимации, аудио и видео
Интуитивно понятны и просты в использовании (не требуют обширной справочной документации)
Доступны для людей с ограниченными возможностями (*http://developer.android.com/guide/topics/ui/accessibility/index.html*)
Предоставляют причины и средства для передачи другим пользователям информации о приложении (например, дают возможность опубликовать игровой результат в Facebook или Twitter)
Предоставляют дополнительный контент там, где это уместно (игровые уровни, статьи, головоломки и т. д.)
Локализуются (см. главу 2) для каждой страны, в которой распространяются (например, перевод текстовых и звуковых файлов, использование разной графики в зависимости от местонахождения пользователя и т. д.)
Превосходят приложения конкурентов по быстродействию, функциональности или простоте использования

Таблица 1.16 (*окончание*)

Характеристики успешных приложений
Эффективно используют встроенные возможности устройства
Не требуют лишних привилегий
Проектируются для оптимального выполнения в широком спектре устройств Android
Не теряют актуальности с выходом новых устройств — разработчик точно указывает набор аппаратных возможностей, используемых приложением, чтобы магазин Google Play мог фильтровать и отображать приложение только для совместимых устройств (*http://android-developers.blogspot.com/2010/06/future-proofing-your-app.html*)

1.11. Ресурсы для разработчиков

В табл. 1.17 перечислены некоторые важнейшие документы с сайта Android Developer. В ходе углубленного изучения разработки Android-приложений у вас могут возникнуть вопросы по инструментам, особенностям проектирования, безопасности и т. д. Существует целый ряд новостных групп и форумов для разработчиков Android, на которых можно найти новейшие объявления или задать вопросы (табл. 1.18). В табл. 1.19 приведены некоторые сайты, на которых можно найти рекомендации, видеоролики и ресурсы по программированию для Android.

Таблица 1.17. Важная электронная документация для Android-разработчиков

Название	URL
Компоненты приложений	http://developer.android.com/guide/components/index.html
Использование эмулятора Android	http://developer.android.com/tools/devices/emulator.html
Справочник пакетов	http://developer.android.com/reference/packages.html
Справочник классов	http://developer.android.com/reference/classes.html
Дизайн приложений	http://developer.android.com/design/index.html
Резервное копирование	http://developer.android.com/guide/topics/data/backup.html
Советы по безопасности	http://developer.android.com/training/articles/security-tips.html
Android Studio	http://developer.android.com/sdk/index.html
Отладка	http://developer.android.com/tools/debugging/index.html

Название	URL
Справочная информация по инструментам	http://developer.android.com/tools/help/index.html
Рекомендации по производительности	http://developer.android.com/training/articles/perf-tips.html
Обеспечение быстрой реакции интерфейса	http://developer.android.com/training/articles/perf-anr.html
Контрольный список публикации (для Google Play)	http://developer.android.com/distribute/tools/launch-checklist.html
Основы публикации приложений	http://developer.android.com/distribute/googleplay/start.html
Управление памятью приложения	http://developer.android.com/training/articles/memory.html
Соглашение о распространении приложений через Google Play для разработчиков	http://play.google.com/about/developer-distribution-agreement.html

Таблица 1.18. Новостные группы и форумы Android

Название	Подписка	Описание
Android Discuss	Подписка через Google Groups: android-discuss Подписка по электронной почте: android-discuss-subscribe@googlegroups.com	Общая конференция по программированию для Android, в которой можно получить ответы на вопросы по разработке приложений
Stack Overflow	http://stackoverflow.com/questions/tagged/android	Список вопросов начального уровня по разработке для Android и основным приемам программирования
Android Developers	http://groups.google.com/forum/?fromgroups#!forum/android-developers	Вопросы диагностики, проектирования графического интерфейса, оптимизации и т. д. для опытных разработчиков
Android Forums	http://www.androidforums.com	Здесь можно задать вопросы, обменяться информацией с другими разработчиками и найти форумы конкретных устройств на базе Android

Таблица 1.19. Новостные группы и форумы Android

Информация, видео, ресурсы	URL
Примеры кода и приложений от Google	https://github.com/google (используйте фильтр «android»)
Сайт Bright Hub™ с рекомендациями для Android-программистов и практическими руководствами	http://www.brighthub.com/mobile/google-android.aspx
The Android Developers Blog	http://android-developers.blogspot.com/
HTC's Developer Center for Android	http://www.htcdev.com/
Сайт Android-разработки компании Motorola	http://developer.motorola.com/
Top Android Users on Stack Overflow	http://stackoverflow.com/tags/android/topusers
AndroidDev Weekly Newsletter	http://androiddevweekly.com/
Codependent (блог Чета Хаазе)	http://graphics-geek.blogspot.com/
Блог Ромена Гая	http://www.curious-creature.org/category/android/
Канал для Android-разработчиков на YouTube®	http://www.youtube.com/user/androiddevelopers
Видео с конференции Google I/O 2015 Developer Conference	https://events.google.com/io2015/videos

1.12. Резюме

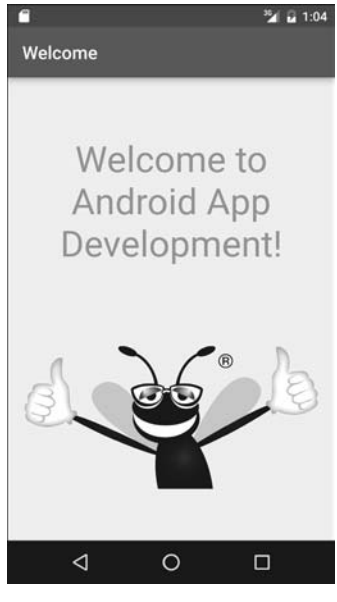
В этой главе вашему вниманию была представлена краткая история Android и функциональные свойства этой платформы. Были указаны ссылки на некоторую ключевую документацию в Интернете, группы новостей и форумы, которые позволят вам связаться с сообществом разработчиков. В этой главе был приведен обзор основных возможностей операционной системы Android, а также перечислены пакеты Java, Android и Google, которые позволят использовать аппаратную и программную функциональность для создания Android-приложений; многие из этих пакетов будут использованы в книге. Также были представлены язык программирования Java и Android SDK, жесты Android, способы их выполнения на устройствах Android и эмуляторах. Вашему вниманию был предложен краткий обзор концепций объектной технологии, в том числе классы, объекты, атрибуты и аспекты поведения. Мы протестировали приложение `Tip Calculator` на эмуляторах смартфона и планшета Android.

В главе 2 мы построим свое первое Android-приложение в Android Studio. Приложение выводит текст и графику. Также будут рассмотрены средства обеспечения доступности и интернационализации Android-приложений.

2 Приложение Welcome

Визуальное построение графического интерфейса, макеты, доступность и интернационализация

В этой главе…



- Изучение основных возможностей интегрированной среды Android Studio, применяемых для написания, выполнения и отладки Android-приложений
- Создание нового проекта приложения в среде разработки
- Визуальное построение графического интерфейса пользователя (без программирования) в макетном редакторе
- Вывод текста и графики в графическом интерфейсе
- Изменение свойств компонентов графического интерфейса
- Создание простого приложения Android и его выполнение в эмуляторе
- Повышение доступности приложения для пользователей с ослабленным зрением посредством назначения строк, используемых функциями Android TalkBack и Explore-by-Touch
- Поддержка интернационализации, чтобы приложение могло выводить локализованные строки на разных языках

2.1. Введение

В этой главе мы создадим приложение `Welcome`. Это простое приложение выводит приветственное сообщение и изображение. Мы воспользуемся Android Studio для создания простого Android-приложения (рис. 2.1), которое будет работать как на планшетах, так и на телефонах на базе Android как в портретной, так и в альбомной ориентации:

- в портретной ориентации высота устройства больше его ширины;
- в альбомной ориентации ширина устройства больше его высоты.

В *макетном редакторе* Android Studio графический интерфейс строится простым перетаскиванием. Также будет рассмотрено прямое редактирование разметки XML. Наконец, мы запустим приложение на *эмуляторе* Android (и на физическом устройстве, если оно доступно).

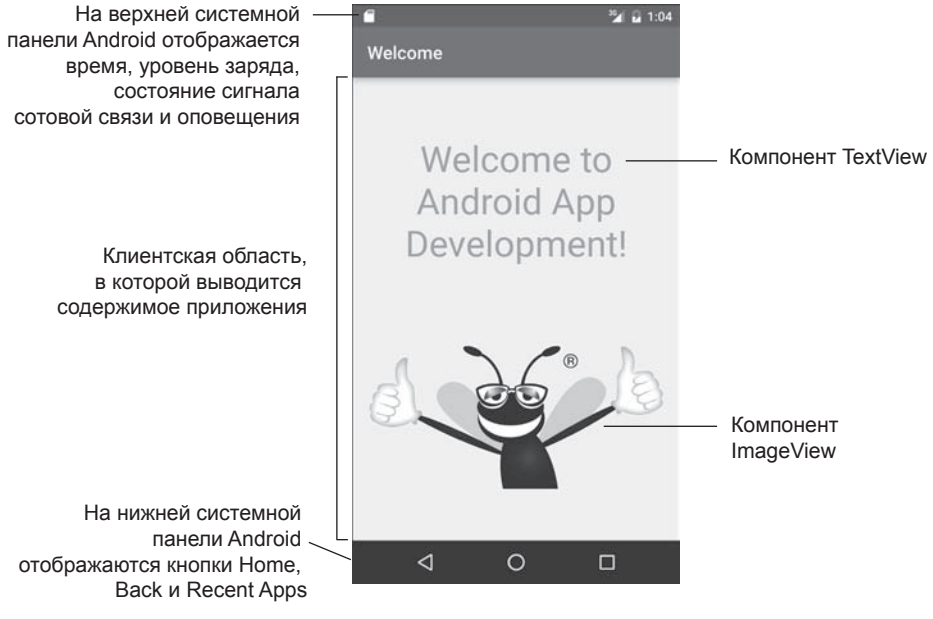


Рис. 2.1. Приложение Welcome в эмуляторе Android

Вы узнаете, как сделать приложение *доступным* для людей с ограниченными возможностями. Благодаря технологии Android *Explore by Touch* пользователь с ослабленным зрением может прикасаться к различным элементам экрана, а экранный диктор TalkBack будет произносить описания этих элементов.

Будет рассмотрен процесс тестирования этих возможностей, доступных только на устройствах Android.

В завершение вы узнаете, как организовать вывод строк, *локализованных* для разных языков. Мы изменим настройки локального контекста в эмуляторе и протестируем приложение на испанском языке! В ходе выполнения приложения Android выбирает правильные строки в зависимости от локального контекста; мы покажем, как изменить локальный контекст на устройстве. Предполагается, что вы ознакомились с предисловием, разделом «Подготовка» и разделом 1.9.

2.2. Обзор применяемых технологий

В этом разделе представлены технологии, которые будут использоваться при построении приложения `Welcome`.

2.2.1. Android Studio

В разделе 2.3 мы воспользуемся интегрированной средой разработки (IDE) Android Studio для создания нового приложения. Как вы увидите, среда создает простейший графический интерфейс с текстом “Hello world!”. Далее мы воспользуемся режимами представления `Design` и `Text`, а также окном свойств для построения простого графического интерфейса с текстом и изображением (см. раздел 2.5).

2.2.2. LinearLayout, TextView и ImageView

Компоненты графического интерфейса в Android называются *представлениями* (views). Вертикальное представление `LinearLayout` используется для размещения текста и графики так, чтобы каждое представление занимало половину вертикального пространства `LinearLayout`. Компонент `LinearLayout` также позволяет размещать представления по горизонтали.

Для вывода текста в приложении будет использоваться компонент TextView, а графика будет отображаться в компоненте `ImageView`. Графический интерфейс, созданный для приложения по умолчанию, содержит компонент `TextView`. Различные параметры этого компонента — текст, размер шрифта, цвет текста, размер относительно компонента `ImageView` в `LinearLayout` и т. д. — настраиваются в окне свойств среды разработки (см. раздел 2.5.5). Затем мы перетащим компонент `ImageView` с палитры в макет графического интерфейса (см. раздел 2.5.4) и настроим его свойства, включая источник графических данных и позицию в `LinearLayout`.

2.2.3. XML

Язык *XML* (eXtensible Markup Language, то есть расширяемый язык разметки) является естественным способом описания графических интерфейсов. Разметка XML хорошо читается как человеком, так и компьютером; в контексте Android она используется для описания макетов используемых компонентов и их атрибутов: размера, позиции, цвета, размера текста, полей и отступов. Android Studio разбирает разметку XML, чтобы отобразить макет в макетном редакторе и сгенерировать код Java, формирующий графический интерфейс на стадии выполнения. Также файлы XML используются для хранения ресурсов приложения: строк, чисел, цветов и т. д. (раздел 2.2.4).

2.2.4. Ресурсы приложения

Все строки и числовые значения рекомендуется определять в файлах ресурсов, которые размещаются во вложенных папках папки `res` проекта. В разделе 2.5.5 вы узнаете, как создавать ресурсы для строк (например, для текста в компоненте `TextView`) и метрик (например, размера шрифта). Также будет продемонстрировано использование цвета, выбранного на палитре цветов `Material Design`, для определения цвета шрифта `TextView`:

http://www.google.com/design/spec/style/color.html

2.2.5. Доступность

Android содержит разнообразные *специальные средства*, упрощающие использование устройств людьми с ограниченными возможностями. Например, слабовидящие пользователи могут воспользоваться функцией `TalkBack` — экранным диктором, читающим текст на экране (или текст, предоставленный разработчиком) для лучшего понимания назначения и содержимого компонента. Функция Android `Explore by Touch` позволяет прикоснуться к экрану, чтобы диктор `TalkBack` зачитал информацию о содержимом экрана рядом с точкой касания. В разделе 2.7 показано, как включить эти функции и как настроить компоненты графического интерфейса приложения для улучшения доступности.

2.2.6. Интернационализация

Устройства на базе Android используются во всем мире. Чтобы ваши приложения охватывали как можно большую аудиторию, подумайте о том, чтобы адаптировать их для разных культур и языков — этот процесс называется *интернационализацией*. В разделе 2.8 показано, как в приложении `Welcome` предоставить

текст на испанском языке для компонента `TextView` и строки доступности для компонента `ImageView` и как затем протестировать приложение на виртуальном устройстве AVD, настроенном для испанского языка.

2.3. Создание приложения

Примеры книги разрабатывались с использованием версии Android 6 SDK, актуальной на момент написания книги. В этом разделе будет показано, как создать новый проект в IDE, а другие функции IDE будут представлены в книге.

2.3.1. Запуск Android Studio

Как и в разделе 1.9, запустите Android Studio с помощью значка  .

Среда разработки отображает окно `Welcome` (рис. 1.17) или последний проект, открывавшийся ранее.

2.3.2. Создание нового проекта

Проект — это группа связанных файлов (например, файлы кода, ресурсы и графические файлы), образующих приложение. Работа над приложением начинается с создания проекта. Нажмите `Start a new Android Studio project` на странице `Welcome`, или, если открыт проект, — выберите команду меню `File` ▸ `New` ▸ `New Project...`. На экране появляется диалоговое окно `Create New Project` (рис. 2.2).

2.3.3. Диалоговое окно Create New Project

На шаге `Configure your new project` мастера `Create New Project` (рис. 2.2) введите следующую информацию, после чего нажмите `Next`.

1. `Application Name:` — имя приложения. Введите в этом поле строку `Welcome`.
2. `Company Domain:` — доменное имя веб-сайта компании. Мы использовали домен нашего сайта *deitel.com*; в ходе обучения можно использовать имя *example.com*, но его придется изменить, если вы собираетесь распространять свое приложение.
3. `Package Name:` — имя пакета Java для исходного кода приложения. Android и магазин Google Play используют это имя в качестве *уникального идентификатора* приложения, которое должно оставаться постоянным во всех версиях приложения, которые вы будете отправлять в магазин Google Store. Имя пакета обычно начинается с доменного имени вашей компании или

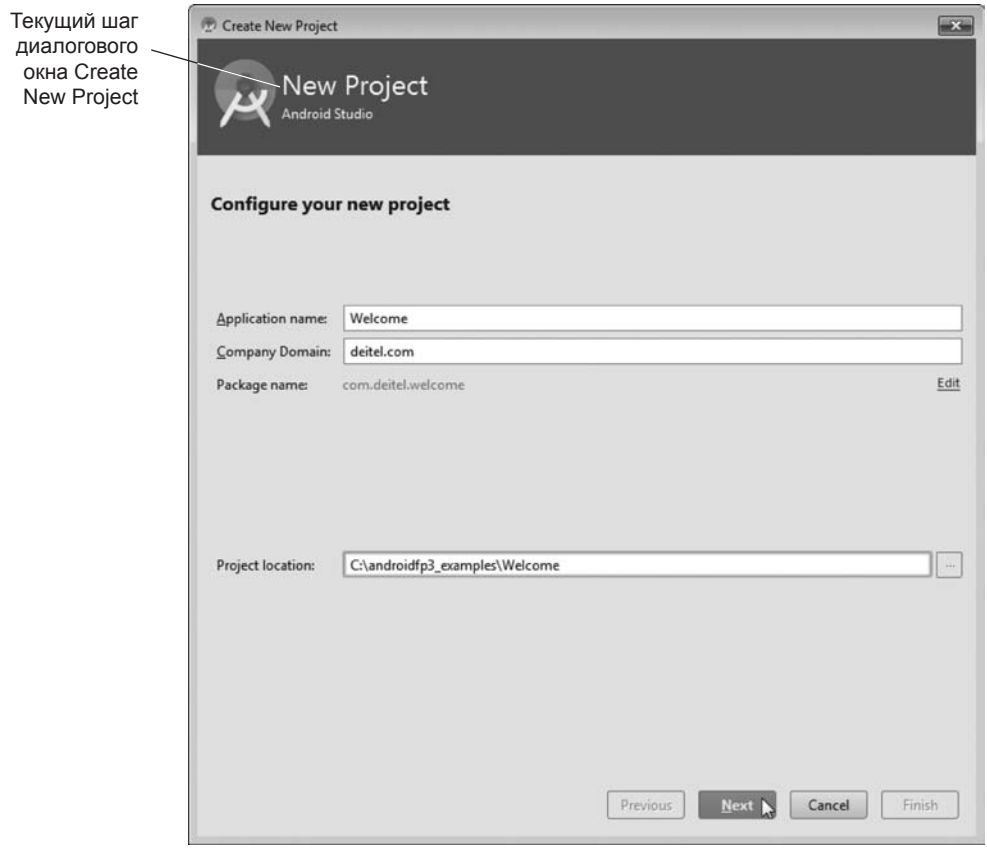


Рис. 2.2. Диалоговое окно Create New Project — шаг New Project

учреждения, записанного *в обратном порядке*. Например, мы используем доменное имя *deitel.com*, поэтому имена наших пакетов начинаются с префикса *com.deitel*. Далее обычно следует имя приложения. По общепринятым соглашениям в имени пакета используются только буквы нижнего регистра без пробелов. По умолчанию IDE выбирает имя пакета на основании текста, вводимого в полях `Application Name` и `Company Domain`. Чтобы изменить имя `Package`, щелкните на ссылке `Edit` справа от сгенерированного имени пакета.

4. `Project Location:` — путь к папке на вашем компьютере, в которой будет храниться проект. По умолчанию Android Studio размещает папки новых проектов во вложенной папке `AndroidStudioProjects` каталога учетной записи пользователя. Имя папки проекта состоит из имени проекта, из которого удаляются пробелы. Вы можете изменить путь к папке проекта; для этого введите путь или щелкните на кнопке (...) справа от поля и выберите папку для хранения проекта. После того как папка будет выбрана, щелкните на кнопке `OK`, а затем перейдите к следующему шагу кнопкой `Next`.

ЗАЩИТА ОТ ОШИБОК 2.1

Если путь к папке для сохранения проекта содержит пробелы, диалоговое окно Create New Project выводит сообщение «Путь содержит пробелы. Это нежелательно из-за возможных проблем на некоторых платформах». Чтобы решить проблему, щелкните на кнопке (…) справа от поля Project location и выберите папку, имя которой не содержит пробелов; в противном случае приложение может не компилироваться или выполняться некорректно.

2.3.4. Шаг Target Android Devices

На шаге `Target Android Devices` диалогового окна `Create New Project` (рис. 2.3):

Рис. 2.3. Диалоговое окно Create New Project — шаг Target Android Devices

1. Установите флажки всех типов устройств Android (`Phone and Tablet`, `TV`, `Wear`, `Android Auto` и `Glass`), которые должны поддерживаться приложением. Для приложения `Welcome` должен быть установлен только флажок `Phone and Tablet`.
2. Выберите значение в раскрывающемся списке `Minimum SDK` для каждого выбранного типа устройств, щелкните на кнопке `Next`. В поле `Minimum SDK` выбирается минимальный уровень Android API, необходимый для запуска приложения. Этот параметр разрешает выполнять приложение на устройствах с заданным уровнем API и выше. Выберите для приложений этой книги значение `API23: Android 6.0 (Marshmallow)` и щелкните на кнопке `Next`. В табл. 2.1 перечислены версии Android SDK и соответствующие им уровни API — другие версии SDK объявлены устаревшими и не должны использоваться. Информацию о процентном соотношении устройств на базе Android, использующих каждую платформу, можно найти по адресу

   http://developer.android.com/about/dashboards/index.html

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 2.1

Меньшие значения Minimum SDK позволят вашему приложению выполняться на большем количестве устройств, например на момент написания книги уровень API 15 обеспечивал возможность выполнения на 95% устройств. Обычно следует ориентироваться на минимальный уровень API, на котором может работать приложение. Позаботьтесь о том, чтобы новые возможности не использовались на старых платформах, если ваше приложение устанавливается на этих платформах.

Таблица 2.1. Версии Android SDK и уровни API (http://developer.android.com/about/dashboards/index.html)

Версия SDK	Уровень API
6.0	23
5.1	22
5.0	21
4.4	19
4.3	18
4.2.x	17
4.1.x	16
4.0.3–4.0.4	15
2.3.3–2.3.7	10
2.2	8

2.3.5. Шаг Add and Activity to Mobile

На шаге `Add an Activity to Mobile` (рис. 2.4) выбирается *шаблон* приложения. Шаблоны предоставляют заготовки для стандартных вариантов дизайна и логики приложения.

Рис. 2.4. Диалоговое окно Create New Project — шаг Add an Activity to Mobile

В табл. 2.2 кратко описаны четыре наиболее популярных шаблона, представленных на рис. 2.4. Для нашего приложения выберите пустой шаблон `Empty Activity` и нажмите `Next`. Этот шаблон определяет приложение с одним экраном, на котором выводится текст «Hello World!». Остальные шаблоны будут использоваться в других главах. Для многоэкранных приложений также можно

определить новый экран, добавив одну из активностей из табл. 2.2 в существующее приложение. Например, в приложении `Flag Quiz` из главы 4 будет добавлена активность `Settings Activity` для экрана, на котором пользователь сможет задать настройки.

Таблица 2.2. Шаблоны Activity

Шаблон	Описание
Blank Activity	Используется для одноэкранных приложений, в которых большая часть графического интерфейса строится разработчиком. Включает панель действий в верхней части приложения, в которой выводится имя приложения, а также могут отображаться элементы управления для взаимодействия с приложением. Также на панели отображается кнопка FloatingActionButton материального дизайна
Fullscreen Activity	Используется для одноэкранных приложений (по аналогии с `Blank Activity`), занимающих весь экран, но с возможностью переключения режима видимости панели состояния устройства и панели действий приложения
Master/Detail Flow	Используется для приложений с главным списком, из которого пользователь может выбрать один вариант для просмотра подробной информации (как во встроенных приложениях `Email` и `Contacts`). Включает основную логику выбора элементов в главном списке и отображения детализированного представления этого элемента. На планшетах главный список и детализация выводятся рядом друг с другом на одном экране. На телефонах главный список выводится на одном экране, а при выборе варианта детализация отображается на отдельном экране

2.3.6. Шаг Customize the Activity

Этот шаг (рис. 2.5) зависит от выбора шаблона на предыдущем шаге. Для шаблона `Empty Activity` он позволяет задать:

- `Activity Name` — имя активности (по умолчанию IDE предлагает имя `MainActivity`). Это имя субкласса `Activity`, управляющего выполнением приложения. Начиная с главы 3 мы будем изменять этот класс для реализации функциональности приложения.
- `Layout Name` — имя файла макета (по умолчанию IDE предлагает имя файла `activity_main`). В файле (с расширением `.xml`) хранится представление графического интерфейса приложения в формате XML. В этой главе мы построим интерфейс (раздел 2.5) визуальными средствами.

Оставьте настройки по умолчанию и нажмите `Finish`, чтобы завершить создание проекта.

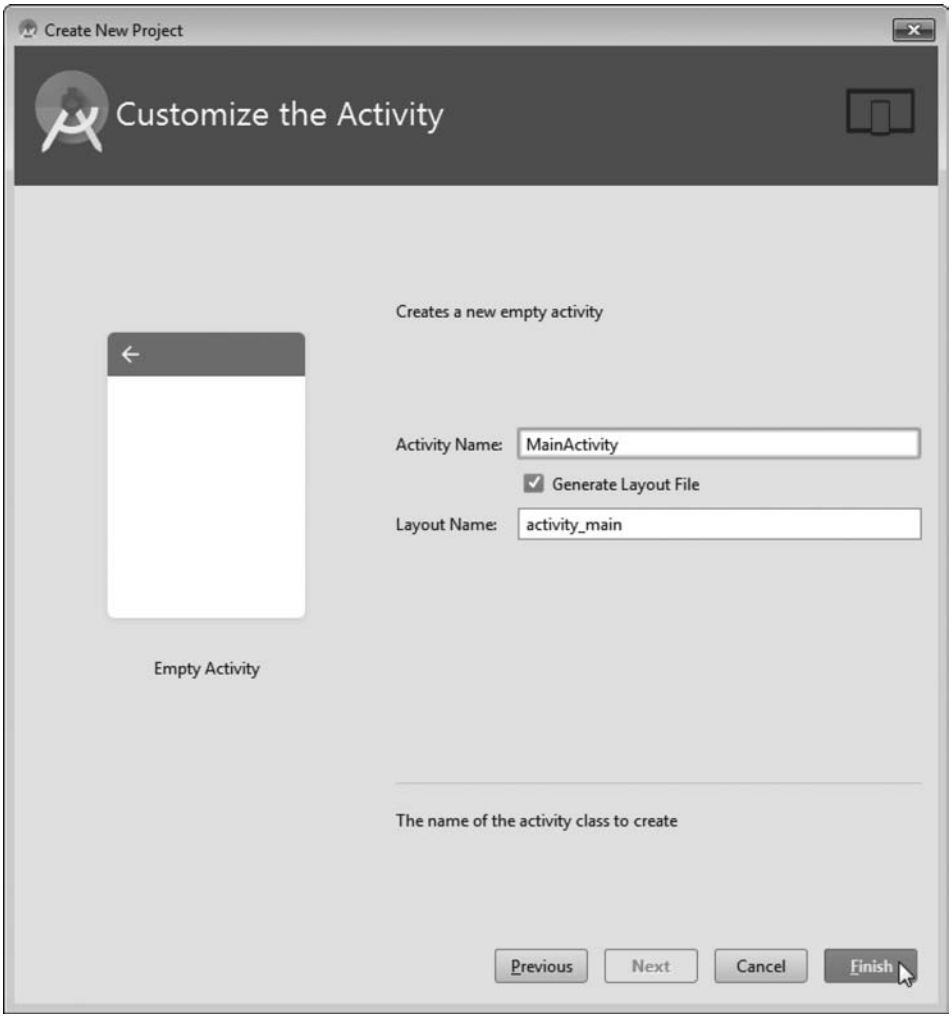


Рис. 2.5. Диалоговое окно Create New Project — шаг Customize the Activity

2.4. Окно Android Studio

При завершении создания проекта IDE открывает файлы `MainActivity.java` и `activity_main.xml`. Закройте файл `MainActivity.java`, чтобы окно IDE выглядело так, как показано на рис. 2.6. В IDE отображается макетный редактор для построения графического интерфейса приложения. В этой главе мы ограничимся средствами, необходимыми для построения приложения `Welcome`. Другие возможности IDE будут представлены позже.

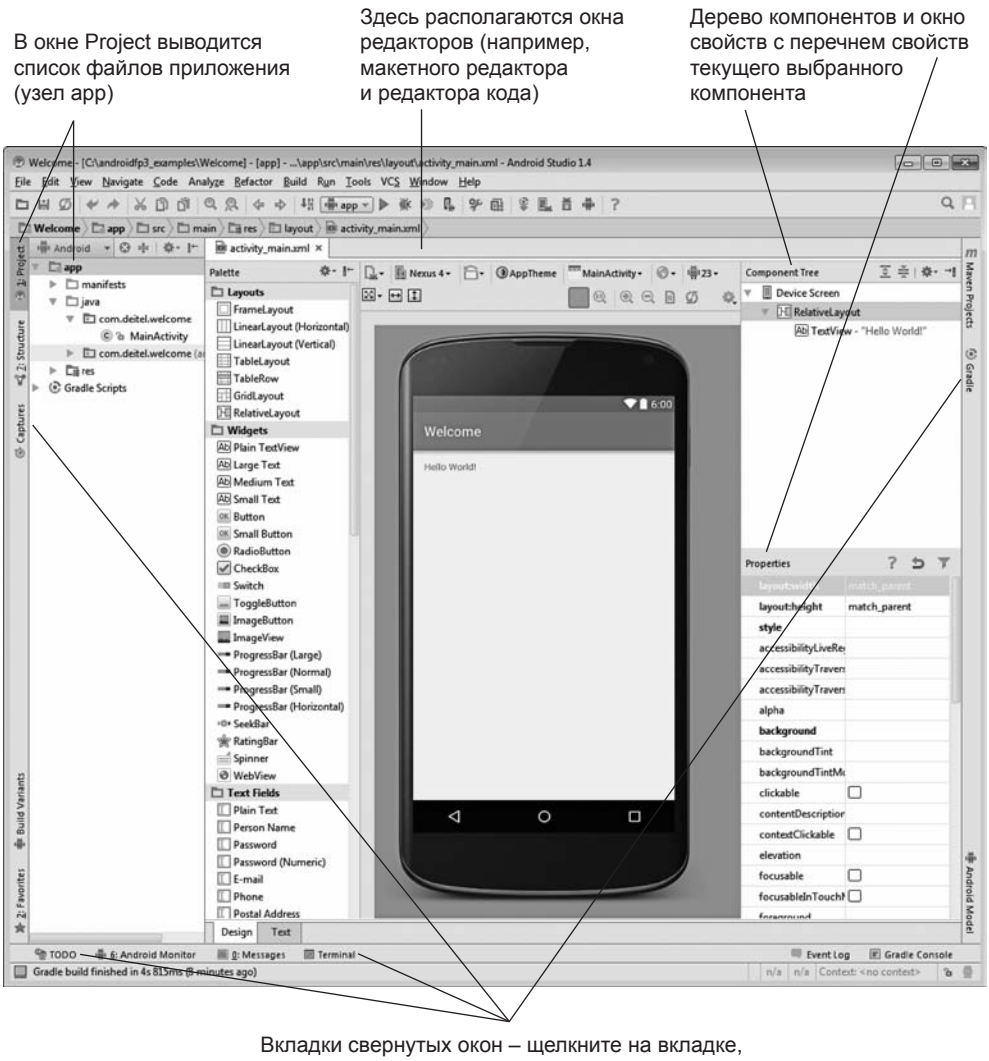


Рис. 2.6. Проект Welcome в Android Studio

2.4.1. Окно Project

Окно `Project` предоставляет доступ ко всем файлам проекта. В IDE можно одновременно открыть несколько проектов — каждый будет представлен узлом верхнего уровня. На рис. 2.7 изображен проект приложения `Welcome`

в окне `Project` с раскрытым узлом `res` и его вложенной папкой `layout`. Папка `app` содержит файлы, используемые при создании графического интерфейса и логики приложения. Внутри нее все содержимое делится на вложенные папки с файлами. В этой главе используются только файлы из папки `res`, которая будет рассмотрена в разделе 2.4.4; другие папки будут описаны в последующих главах, когда мы начнем их использовать.

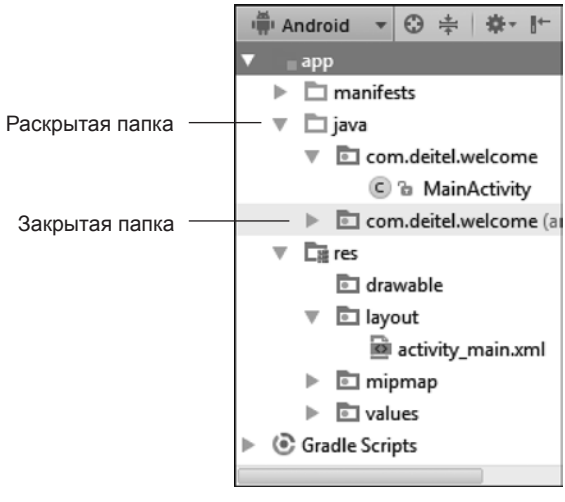


Рис. 2.7. Окно Project

2.4.2. Окна редактора

Справа от окна `Project` на рис. 2.6 расположено окно макетного редактора. Если сделать двойной щелчок на файле в окне `Project`, его содержимое открывается в окне соответствующего редактора (в зависимости от типа файла). Для файлов Java открывается редактор исходного кода Java. Для файла XML, представляющего графический интерфейс (такого, как `activity_main.xml`), по умолчанию выбирается вкладка `Design` макетного редактора; вы можете щелкнуть на вкладке `Text`, чтобы просматривать разметку XML параллельно с предварительным изображением интерфейса (если оно не отображается, вызовите его командой `View` ▸ `Tool Windows` ▸ `Preview`). Для других файлов XML отображается специализированный или текстовый редактор XML (в зависимости от назначения файлов XML). Редакторы кода Java и XML позволяют быстро создавать правильный код благодаря *автозавершению* — нажатие клавиши Enter (или Return) во время ввода обеспечивает автоматическое завершение элемента кода Java или имени текущего выделенного атрибута, значения или элемента XML.

2.4.3. Окно Component Tree

Когда макетный редактор открыт в режиме `Design`, в правой части IDE выводится окно `Component Tree` с деревом компонентов (см. рис. 2.6). В этом окне отображаются макеты и представления (компоненты), образующие графический интерфейс, и отношения «родитель—потомок» между ними, например макет (родитель) может содержать много вложенных представлений (потомков), включая другие макеты.

2.4.4. Файлы ресурсов приложения

Файлы макетов — такие как `activity_main.xml`, — считаются *ресурсами приложения* и хранятся в папке `res` проекта. В этой папке находятся вложенные папки для разных типов ресурсов. В табл. 2.3 перечислены папки, используемые в проекте Welcome; другие папки (`menu`, `animator`, `anim`, `color`, `mipmap`, `raw` и `xml`) будут представлены позже, когда они будут использоваться в книге.

Таблица 2.3. Папки ресурсов, используемые в этой главе

Папка	Описание
drawable	Папки с именами, начинающимися с drawable, обычно содержат изображения. В них также могут храниться файлы XML, описывающие геометрические фигуры и другие типы экранных объектов (например, изображения, представляющие нажатое и ненажатое состояние кнопки)
layout	Папки с именами, начинающимися с layout, содержат файлы XML с описанием графического интерфейса, например файл activity_main.xml
values	Папки с именами, начинающимися с values, содержат файлы со значениями массивов (arrays.xml), цветов (colors.xml), размеров (dimen.xml; такие значения, как ширина, высота и размеры шрифтов), строки (strings.xml) и стили (styles.xml). Эти имена файлов используются по общепринятым соглашениям, но не являются обязательными: на самом деле все ресурсы этих типов можно разместить в одном файле. Все эти жестко фиксированные данные рекомендуется определять в виде ресурсов, чтобы их можно было изменить без модификации исходного кода Java. Например, если ссылки на некую метрику встречаются в нескольких местах кода, проще один раз отредактировать файл ресурсов, чем искать все вхождения фиксированного значения в исходных файлах Java

2.4.5. Макетный редактор

При создании проекта IDE открывает файл `activity_main.xml` приложения в макетном редакторе (рис. 2.8). Также макетный редактор можно открыть двойным щелчком на файле `activity_main.xml` в папке `res/layout` проекта приложения.

Рис. 2.8. Графический интерфейс приложения по умолчанию в макетном редакторе

Выбор типа экрана

Приложения Android могут работать на многих видах устройств. В этой главе мы спроектируем графический интерфейс для телефона на базе Android. Как упоминалось в разделе «Подготовка», для этой цели будет использоваться виртуальное устройство (AVD), эмулирующее телефон Google Nexus 6. Макетный

редактор поддерживает многочисленные конфигурации устройств для разных размеров и разрешений экранов, которые могут использоваться для проектирования интерфейса. В этой главе мы воспользуемся предопределенным эмулятором **Nexus 6**, который выбирается в списке устройств на рис. 2.8. Это не означает, что приложение может выполняться только на устройстве Nexus 6, просто этот дизайн используется для устройств, сходных по размеру экрана и разрешению с Nexus 6. В следующих главах вы узнаете, как спроектировать графический интерфейс, нормально масштабируемый на широком диапазоне устройств.

2.4.6. Графический интерфейс по умолчанию

Графический интерфейс по умолчанию (рис. 2.7) для шаблона `Blank Page` состоит из компонента `RelativeLayout` со светло-серым фоном (определяется темой, выбранной при создании проекта) и компонентом `TextView` с текстом `"Hello world!"`. Компонент `RelativeLayout` размещает компоненты графического интерфейса *относительно друг друга или самого макета*, например при помощи `RelativeLayout` можно указать, что компонент должен находиться ниже другого компонента, с горизонтальным выравниванием по центру. В приложении `Welcome` компонент `RelativeLayout` будет заменен вертикальным компонентом `LinearLayout`, который размещает текст и изображение по вертикали так, что каждый компонент занимает половину высоты. Компонент `TextView` предназначен для вывода текста, а компонент `ImageView` — для вывода графики. Все упомянутые компоненты будут более подробно рассмотрены в разделе 2.5.

2.4.7. Разметка XML для графического интерфейса по умолчанию

Как упоминалось ранее, в XML-файле с именем `activity_main.xml` хранится описание графического интерфейса в формате XML (листинг 2.1). Отступы в сгенерированной разметке были сокращены для экономии места. Мы отредактируем эту разметку, чтобы заменить компонент `RelativeLayout` на `LinearLayout`.

Листинг 2.1. Исходное содержимое файла activity_main.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 3     xmlns:tools="http://schemas.android.com/tools"
 4     android:layout_width="match_parent"
 5     android:layout_height="match_parent"
 6     android:paddingBottom="@dimen/activity_vertical_margin"
 7     android:paddingLeft="@dimen/activity_horizontal_margin"
 8     android:paddingRight="@dimen/activity_horizontal_margin"
 9     android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context=".MainActivity">
```

```
11
12     <TextView
13         android:layout_width="wrap_content"
14         android:layout_height="wrap_content"
15         android:text="@string/hello_world" />
16
17 </RelativeLayout>
```

Значения атрибутов, начинающиеся с `@`, такие как

```
@dimen/activity_vertical_margin
```

в строке 6, представляют *ресурсы* со значениями, определяемыми в других файлах. По умолчанию редактор XML отображает фактическое значение ресурса (`16dp` для ресурса в строке 6) и выделяет его светло-зеленым фоном (или светло-серым, если вы используете «темную» тему Android Studio).

Это позволяет разработчику видеть фактическое значение ресурса, используемое в конкретном контексте. Если щелкнуть на фактическом значении ресурса (`16dp` для `@dimen/activity_vertical_margin`), редактор вместо этого выводит соответствующее имя ресурса.

2.5. Построение графического интерфейса приложения

А теперь создадим графический интерфейс пользователя для приложения `Welcome`. Макетный редактор (Layout Editor) позволяет создать графический интерфейс пользователя путем перетаскивания в окно приложения компонентов GUI, таких как `Button`, `TextView`, `ImageView` и др. По умолчанию описание макета для шаблона `Empty App` хранится в XML-файле с именем `activity_main.xml` в папке `res/layout`. В этой главе мы воспользуемся макетным редактором и окном `Component Tree` для построения приложения. Разметка XML в файле `activity_main.xml` будет отредактирована только для того, чтобы изменить способ размещения компонентов `TextView` и `ImageView`, используемых в приложении.

2.5.1. Добавление изображения в проект

Для нашего приложения в проект необходимо включить «фирменного» жука Deitel[1] (`bug.png`). Эти изображения хранятся в папке `images/Welcome` примеров

[1] Прежде чем использовать графику в приложении, убедитесь в том, что у вас имеются необходимые лицензионные права. Некоторые правила лицензирования графики требуют оплаты, в других случаях предоставляется лицензия для свободного распространения или лицензия Creative Commons (*creativecommons.org*).

книги. Имена файлов графических ресурсов (а также всех остальных ресурсов, о которых вы узнаете в последующих главах) должны записываться *в нижнем регистре*.

Папки drawable

Так как экраны устройств Android обладают разными размерами, разрешением и плотностью пикселов (DPI, Dot Per Inch), разработчик обычно предоставляет изображения с разными разрешениями, а операционная система выбирает графику на основании плотности пикселов устройства. По этой причине папка `res` вашего проекта содержит несколько вложенных папок, имена которых начинаются с префикса *drawable*. Например, изображения для устройств с плотностью пикселов, близкой к плотности экрана телефона Google Nexus 6 (560 dpi) для нашего AVD, будут храниться в папке `drawable-xxxhdpi` (табл. 2.4). Изображения для устройств с более низкой плотностью пикселов размещаются в других папках `drawable` — обычно в папке, наиболее точно описывающей плотность пикселов реального устройства.

Таблица 2.4. Плотности пикселов устройств Android

Плотность	Описание
drawable-ldpi	Низкая плотность — приблизительно 120 точек на дюйм
drawable-mdpi	Средняя плотность — приблизительно 160 точек на дюйм
drawable-hdpi	Высокая плотность — приблизительно 240 точек на дюйм
drawable-xhdpi	Очень высокая плотность — приблизительно 320 точек на дюйм
drawable-xxhdpi	Сверхвысокая плотность — приблизительно 480 точек на дюйм
drawable-xxxhdpi	Ультравысокая плотность — приблизительно 640 точек на дюйм

Android Studio отображает только одну папку `drawable` с графическими ресурсами приложения, даже если проект содержит ресурсы для разных значений плотности. Для ресурсов, хранящихся в папке `drawable-xxxhdpi`, Android Studio отображает в папке `drawable` проекта строку

```
имя_файла.xml (xxxhdpi)
```

В этом приложении мы предоставляем только одну версию каждого изображения. Если Android не находит изображения в подходящей папке `drawable`, то версия изображения из другой папки масштабируется до нужной плотности (вверх или вниз, как потребуется). По умолчанию Android Studio создает только папку `drawable` без уточняющего суффикса `dpi`; она будет использоваться

в первом приложении. За подробной информацией о поддержке разных видов и размеров экранов в Android обращайтесь по адресу

http://developer.android.com/guide/practices/screens_support.html

СОВЕТ ПО ОФОРМЛЕНИЮ 2.1

Изображения низкого разрешения плохо масштабируются. Чтобы графика выглядела нормально, устройству с высокой плотностью пикселов нужны изображения с более высоким разрешением, чем устройству с низкой плотностью.

Добавление файла bug.png в проект

Чтобы добавить изображения в проект, выполните следующие действия.

1. В окне `Project` откройте папку `res` проекта.
2. Найдите и откройте папку `Welcome` в папке `images` в вашей файловой системе.
3. Скопируйте файл `bug.png`. В окне `Project` Android Studio выделите папку `res/drawable` и вставьте файл в эту папку.
4. В открывшемся диалоговом окне `Copy` щелкните на кнопке `OK`.

Теперь скопированный файл можно использовать в приложении.

2.5.2. Добавление значка приложения

Когда ваше приложение устанавливается на устройстве, его значок и имя появляются вместе с остальными установленными приложениями в *лаунчере* — оболочке, вызываемой значком на главном экране устройства. Чтобы добавить значок приложения, щелкните правой кнопкой мыши на папке `res` и выберите команду `New` ▸ `Image Asset`. На экране появляется окно `Asset Studio` (рис. 2.9), в котором можно выбрать значок приложения из существующих изображений, графических заготовок или текста.

В нашем приложении используется изображение `DeitelOrange.png` из папки `images`. Чтобы использовать его, выполните следующие действия.

1. Щелкните на кнопке (…) справа от поля `Image File:`.
2. Перейдите в папку `images` в папке примеров книги.
3. Выделите файл `DeitelOrange.png` и щелкните на кнопке `OK`. Предварительные версии масштабированных изображений отображаются в области `Preview` диалогового окна.
4. Щелкните на кнопке `Next`, затем на кнопке `Finish`.

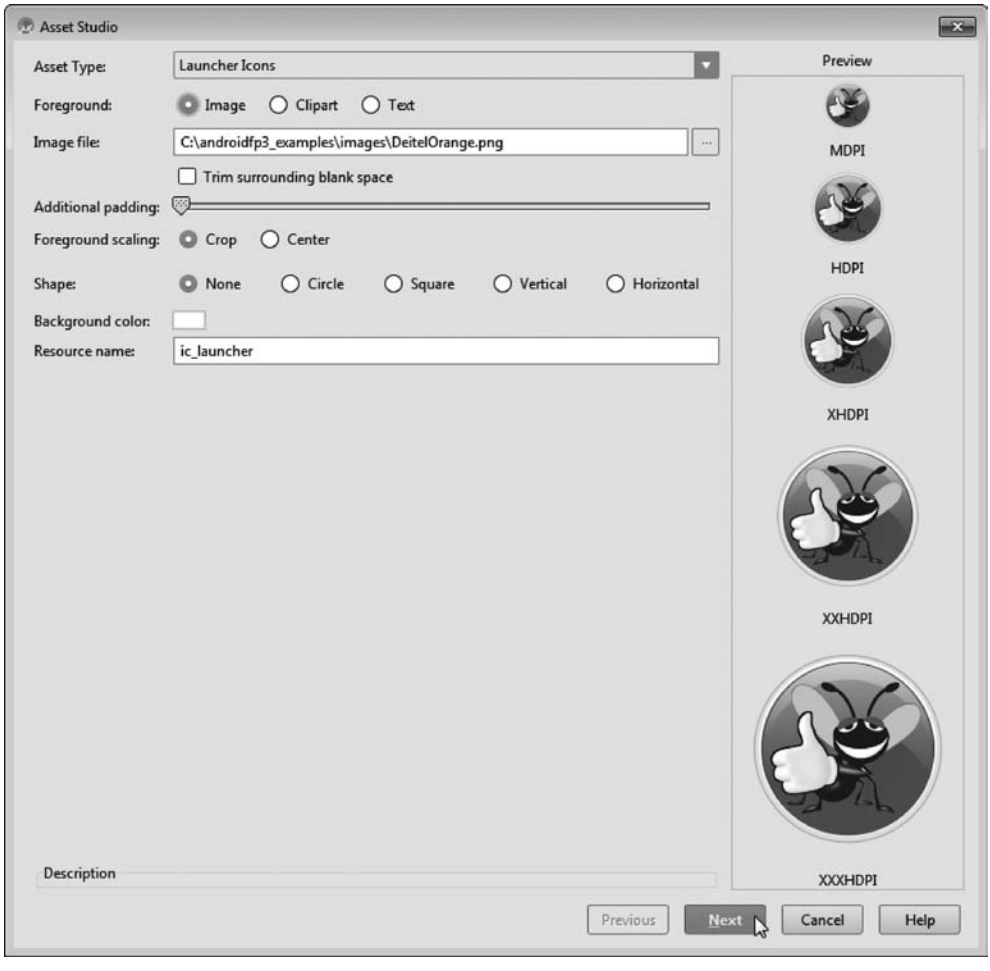


Рис. 2.9. Настройка значка приложения в окне Asset Studio

IDE создает несколько масштабированных версий изображения с именем `ic_launcher.png`, и помещает их в папки `res/mipmap` проекта[1]. Папки `mipmap` похожи на папки `drawable`, но они предназначены для значка приложения. При отправке приложения в Google Play можно приложить несколько версий приложения для разных размеров и разрешений экрана. Все изображения в папках `mipmap` включаются в каждую версию приложения, тогда как лишние папки `drawable`

[1] За информацией о происхождении термина обращайтесь по адресу *https://en.wikipedia.org/wiki/Mipmap*.

для некоторых плотностей пикселов можно исключить, чтобы уменьшить размер устанавливаемого файла для конкретного устройства.

СОВЕТ ПО ОФОРМЛЕНИЮ 2.2

Изображения не всегда хорошо масштабируются. Для приложений, которые вы собираетесь продавать через Google Play, стоит обратиться к профессиональному дизайнеру за версиями значков для разных разрешений. В главе 10 рассматривается процесс отправки приложений в магазин Google Play и приводится список нескольких компаний, предоставляющих услуги по дизайну значков (бесплатные или платные).

2.5.3. Замена RelativeLayout на LinearLayout

Когда вы открываете файл с макетом XML, в макетном редакторе появляется графический интерфейс макета, а представления макета и иерархические отношения между ними отображаются в окне `Component Tree` (рис. 2.10). Чтобы настроить параметры макета или представления, можно выделить его в макетном редакторе или в окне `Component Tree`, а затем в окне свойств под окном `Component Tree` задать значения свойств без непосредственного редактирования XML. При построении и изменении более сложных макетов часто бывает проще напрямую работать с окном `Component Tree`.

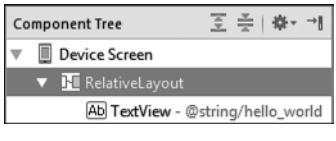


Рис. 2.10. Иерархическое представление графического интерфейса в окне Component Tree

Для некоторых изменений графического интерфейса — например, для замены стандартного компонента `RelativeLayout` на `LinearLayout` — приходится редактировать разметку XML макета. (Не исключено, что в будущем Google усовершенствует возможности макетного редактора.) Для этого:

1. Щелкните на вкладке `Text` в нижней части макетного редактора, чтобы переключиться из режима `Design` на разметку XML макета.
2. В верхней части разметки (строка 2 в листинге 2.1) сделайте двойной щелчок на имени элемента XML `RelativeLayout`, чтобы выделить его, и начинайте вводить строку `LinearLayout`.

3. Во время ввода строки 2 IDE синхронно изменяет соответствующий тег XML (строка 17 в листинге 2.1), а на экране появляется окно автозавершения с именами элементов, начинающихся с введенных букв. Как только в окне появится и будет выделено имя `LinearLayout`, нажмите Enter (или Return). Android Studio автоматически завершает редактирование.
4. Сохраните изменения и вернитесь на вкладку `Design` макетного редактора.

Окно `Component Tree` должно выглядеть так, как показано на рис. 2.11.

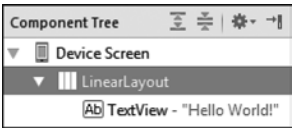


Рис. 2.11. Дерево Component Tree после замены RelativeLayout на LinearLayout

2.5.4. Изменение свойств id и orientation компонента LinearLayout

В этом разделе мы настроим значения свойств `LinearLayout`. Как правило, каждому макету и компоненту следует присвоить удобное имя, по которому вы легко узнаете каждое представление в окне `Component Tree` и сможете работать с представлениями на программном уровне (как это будет сделано в последующих приложениях).

Когда графический интерфейс отображается в макетном редакторе, вы можете воспользоваться окном свойств под окном `Component Tree` (см. рис. 2.6) для настройки свойств выделенного представления. Наиболее часто используемые свойства представления также можно изменить другим, более быстрым способом (как будет сделано в нашем разделе) — двойным щелчком на представлении. В макетном редакторе открывается небольшое диалоговое окно, в котором можно задать свойство `id` и другие свойства, зависящие от конкретного представления:

- Для `LinearLayout` можно указать, как должны располагаться дочерние представления макета — по горизонтали или по вертикали (`orientation`).
- Для `TextView` можно задать выводимый текст (`text`).
- Для `ImageView` можно задать источник отображаемых графических данных (`src`).

Чтобы изменить свойство `orientation` компонента `LinearLayout`, сделайте двойной щелчок на белом фоне виртуального экрана в макетном редакторе. Открывается диалоговое окно с основными свойствами `LinearLayout`; выберите

в раскрывающемся списке `orientation:` значение `vertical` (рис. 2.12). Значение свойства изменяется, а диалоговое окно закрывается. Чтобы задать имя представления, измените значение свойства `id`, которое задается в разметке XML макета атрибутом `android:id`. Сделайте двойной щелчок на белом фоне виртуального экрана, введите имя `welcomeLinearLayout` в поле `id:` и нажмите клавишу Enter (или Return), чтобы задать значение свойства и закрыть диалоговое окно.

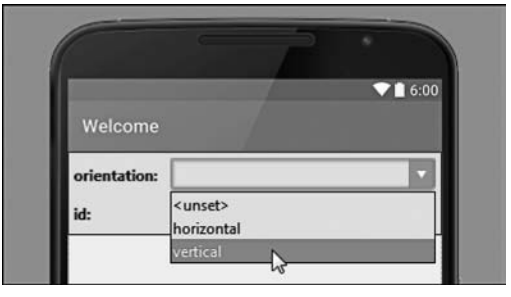


Рис. 2.12. Назначение свойства orientation компонента LinearLayout

Представление свойства id в разметке XML

В разметке XML макета (которую можно просмотреть при помощи вкладки `Text` в нижней части макетного редактора) свойство `android:id` компонента `LinearLayout` имеет следующее значение:

```
@+id/welcomeLinearLayout
```

Знак `+` в синтаксисе `@+id` указывает на создание нового свойства `id` с идентификатором, следующим за косой чертой (`/`). В некоторых случаях разметка XML содержит тот же синтаксис без знака `+` для ссылки на существующее представление — например, для определения отношений между представлениями в `RelativeLayout`.

2.5.5. Изменение свойств id и text компонента TextView

Сгенерированный по умолчанию графический интерфейс приложения `Welcome` уже содержит компонент `TextView`, поэтому мы просто изменим его свойства.

Настройка свойства id компонента TextView

Сделайте двойной щелчок на компоненте `TextView` в макетном редакторе, после чего в открывшемся диалоговом окне задайте свойству `id:` значение `welcomeTextView` и нажмите клавишу Enter (или Return).

Настройка свойства text компонента TextView с использованием строкового ресурса

Документация Android по ресурсам приложений:

http://developer.android.com/guide/topics/resources/index.html

рекомендует размещать строки, строковые массивы, изображения, цвета, размеры шрифтов, метрики и другие ресурсы приложений в файлах XML во вложенных папках папки `res` проекта, что позволяет управлять этими ресурсами из кода приложения Java. Этот прием называется *внешним размещением ресурсов*. Например, при внешнем размещении цветов для обновления всех компонентов, использующих один и тот же цвет, достаточно изменить значение цвета в централизованном файле ресурсов.

Если вы захотите локализовать свое приложение на нескольких языках, хранение строк *отдельно* от кода приложения упростит их изменение. В папке `res` проекта находится вложенная папка `values` с файлом `strings.xml`, в которой хранятся строки приложения по умолчанию — англоязычные для нашего приложения. Чтобы предоставить набор локализованных строк для других языков, можно создать отдельную папку для каждого языка (как это будет сделано в разделе 2.8).

Чтобы задать свойство `text` компонента `TextView`, создайте новый строковый ресурс в файле `strings.xml`.

1. Либо сделайте двойной щелчок на компоненте `welcomeTextView` в макетном редакторе, либо выделите компонент `welcomeTextView` и найдите его свойство `text` в окне свойств.
2. Щелкните на кнопке (...) справа от значения свойства, чтобы открыть диалоговое окно `Resources`.
3. В диалоговом окне `Resources` щелкните на кнопке `New Resource`, выберите вариант `New String Value...` для отображения диалогового окна `New String Value Resource` и заполните поля `Resource Name:` и `Resource value:` так, как показано на рис. 2.13. Оставьте остальные значения без изменений (они будут рассмотрены в последующих разделах и главах) и щелкните на кнопке `OK`, чтобы создать новый строковый ресурс с именем `welcome` и задать его значением свойства `text` компонента `TextView`.

Свойство `text` в окне свойств должно выглядеть так, как показано на рис. 2.14. Префикс `@string/` означает, что значение свойства `text` берется из строкового ресурса, а имя `welcome` определяет конкретный строковый ресурс. По умолчанию ресурс размещается в файле `strings.xml` (из папки `res/values` проекта).

Рис. 2.13. Диалоговое окно New String Value Resource

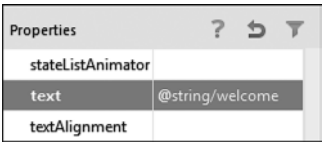


Рис. 2.14. Окно свойств после изменения свойства text компонента TextView

2.5.6. Настройка свойства textSize компонента TextView — пикселы, независимые от плотности и масштабирования

Размеры компонентов GUI и текста на экране Android могут определяться с помощью различных единиц измерения (табл. 2.5). Документация, описывающая различные размеры экранов, находится на веб-сайте по адресу

developer.android.com/guide/practices/screens_support.html

и рекомендует для определения размеров компонентов GUI и других экранных элементов использовать *пикселы, независимые от плотности*, а размеры шрифтов задавать с помощью *пикселов, независимых от масштабирования*.

Таблица 2.5. Единицы измерения

Единица измерения	Описание
px	Пиксел
dp или dip	Пиксел, независимый от плотности
sp	Пиксел, независимый от масштабирования
in	Дюймы
mm	Миллиметры

Задание размеров в пикселах, независимых от плотности (dp или dip), позволяет платформе Android автоматически *масштабировать* графический интерфейс пользователя в зависимости от плотности пикселов экрана физического устройства. Размер пиксела, независимого от плотности, эквивалентен размеру физического пиксела на экране с разрешением 160 dpi (точек на дюйм). На экране с разрешением 240 dpi размер пиксела, независимого от плотности, будет масштабироваться с коэффициентом 240/160 (то есть 1,5). Таким образом, компонент, размер которого составляет 100 пикселов, независимых от плотности, будет масштабирован до размера в 150 *физических пикселов* на таком экране. На экране с разрешением 120 точек на дюйм каждый независимый от плотности пиксел масштабируется с коэффициентом 120/160 (то есть 0,75). Значит, 100 независимых от плотностей пикселов превратятся на таком экране в 75 физических пикселов. Пикселы, *независимые от масштабирования*, масштабируются так же, как и пикселы, независимые от плотности, но их масштаб зависит также и от *предпочитаемого размера шрифта*, выбираемого пользователем (в настройках устройства).

Создание ресурса для размера шрифта на телефоне

А теперь увеличим размер шрифта для компонента `TextView`. Чтобы изменить размер шрифта:

1. Выделите компонент `welcomeTextView` в макетном редакторе.
2. Найдите свойство `textSize` и щелкните в правом столбце, чтобы появилась кнопка с многоточием (…). Щелкните на кнопке, чтобы открыть диалоговое окно `Resources`.
3. В диалоговом окне `Resources` щелкните на кнопке `New Resource` и выберите вариант `New Dimension Value`…. Открывается диалоговое окно `New Dimension Value Resource`.
4. В открывшемся диалоговом окне введите в поле `Resource name` имя ресурса `welcome_textsize`, а в поле `Resource value` — имя ресурса `40sp`. Щелкните на кнопке `OK`, чтобы закрыть диалоговое окно и вернуться к окну `Resources`. Буквы *sp* в значении *40sp* означают, что речь идет о пикселах, *независимых от масштабирования*. Буквы *dp* в значении (например, `10dp`) обозначают *пикселы, независимые от плотности*. Мы используем значение `40sp` для отображения текста на телефоне.

Теперь значение свойства `textSize` в окне свойств выглядит так:

```
@dimen/welcome_textsize
```

Префикс `@dimen/` означает, что значение свойства `textSize` является ресурсом размера (метрики), а имя `welcome_textsize` определяет конкретный строковый

ресурс. По умолчанию ресурс размещается в файле `dimens.xml` (из папки `res/values` проекта).

Создание ресурса для размера шрифта на планшете

Размер шрифта `40sp` хорошо подходит для устройств с экраном, как у телефона, но для планшетов его недостаточно. Android может автоматически выбирать разные ресурсы в зависимости от размера устройства, ориентации, плотности пикселов, языков, локальных контекстов и т. д. Чтобы задать отдельный размер шрифта для устройства с увеличенным экраном (например, для планшета):

1. Снова откройте диалоговое окно `New Dimension Value Resource` (см. выше).
2. Введите в поле `Resource name` имя `welcome_textsize` (имена ресурсов должны совпадать, чтобы система Android автоматически выбирала нужные значения ресурсов), а в поле `Resource value` — значение `80sp`.
3. Создайте новую папку `values` для устройств с большим экраном (например, планшетов), высота и ширина которого не менее 600dp. В диалоговом окне `New Dimension Value Resource` снимите флажок `values` и щелкните на кнопке `Add` (➕), чтобы открыть диалоговое окно `New Resource Directory`. В списке `Available qualifiers` выберите пункт `Screen Width` и щелкните на кнопке `>>`, чтобы добавить квалификатор ширины экрана в список выбранных квалификаторов `Chosen qualifiers`. Введите `600` в поле `Screen width`.
4. Добавьте квалификатор `Screen Height` в список `Chosen qualifiers` и введите в поле `Screen height` значение `600`.
5. Щелкните на кнопке `OK`, чтобы создать новую папку ресурсов с именем `values-xlarge`.
6. В диалоговом окне `New Dimension Value Resource` установите флажок `values-w600dp-h600dp` и щелкните на кнопке `OK`. В файле `dimens.xml`, который сохраняется в папке `res/values-w600dp-h600dp`, создается еще один ресурс `welcome_textsize`. Android использует этот ресурс для устройств с шириной и высотой экрана не менее 600dp (то есть для большинства планшетов Android). В Android Studio новый файл ресурсов `dimens.xml` отображается в узле `res/values/dimens.xml` в виде `dimens.xml (w600dp-h600dp)`.

2.5.7. Настройка свойства textColor компонента TextView

Если в приложении должны использоваться нестандартные цвета, рекомендации Google Material Design предлагают использовать цвета из палитры Material Design по адресу

http://www.google.com/design/spec/style/color.html

Цвета задаются в формате RGB или ARGB. Значение RGB состоит из целых значений в диапазоне 0–255, определяющих интенсивность трех цветовых составляющих: красной, зеленой и синей. Пользовательские цвета определяются в *шестнадцатеричном* формате, так что составляющие RGB лежат в диапазоне 00–FF (или 0–255 в десятичной записи).

Android также поддерживает составляющую альфа-канала (уровня прозрачности) в диапазоне от 0 (полная прозрачность) до 255 (полная непрозрачность). Чтобы использовать альфа-канал, задайте цвет в формате *#AARRGGBB*, где первые две шестнадцатеричные цифры (*AA*) представляют альфа-канал.

Если обе цифры каждой цветовой составляющей совпадают, допускается использование сокращенных форматов *#RGB* или *#ARGB*. Например, запись `#9AC` интерпретируется как `#99AACC`, а `#F9AC` — как `#FF99AACC`.

Чтобы задать свойству `textColor` компонента `TextView` новый цветовой ресурс:

1. В окне свойств щелкните на кнопке с многоточием (…), чтобы вызвать диалоговое окно `Resources`. Щелкните на кнопке `New Resource` и выберите `New Color Value`….
2. В диалоговом окне `New Color Value Resource` введите в поле `Resource name` имя `welcome_text_color`, а в поле `Resource value` — значение `#2196F3` (рис. 2.15). Щелкните на кнопке `OK`.

Рис. 2.15. Создание нового ресурса цвета для свойства textColor компонента TextView

2.5.8. Настройка свойства gravity компонента TextView

Чтобы многострочный текст в `TextView` выравнивался по центру, задайте его свойству `gravity` значение `center`. Для этого разверните узел свойства и установите флажок `center` (рис. 2.16).

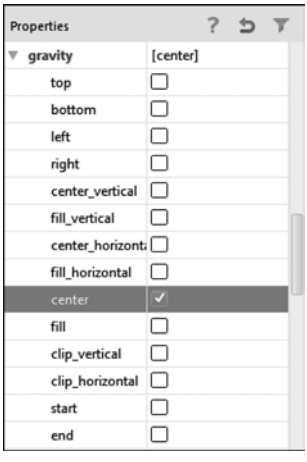


Рис. 2.16. Возможные значения свойства gravity компонента TextView

2.5.9. Настройка свойства layout:gravity компонента TextView

Каждое представление, включаемое в макет, обладает различными свойствами для настройки размера и его позиции в макете. Если выделить представление в макетном редакторе или окне `Component Tree`, в окне свойств выводится перечень свойств: сначала идут свойства `layout` и `style`, далее идут специфические свойства представления, упорядоченные по алфавиту (рис. 2.17).

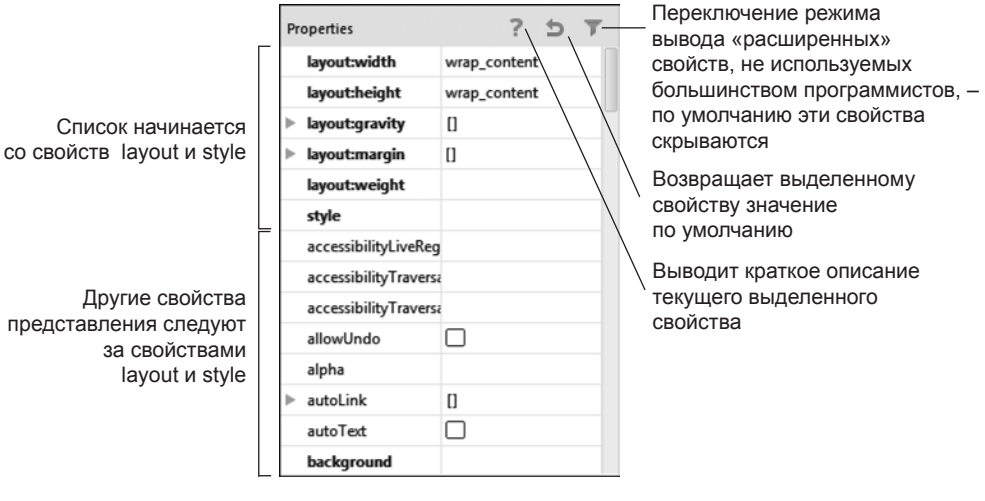


Рис. 2.17. Окно свойств; в начале списка идут свойства layout и style

В этом приложении компонент `TextView` должен выравниваться по горизонтали внутри `LinearLayout`. Для этого следует настроить свойство `layout:gravity` для горизонтального выравнивания по центру.

1. Выделите компонент `TextView`, раскройте узел `layout:gravity` в окне свойств.
2. Щелкните на поле значения справа от значения `center`, выберите вариант `horizontal` (рис. 2.18).

Рис. 2.18. Настройка свойства layout:gravity компонента TextView

В файле разметки XML свойства `layout` представлены атрибутами, имена которых начинаются с `layout_`. Упомянутое свойство `layout:gravity` в XML выглядит так:

```
android:layout_gravity="center_horizontal"
```

2.5.10. Настройка свойства layout:weight компонента TextView

Компонент `LinearLayout` может пропорционально изменять размеры своих дочерних элементов на основании весов `layout:weight`, определяющих размер представления относительно других представлений макета. По умолчанию у всех представлений, добавляемых в `LinearLayout`, значение `layout:weight` равно 0; это означает, что пропорциональное изменение размеров не применяется.

В нашем приложении каждый из компонентов `TextView` и `ImageView` должен занимать половину вертикального пространства `LinearLayout`. Для этого свойствам `layout:weight` обоих представлений задаются одинаковые значения. `LinearLayout` распределяет пространство между представлениями на основании отношения `layout:weight` каждого представления к сумме всех весов. В нашем

приложении свойству `layout:weight` обоих компонентов `TextView` и `ImageView` задается значение 1 (раздел 2.5.11) — сумма `layout:weight` равна 2, поэтому каждое представление занимает 1/2 высоты контейнера.

Если вы хотите, чтобы компонент `TextView` занимал треть высоты `LinearLayout`, задайте его свойству `layout:weight` значение 1, а свойству `layout:weight` компонента `ImageView` значение 2. В этом случае сумма весов `layout:weight` равна 3, так что `TextView` будет занимать 1/3 высоты, а `ImageView` — 2/3 высоты.

Задайте свойству `layout:weight` компонента `TextView` значение 1. Макетный редактор выводит слева от свойства `layout:height` изображение лампочки (); если это не происходит немедленно, щелкните на свойстве `layout:height` в окне свойств. Эти значки (они генерируются инструментом IDE, называемым *Android Lint*) предупреждают о потенциальных проблемах и помогают исправить их. Если щелкнуть на лампочке, IDE выводит сообщение: «Для улучшения производительности используйте layout_height со значением 0dp вместо wrap_content». Щелкните на сообщении, чтобы применить рекомендацию. С этим изменением `LinearLayout` вычисляет размеры своих дочерних компонентов более эффективно. Окно макетного редактора должно выглядеть так, как показано на рис. 2.19.

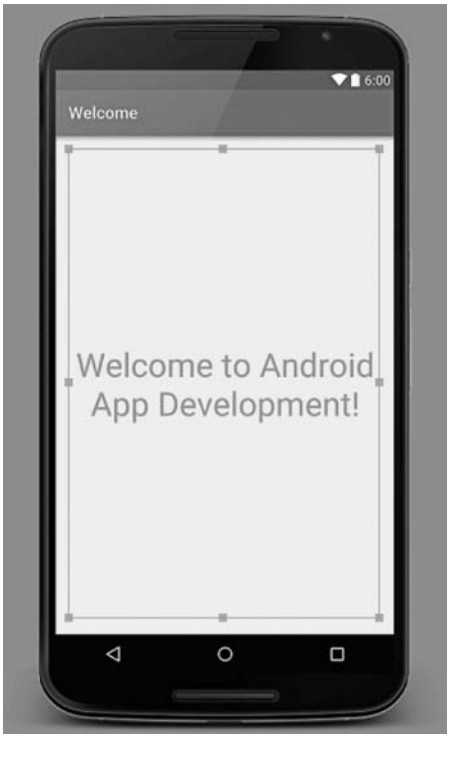


Рис. 2.19. Окно макетного редактора после настройки свойств TextView

ЗАЩИТА ОТ ОШИБОК

Android Lint проверяет проект на наличие типичных ошибок и советует, как улучшить его безопасность, быстродействие, доступность, интернационализацию и т. д. Некоторые проверки выполняются во время построения приложений и написания кода. Вы также можете выполнить команду Analyze ▸ Inspect Code... для дополнительной проверки конкретных файлов всего проекта. За дополнительной информацией обращайтесь по адресу *http://developer.android.com/tools/help/lint.html*. Информация о параметрах конфигурации и выходных данных Android Lint доступна по адресу *http://developer.android.com/tools/debugging/improving-w-lint.html*.

2.5.11. Добавление компонента ImageView для вывода изображения

А теперь добавим в графический интерфейс компонент `ImageView` для вывода изображения, добавленного в проект в разделе 2.5.1. Для этого следует перетащить компонент `ImageView` из раздела `Widgets` палитры в область холста (canvas). При перетаскивании представления на холст макетный редактор отображает оранжевые и зеленые линейки, а также экранную подсказку.

- Оранжевые линейки представляют границы каждого существующего представления в макете.
- Зеленые линейки указывают позицию нового представления относительно существующих представлений. По умолчанию новые представления добавляются к нижнему краю вертикального компонента `LinearLayout`, если только вы не наведете указатель мыши на оранжевый прямоугольник, ограничивающий верхнее представление макета.
- *Подсказка* (tooltip) сообщает, как будет настроен компонент, если отпустить его в текущей позиции мыши.

Чтобы добавить компонент `ImageView` и настроить его:

1. Откройте категорию `Widgets` на палитре и перетащите компонент `ImageView` на холст (рис. 2.20). *Прежде чем отпускать кнопку мыши*, убедитесь в том, что в подсказке выводится значение `center` — это означает, что макетный редактор настроит свойство `layout:gravity` компонента `ImageView` для горизонтального выравнивания `ImageView` по центру контейнера `LinearLayout`. Когда вы отпустите кнопку мыши, макетный редактор считает, что вес `layout:weight` компонента `ImageView` совпадает с весом `TextView`, и задает `layout:weight` значение 1. Также он задает `layout_height` значение 0dp, как было сделано для `TextView`. Новый компонент `ImageView` размещается под `TextView` в макете и под `welcomeTextView` в окне `Component Tree`. Свойства `ImageView` отображаются в окне свойств.

Подсказка сообщает, что макетный редактор задаст свойству layout:gravity нового компонента ImageView значение, обеспечивающее горизонтальное выравнивание ImageView по центру LinearLayout

Оранжевые линейки обозначают границы TextView

Во время перетаскивания ImageView в интерфейс приложения макетный редактор выводит зеленую линейку, обозначающую положение ImageView (или любого другого перетаскиваемого представления)

Рис. 2.20. Перетаскивание компонента ImageView в интерфейс приложения

2. В окне свойств найдите свойство `src` компонента `ImageView` (оно определяет выводимое изображение) и щелкните на кнопке с многоточием в поле значения. Открывается диалоговое окно `Resources` (рис. 2.21). Когда оно появится на экране, введите строку `bug`, чтобы найти в списке ресурсов изображение, добавленное в разделе 2.5.1, затем щелкните на кнопке `OK`. Для каждого изображения, помещаемого в папку `drawable`, IDE генерирует уникальный идентификатор (то есть имя) ресурса, по которому можно ссылаться на это изображение. Идентификатор ресурса представляет собой имя графического файла без расширения — `bug` для файла `bug.png`.
3. Сделайте двойной щелчок на изображении `ImageView` в макетном редакторе и задайте его свойству `id:` значение `bugImageView`.

Графический интерфейс должен выглядеть так, как показано на рис. 2.22. Если выделить компонент `ImageView` в макетном редакторе, Android Lint отображает рядом с ним изображение лампочки (). Если щелкнуть на лампочке, IDE выводит сообщение об отсутствии свойства для пользователей с ослабленным зрением. Недостаток будет исправлен в разделе 2.7.

Рис. 2.21. Выбор ресурса изображения в диалоговом окне Resources

Рис. 2.22. Готовый графический интерфейс

2.5.12. Предварительный просмотр

Android Studio также дает возможность увидеть, как ваше приложение будет выглядеть в альбомной ориентации и как оно смотрится на разных устройствах. Чтобы переключиться между портретной и альбомной ориентацией, щелкните на кнопке перехода к следующему состоянию `Go to next state` () на панели инструментов в верхней части макетного редактора. Так вы сможете определить, правильно ли адаптируется ваше приложение к разным вариантам ориентации. Чтобы оценить внешний вид приложения на разных устройствах, откройте раскрывающийся список виртуальных устройств (рис. 2.8) и выберите пункт `Preview All Screen Sizes`. На экране появляются миниатюрные изображения (рис. 2.23)

Рис. 2.23. Предварительный просмотр приложения Welcome для разных устройств

для многих пунктов списка виртуальных устройств — одни в портретной, другие в альбомной ориентации. По ним вы сможете быстро определить, правильно ли интерфейс приложения работает на разных устройствах. Чтобы вернуться к режиму вывода одного устройства, щелкните на раскрывающемся списке виртуальных устройств и выберите пункт `Remove Previews`. Также можно переключиться на режим просмотра для конкретного устройства, выбрав это устройство в раскрывающемся списке виртуальных устройств.

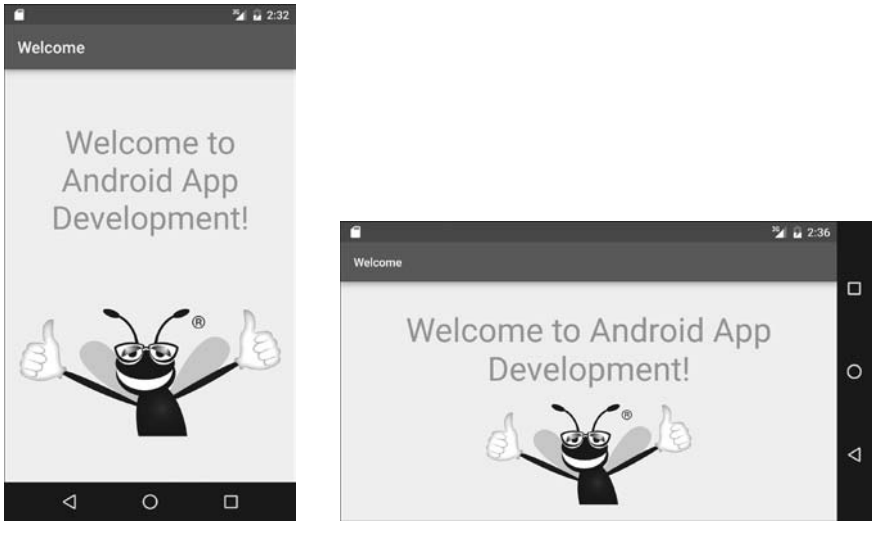


Рис. 2.24. Приложение Welcome в AVD для Nexus 6

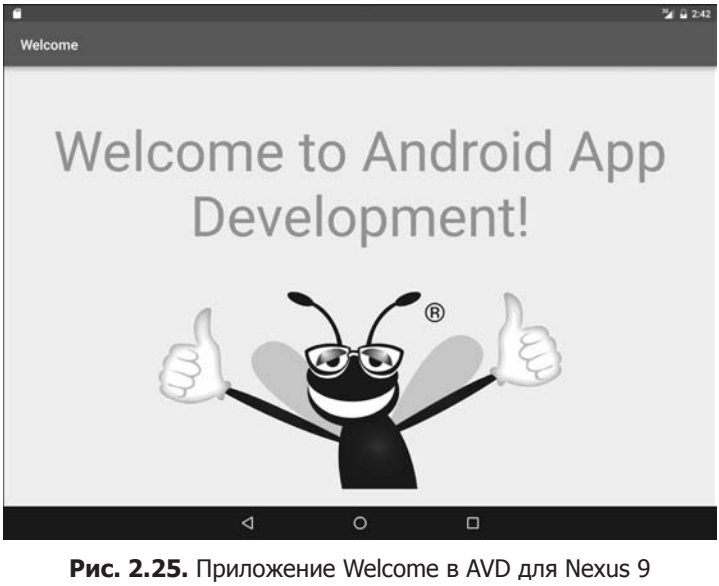


Рис. 2.25. Приложение Welcome в AVD для Nexus 9

2.6. Выполнение приложения Welcome

Теперь все готово к выполнению приложения `Welcome`. Выполните действия из раздела 1.9.3, чтобы запустить приложение на созданных ранее AVD для телефона Nexus 6 и планшета Nexus 9. На рис. 2.24–2.25 показано приложение в AVD для Nexus 6 (в портретной и альбомной ориентации) и Nexus 9 (в альбомной ориентации) соответственно. Чтобы переключить AVD между портретной и альбомной ориентацией, нажмите Ctrl+F11. Как правило, для приложений, работающих как на телефонах, так и на планшетах, также предоставляется планшетный макет, более эффективно использующий свободное пространство экрана (эта возможность будет продемонстрирована в следующих главах). Если у вас имеется устройство Android, выполните действия из раздела 1.9.4, чтобы выполнить приложение на устройстве.

2.7. Обеспечение доступности приложения

В Android реализованы средства *доступности*, упрощающие работу с устройствами для пользователей с ограниченными возможностями. Для слабовидящих пользователей функция Android TalkBack проговаривает экранный текст (или текст, предоставленный разработчиком при визуальном или программном построении графического интерфейса), чтобы помочь пользователю понять назначение компонента. Также существует функция *Explore by Touch*, с помощью которой пользователь может услышать, что находится на экране в точке касания.

Если пользователь касается компонента в режиме TalkBack, воспроизводится вспомогательный текст, а устройство вибрирует, чтобы предоставить обратную связь пользователям с ослабленным слухом. Все стандартные компоненты графического интерфейса Android поддерживают средства доступности. Если компонент выводит текст, то TalkBack по умолчанию читает этот текст, например, когда пользователь касается компонента `TextView`, TalkBack проговаривает текущее содержимое поля. Функция TalkBack включается в приложении `Settings` (раздел `Accessibility`). Здесь же можно включить и другие средства доступности Android — например, средства доступности по умолчанию (такие как увеличенный размер текста по умолчанию и применение жестов для увеличения области экрана). К сожалению, функция TalkBack в настоящее время *не поддерживается* в AVD, и чтобы услышать, как TalkBack читает текст, приложение необходимо запустить на физическом устройстве. При включении TalkBack Android предоставляет возможность просмотреть руководство по использованию функций TalkBack и Explore by Touch.

Включение функции TalkBack для компонентов ImageView

В приложении `Welcome` нам не нужен более содержательный текст для `TextView`, потому что TalkBack читает содержимое `TextView`. Однако у компонента `ImageView` не существует текста, который бы читался экранным диктором, если только вы не предоставите его сами. В Android рекомендуется обеспечить возможность использования *каждого* экранного компонента с TalkBack — для этого следует задать значение свойства `contentDescription` каждого компонента, не отображающего текст. По этой причине IDE предупреждает нас о проблеме, отображая лампочку (— рис. 2.22) в макетном редакторе рядом с каждым компонентом `ImageView`. Эти предупреждения (они генерируются служебной программой Android Lint) сообщают, что мы не задали свойство `Content Description` каждого изображения. Текст этого свойства должен помочь пользователю понять назначение компонента. Для компонентов `ImageView` текст должен описывать изображение.

Чтобы добавить описание `contentDescription` для компонента `ImageView` (и избавиться от предупреждений Android Lint), выполните следующие действия.

1. Выделите компонент `bugImageView` в макетном редакторе.
2. В окне свойств щелкните на кнопке с многоточием справа от свойства `contentDescription`, чтобы открыть диалоговое окно `Resources`.
3. Щелкните на кнопке `New Resource` и выберите `New String Value`, чтобы отобразить диалоговое окно `New String Value Resource`.
4. В поле `Resource name` введите текст `deitel_logo`, а в поле `Resource value` — текст «Deitel double-thumbs-up bug logo». Нажмите кнопку `OK`. Новый строковый ресурс автоматически выбирается как значение `contentDescription`.

С заполнением свойства `contentDescription` в макетном редакторе исчезает предупреждающий значок лампочки.

Тестирование приложения с включенной функцией TalkBack

Запустите это приложение на устройстве с включенной функцией TalkBack. Коснитесь компонентов `TextView` и `ImageView` и прослушайте соответствующий текст.

Динамическое создание представлений

Некоторые приложения динамически генерируют компоненты графического интерфейса в ответ на действия пользователя. Для таких компонентов текст доступности может задаваться на программном уровне. Следующие страницы документации разработчика Android содержат дополнительную информацию

о средствах доступности Android и контрольный список, по которому следует действовать при разработке доступных приложений:

http://developer.android.com/design/patterns/accessibility.html
http://developer.android.com/guide/topics/ui/accessibility/index.html

2.8. Интернационализация приложения

Как известно, устройства Android используются во всем мире. Чтобы ваши приложения охватывали как можно большую аудиторию, подумайте о том, чтобы адаптировать их для разных культур и языков. Например, если вы собираетесь распространять свое приложение во Франции, переведите его ресурсы (текст, аудиофайлы и т. д.) на французский язык. Также возможно использовать разные цвета, графику и звуки в зависимости от *локального контекста* (locale). Для каждого локального контекста создается новый специализированный набор ресурсов. Когда пользователь запускает приложение, Android автоматически находит и загружает ресурсы, соответствующие настройкам локального контекста устройства. Проектирование приложений с возможностью такой настройки называется *интернационализацией*. Адаптация ресурсов приложения для каждого локального контекста называется *локализацией*.

2.8.1. Локализация

Важнейшее преимущество определения строковых значений в виде строковых ресурсов (как в нашем приложении) заключается в том, что вы можете легко *локализовать* свое приложение, создавая дополнительные файлы ресурсов в формате XML для других языков. Во всех файлах используются одни и те же имена ресурсов строк, но с разными переводами. Android выбирает ресурсный файл в зависимости от основного языка, выбранного на устройстве пользователя.

2.8.2. Имена папок с локализованными ресурсами

XML-файлы ресурсов, содержащие локализованные строки, размещаются во вложенных папках папки `res` проекта. Android использует специальные правила назначения имен папок для автоматического выбора правильных локализованных ресурсов — например, папка `values-fr` содержит файл `strings.xml` для французского языка, а папка `values-es` содержит файл `strings.xml` для испанского языка. В именах папок также может присутствовать региональная информация — обе папки, `values-en-rUS` и `values-en-rGB`, содержат файл `strings.xml` для английского языка, но первая предназначена для США, а вторая — для Великобритании.

Если локализованные ресурсы для нужного локального контекста отсутствуют, Android использует ресурсы приложения по умолчанию (то есть ресурсы из папки `values` в папке `res`). Правила назначения имен папок с альтернативными ресурсами будут более подробно рассмотрены далее.

2.8.3. Добавление папки локализации в проект приложения

В Android Studio существует специальный редактор Translations Editor для быстрого и удобного добавления переводов существующих строк в приложения. Чтобы добавить в проект переведенные строки, выполните следующие действия.

1. В окне `Project` раскройте узел `values` и откройте файл `strings.xml`.
2. Щелкните на ссылке `Open editor` в правом верхнем углу, чтобы открыть редактор Translations Editor.
3. В левом верхнем углу Translations Editor щелкните на кнопке `Add Locale` () и выберите вариант `Spanish (es)` — для ускорения поиска введите часть названия языка или его сокращение (`es`). После выбора локального контекста из списка создается новый файл `strings.xml (es)`, который добавляется в узел `strings.xml` в окне `Project` (файл сохраняется в папке `values-es` на диске). В редакторе также появляется новый столбец для переводов на испанский язык.
4. Чтобы добавить перевод на испанский для заданного строкового ресурса, щелкните на ячейке перевода, а затем введите испанский текст в поле `Translation:` в нижней части окна. Если строка не должна переводиться (например, если она никогда не выводится для пользователя), установите флажок `Untranslatable` для данного ресурса. Для приложения `Welcome` используйте переводы из раздела 2.8.4.

Повторите эти действия для каждого языка, который вы намерены поддерживать.

2.8.4. Локализация строк

Графический интерфейс нашего приложения содержит компонент `TextView`, в котором выводится текстовое сообщение, и компонент `ImageView` со строкой, описывающей его содержимое. Все эти строки определяются как ресурсы в файле `strings.xml`. Теперь мы можем предоставить переведенные строки, которые будут сохранены в новой версии файла `strings.xml`. Для данного приложения строки

```
"Welcome to Android App Development!"
"Deitel double-thumbs-up bug logo"
```

будут заменены испанскими строками

```
"¡Bienvenido al Desarrollo de App Android!"
"El logo de Deitel que tiene el insecto con dedos pulgares hacia arriba"
```

В окне `Translations Editor`:

1. Щелкните на ячейке перевода `Spanish (es)` первого ресурса и введите в поле `Translation:` в нижней части окна испанскую строку `"¡Bienvenido al Desarrollo de App Android!"`. Если вы не можете ввести специальные испанские символы и знаки на своей клавиатуре, скопируйте испанские строки из файла `res/values-es/strings.xml` в окончательной версии приложения Welcome (из папки `WelcomeInternationalized` в примерах этой главы), после чего вставьте испанскую строку в поле `Translation:`.
2. Щелкните на ячейке со значением ресурса `deitel_logo` и введите в поле `Translation:` строку `"El logo de Deitel que tiene el insecto con dedos pulgares hacia arriba"`.
3. Ресурс `app_name` остается без перевода, хотя в принципе его можно перевести. Окно должно выглядеть так, как показано на рис. 2.26.
4. Сохраните файл `strings.xml` для испанского языка командой `File` ▸ `Save All` или кнопкой `Save All` (💾) на панели инструментов.

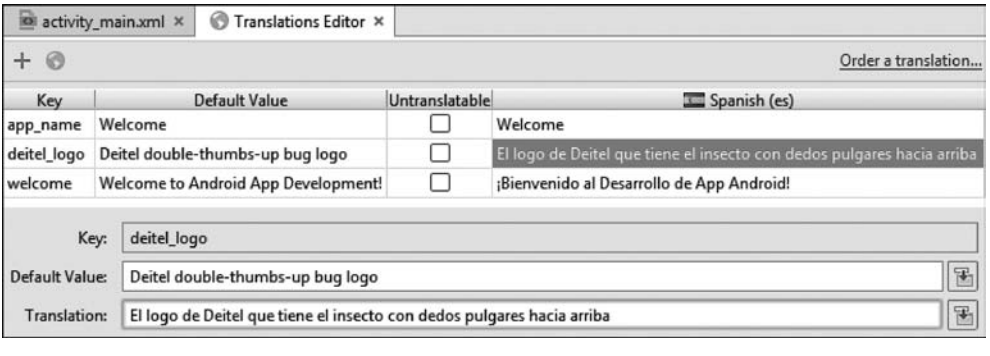


Рис. 2.26. Окно Translations Editor с испанскими строками

2.8.5. Тестирование приложения с испанской локализацией в AVD

Чтобы протестировать приложение с испанской локализацией в AVD, воспользуйтесь приложением `Custom Locale`, установленным в AVD.

1. Нажмите кнопку Home (◉) в AVD.
2. Нажмите кнопку лаунчера (⋮⋮⋮), найдите значок приложения `Custom Locale` и запустите приложение.
3. Прокрутите список до варианта `es - español`, щелкните на нем и нажмите кнопку `SELECT 'ES'`, чтобы изменить локальный контекст AVD.

Эмулятор или устройство переключится на испанский язык.

Теперь запустите приложение `Welcome`; при этом устанавливается и запускается интернационализированная версия. На рис. 2.27 показано приложение, локализованное для испанского языка. В самом начале работы приложения Android проверяет языковые настройки AVD (или устройства), определяет, что в AVD (или на устройстве) включен испанский язык, и использует строковые ресурсы `welcome` и `deitel_logo`, определенные в файле `res/values-es/strings.xml` работающего приложения. Однако следует обратить внимание на то, что имя приложения на панели действий в верхней части приложения все равно выводится на *английском* языке. Дело в том, что мы *не* предоставили локализованную версию строкового ресурса `app_name` в файле `res/values-es/strings.xml`. Вспомните, о чем говорилось ранее: если Android не может найти локализованную версию строкового ресурса, используется версия по умолчанию из файла `res/values/strings.xml`.

Рис. 2.27. Приложение Welcome на испанском языке в Nexus 6 AVD

Возвращение AVD к английскому языку

Чтобы снова переключить AVD на английский язык:

1. Нажмите кнопку Home () в AVD.
2. Нажмите кнопку лаунчера (), найдите значок приложения Custom Locale и запустите приложение.
3. Прокрутите список до варианта en-US - en-us, щелкните на нем и нажмите кнопку SELECT 'EN-US', чтобы изменить локальный контекст AVD.

2.8.6. Тестирование приложения с испанской локализацией на устройстве

Чтобы протестировать приложение с испанской локализацией на устройстве, необходимо изменить языковые настройки вашего устройства.

1. Нажмите кнопку Home () на устройстве.
2. Нажмите кнопку лаунчера (), найдите значок приложения Settings () и запустите приложение.
3. В приложении Settings перейдите к разделу Personal, затем откройте категорию Language & input.
4. Откройте пункт Language (первый элемент списка) и выберите в списке строку Español (España).

Эмулятор или устройство переключается на испанский язык и возвращается к настройкам Language & input, которые теперь отображаются на испанском. Запустите свое приложение из IDE; при этом устанавливается и запускается интернационализированная версия.

Возвращение устройства к английскому языку

Чтобы снова переключить устройство на английский язык:

1. Нажмите кнопку Home () на устройстве.
2. Нажмите кнопку лаунчера (), найдите значок приложения Settings () и запустите приложение.
3. Перейдите к разделу Idioma e introduccion de texto, чтобы вызвать языковые настройки.
4. Откройте пункт Idioma и выберите в списке строку English (United States).

2.8.7. TalkBack и локализация

Функция TalkBack в настоящее время поддерживает английский, испанский, итальянский, французский и немецкий языки. Если запустить приложение `Welcome` на устройстве с испанским языком и включенной функцией TalkBack, то TalkBack будет зачитывать испанские строки при касании компонентов графического интерфейса.

Когда вы впервые переключите устройство на испанский язык и включите TalkBack, Android автоматически загрузит ядро синтеза речи. Если TalkBack *не* произносит испанские строки, значит, загрузка ядра еще не завершилась и установка продолжается. В таком случае попробуйте запустить приложение позднее.

2.8.8. Контрольный список локализации

За дополнительной информацией о локализации ресурсов приложения обращайтесь к документации Android Localization Checklist по адресу

http://developer.android.com/distribute/tools/localization-checklist.html

2.8.9. Профессиональный перевод

Компании, занимающиеся разработкой приложений, часто имеют в штате переводчиков или пользуются услугами других компаний для выполнения перевода. Более того, на сайте Google Play Developer Console (используемом для публикации приложений в магазине Google Play) приведен список компаний, предоставляющих услуги перевода, а в окне `Translations Editor` присутствует ссылка `Order translations...`. За дополнительной информацией о Google Play Developer Console обращайтесь к главе 10 и по адресу

http://developer.android.com/distribute/googleplay/developer-console.html

Также информацию о переводе можно найти по адресу

https://support.google.com/l10n/answer/6227218

2.9. Резюме

В этой главе мы воспользовались средой разработки Android Studio для построения приложения `Welcome`, которое отображает текстовое сообщение и изображение без написания какого-либо кода. Мы создали простой графический

интерфейс в макетном редакторе и настроили свойства компонентов в окне свойств.

В XML-файле разметки используемый по умолчанию компонент `RelativeLayout` был заменен компонентом `LinearLayout`, который затем был настроен для вертикального расположения представлений. Приложение выводит текст в компоненте `TextView`, а изображения — в компоненте `ImageView`. Мы изменили компонент `TextView` графического интерфейса по умолчанию, чтобы текст выравнивался по центру, увеличенным шрифтом и в одном из цветов стандартной темы. Компоненты `ImageView` перетаскивались мышью из палитры компонентов макетного редактора. Как и положено, все строки и числовые значения были определены в файлах ресурсов в папке `res` проекта.

Также были представлены средства обеспечения доступности, упрощающие использование устройств людьми с ограниченными возможностями. Вы узнали, как использовать функцию TalkBack — экранный диктор, который помогает пользователю лучше понять назначение и содержимое компонента. Функция Android Explore by Touch позволяет прикоснуться к экрану, чтобы диктор TalkBack зачитал информацию о содержимом экрана рядом с точкой касания. Для компонента `ImageView` приложения было предоставлено описание контента, который может использоваться функциями TalkBack и Explore by Touch.

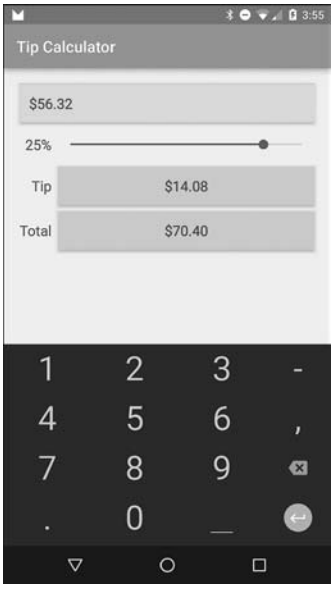
Наконец, вы научились пользоваться средствами интернационализации Android, для того чтобы ваши приложения охватывали как можно большую аудиторию. Приложение `Welcome` было локализовано испанскими строками для текста `TextView` и описаниями для компонентов `ImageView`, после чего оно было протестировано на виртуальном устройстве AVD, настроенном для испанского языка.

Разработка Android сочетает в себе визуальное построение графического интерфейса с программированием на языке Java. В следующей главе будет разработано простое приложение `Tip Calculator` для подсчета размера чаевых в ресторанах; при этом макетный редактор будет использоваться для построения графического интерфейса, а программирование на Java — для определения поведения.

3 Приложение Tip Calculator

Знакомство с GridLayout, EditText, SeekBar, обработкой событий, NumberFormat, настройкой темы приложения и определением функциональности приложения на языке Java

В этой главе…



- Изменение темы графического интерфейса по умолчанию
- Настройка цветов темы
- Построение графического интерфейса с использованием компонента GridLayout
- Использование окна Component Tree для добавления представлений в GridLayout.
- Использование компонентов TextView, EditText и SeekBar
- Применение объектно-ориентированных возможностей Java, включая классы, объекты, интерфейсы, анонимные внутренние классы и наследование, для расширения функциональности приложений Android
- Программное изменение текста в TextView
- Использование обработки событий при взаимодействии с пользователем с помощью компонентов EditText и SeekBar
- Отображение виртуальной клавиатуры по умолчанию при выполнении приложения
- Ограничение приложения портретной ориентацией

3.1. Введение

Приложение `Tip Calculator` (рис. 3.1, *а*) вычисляет и отображает чаевые на основании счета в ресторане. После ввода пользователем выставленного счета с цифровой клавиатуры приложение вычисляет и отображает размер чаевых и величину итогового счета (с учетом 15% чаевых по умолчанию). Пользователь может указать собственную ставку в диапазоне от 0% до 30%. Для этого нужно переместить *ползунок* компонента `Seekbar`, после чего обновляется величина процентной ставки, пересчитывается сумма чаевых и общая сумма. Все числовые значения отображаются с использованием форматирования *с учетом контекста*. На рис. 3.1, *б* показано окно приложения после того, как пользователь ввел счет 56,32 и выбрал процент чаевых 25%.

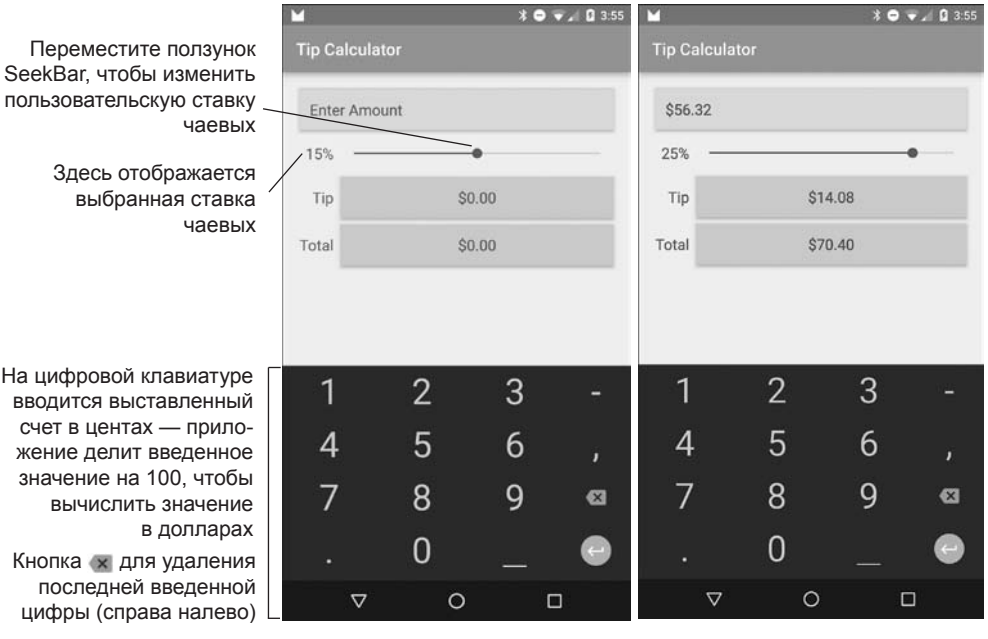


Рис. 3.1. Ввод выставленного счета и вычисление чаевых: *а* — исходный вид графического интерфейса; *б* — графический интерфейс после ввода суммы 56,32 и выбора пользовательской ставки чаевых 25%

Начнем с тестирования приложения. Затем будет приведен краткий обзор технологий, применяемых для создания приложения. Для построения графического интерфейса будет использоваться макетный редактор Android Studio и окно `Component Tree`. В завершение главы будет приведен полный код Java и подробное описание кода.

О клавиатуре на снимках экрана

Клавиатура на рис. 3.1 может выглядеть иначе в зависимости от версии AVD или устройства, а также от того, была ли выбрана или установлена специальная клавиатура на вашем устройстве. Мы настроили AVD для отображения темной клавиатуры на снимках экрана для улучшения контраста.

1. Нажмите кнопку `Home` () на AVD или физическом устройстве.
2. Нажмите кнопку лаунчера (), зайдите в `Settings` и запустите приложение.
3. В AVD выберите строку `Android Keyboard (AOSP)`.
4. На устройстве выберите `Google Keyboard` — предполагается, что вы используете стандартную клавиатуру Android.
5. Перейдите к разделу `Appearance & layouts` и выберите раздел `Theme`.
6. Выберите вариант `Material Dark`, чтобы переключиться на клавиатуру с темным фоном.

3.2. Тестирование приложения Tip Calculator

Запуск и выполнение приложения

Выполните действия, описанные в разделах 1.9.1 и 1.9.3, чтобы открыть проект `Tip Calculator` в Android Studio и запустить приложение в Nexus 6 AVD. При желании выполните шаги из раздела 1.9.4, чтобы запустить приложение на телефоне Android.

Ввод суммы

Введите на цифровой клавиатуре сумму 56,32. Если при вводе будет допущена ошибка, нажмите кнопку удаления (), чтобы стереть последнюю введенную цифру. И хотя на клавиатуре имеется кнопка с точкой, приложение настроено так, что оно принимает только цифры от 0 до 9 — остальные кнопки на клавиатуре игнорируются, а устройство Android начинает вибрировать, сообщая о нажатии недопустимой кнопки. Каждый раз, когда вы касаетесь цифры или удаляете цифру, приложение:

- преобразует его в число;
- делит число на 100 и выводит новую величину счета;
- пересчитывает чаевые и сумму счета на основании текущего процента чаевых (15% по умолчанию) и выводит содержимое компонентов `TextView` с новой суммой чаевых и общей суммой.

Если удалить все цифры, приложение снова выводит строку `Enter Amount` в синем поле `TextView`, и строку `0.00` в оранжевом поле `TextView`. Приложение делит значение на 100,0 и выводит результат в синем поле `TextView`. Затем приложение вычисляет и обновляет чаевые и сумму счета в оранжевых полях `TextView`.

Все денежные суммы выводятся в финансовом формате текущего локального контекста, а процент чаевых — в формате процентов текущего локального контекста. Для локального контекста США пользователь вводит четыре цифры —5, 6, 3 и 2, а величина счета последовательно отображается в виде `$0.05`, `$0.56`, `$5.63` и `$56.32` соответственно.

Выбор пользовательского процента чаевых

Компонент `Seekbar` задает пользовательский процент чаевых. Перетащите ползунок `Seekbar` до отметки 25% (рис. 3.1, *б*). В процессе перетаскивания происходит непрерывное обновление величины чаевых и общего счета для текущего процента. По умолчанию компонент `Seekbar` позволяет выбирать значения в диапазоне от 0 до 100, но в нашем приложении шкала ограничивается максимальным значением 30.

3.3. Обзор применяемых технологий

В этом разделе представлены возможности Java и Android, используемые для построения приложения `Tip Calculator`. Предполагается, что вы *уже* знакомы с объектно-ориентированным программированием на языке Java. В частности, мы будем:

- использовать различные классы Android для создания объектов;
- вызывать методы объектов и классов Android;
- определять и вызывать пользовательские методы;
- использовать наследование для создания класса, определяющего функциональность приложения `Tip Calculator`;
- использовать обработку событий, анонимные внутренние классы и интерфейсы для обработки взаимодействий пользователя с интерфейсом.

3.3.1. Класс Activity

В отличие от многих приложений Java, приложения Android не содержат метода main. Вместо этого в них используются четыре типа исполняемых компонентов — *активности* (activities), *службы* (services), *провайдеры контента*

и *широковещательные приемники* (broadcast receivers). В этой главе рассматриваются активности, определяемые как субклассы `Activity` (пакет `android.app`). Приложение может иметь много активностей, одна из которых — первое, что вы видите при запуске приложения. Пользователи взаимодействуют с активностями через *представления* (views) — компоненты GUI, наследующие от класса `View` (пакет `android.view`).

До выхода Android 3.0 с каждым экраном приложения обычно связывалась отдельная активность. Как будет показано в главе 4, активность может управлять несколькими *фрагментами* (fragments). На телефоне каждый фрагмент обычно занимает целый экран, а активность переключается между фрагментами на основании взаимодействий пользователя. На планшетах активности часто отображают несколько фрагментов на экран, чтобы более эффективно использовать доступное пространство.

3.3.2. Методы жизненного цикла активности

На протяжении своего существования активность может находиться в одном из нескольких состояний — *активном* (то есть выполняемом), *приостановленном* или *остановленном*. Переходы активностей между этими состояниями происходят в ответ на различные *события*.

- «Активная активность» отображается на экране и «обладает фокусом» — то есть взаимодействует с пользователем.
- Приостановленная активность видна на экране, но *не обладает* фокусом (например, на время отображения диалогового окна с сообщением). Пользователь не может взаимодействовать с приостановленной активностью, пока она снова не станет активной — например после того, как пользователь закроет диалоговое окно.
- Остановленная активность *не отображается* на экране и, вероятно, будет уничтожена системой, когда потребуется освободить занимаемую ею память. Активность *останавливается*, когда другая активность переходит в *активное* состояние. Например, когда вы отвечаете на телефонный звонок, приложение, управляющее звонками, становится активным, а предыдущее приложение останавливается.

При переходах активности между этими состояниями исполнительная среда Android вызывает различные *методы жизненного цикла* (все эти методы определяются в классе Activity из пакета `android.app`). В ваших приложениях для *каждой* активности будет переопределяться метод `onCreate`. Этот метод вызывается исполнительной системой Android при запуске активности — то есть когда ее графический интерфейс готов к отображению, чтобы пользователь мог взаимодействовать с активностью. Также у активностей существуют другие

методы жизненного цикла: `onStart`, `onPause`, `onRestart`, `onResume`, `onStop` и `onDestroy`. Большинство этих методов будет рассмотрено в дальнейших главах. Каждый переопределяемый вами метод жизненного цикла активности *должен* вызывать версию метода из суперкласса; в противном случае происходит исключение. Вызов версии суперкласса необходим, потому что каждый метод жизненного цикла в суперклассе `Activity` содержит код, который должен выполняться помимо кода, определяемого вами в переопределенных методах жизненного цикла. За дополнительной информацией о жизненном цикле `Activity` обращайтесь по адресу

http://developer.android.com/reference/android/app/Activity.html

3.3.3. Библиотека AppCompat и класс AppCompatActivity

При использовании новых возможностей на более ранних платформах Android разработчик сталкивается с проблемой обеспечения совместимости. Теперь Google открывает доступ ко многим новым возможностям Android через *Android Support Library* — набор библиотек, благодаря которым разработчик может использовать эти возможности на современных и старых платформах Android.

Одна из таких библиотек — *AppCompat* — обеспечивает поддержку *панели приложения* (которая прежде называлась панелью действий) и других возможностей на устройствах Android 2.1 (API 7) и выше; напомним, что панель приложения впервые появилась в Android 3.0 (API 11). Обновленные шаблоны приложений Android Studio тоже используют библиотеку AppCompat, поэтому новые приложения, которые вы создаете, будут работать почти на всех устройствах Android.

Шаблон Android Studio `Empty Activity` определяет класс `MainActivity` приложения как субкласс `AppCompatActivity` (пакет `package android.support.v7.app`) — непрямого субкласса `Activity`, обеспечивающего использование новых средств Android на современных и старых платформах Android.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 3.1

Если в ваших приложениях с самого начала будет использоваться библиотека AppCompat, то вам не придется переписывать свой код, если вы решите поддержать старые версии Android для расширения потенциальной аудитории своего приложения.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 3.2

Некоторые возможности Android недоступны в ранних версиях даже при использовании библиотеки AppCompat. Например, поддержка печати доступна только в Android 4.4 и выше. Если такие возможности используются в ваших приложениях, вам придется либо ограничить приложение поддерживаемыми платформами, либо заблокировать соответствующую функциональность в неподдерживаемых версиях Android.

За дополнительной информацией о библиотеках поддержки Android, о том, когда их следует использовать и как настроить, обращайтесь по адресу

http://developer.android.com/tools/support-library

3.3.4. Построение представления с использованием компонента GridLayout

Напомним, что *макеты* (layouts) предназначены для размещения представлений в графическом интерфейсе. Компонент `GridLayout` (пакет `android.widget`) используется для размещения компонентов в ячейках прямоугольной таблицы. Ячейки могут занимать сразу несколько строк и столбцов, что позволяет строить достаточно сложные макеты. Обычно для использования `GridLayout` требуется API уровня 14 и выше. Однако библиотека поддержки Android Support Library предоставляет альтернативные версии `GridLayout` и многих других компонентов графического интерфейса, что позволяет использовать их в старых версиях Android. За дополнительной информацией об этой библиотеке и о том, как использовать ее в приложениях, обращайтесь по адресу

http://developer.android.com/tools/support-library/index.html

Макеты и представления более подробно рассматриваются в дальнейших главах — за полным списком обращайтесь по адресу

http://developer.android.com/reference/android/widget/package-summary.html

3.3.5. Создание и настройка графического интерфейса

Мы создадим интерфейс с компонентами `TextView`, `EditText` и `SeekBar` в макетном редакторе IDE (см. главу 2) и окне `Component Tree`, а затем настроим их в окне свойств IDE.

Компонент `EditText` (часто называемый *текстовым полем* в других технологиях) представляет собой субкласс `TextView` (см. главу 2); он предназначен для вывода текста *и* получения текста, вводимого пользователем. Можно создать специализированную версию EditText для цифрового ввода, разрешить пользователям вводить только цифры и ограничить максимальное количество вводимых цифр.

Компонент `SeekBar` представляет целое число (по умолчанию в диапазоне 0–100). Перемещая ползунок, пользователь может выбрать число в диапазоне допустимых значений. Мы настроим `SeekBar` так, чтобы пользователь мог выбрать процент чаевых только из ограниченного диапазона от 0 до 30.

3.3.6. Форматирование чисел в соответствии с локальным контекстом

Класс `NumberFormat` (пакет `java.text`) используется для создания строк денежных величин и процентов в формате локального контекста — это важная составляющая интернационализации. Вы также можете добавить строки доступности и интернационализировать приложение средствами, описанными в разделах 2.7–2.8.

3.3.7. Реализация интерфейса TextWatcher для обработки изменений в компоненте EditText

Мы воспользуемся анонимным внутренним классом для реализации *интерфейса* `TextWatcher` (из пакета `android.text`), чтобы обеспечить реакцию на события при изменении текста в поле `EditText` данного приложения. В частности, метод `onTextChanged` будет использоваться для отображения величины счета, отформатированной по правилам денежных сумм, для вычисления чаевых и общей суммы при вводе пользователем каждой цифры. Читатели, не знакомые с понятием анонимного внутреннего класса, найдут информацию по адресу

http://bit.ly/AnonymousInnerClasses

3.3.8. Реализация интерфейса OnSeekBarChangeListener для обработки изменения позиции ползунка SeekBar

Мы реализуем интерфейс `SeekBar.OnSeekBarChangeListener` (из пакета `android.widget`), чтобы реагировать на перемещения ползунка `SeekBar`. В частности, метод `onProgressChanged` будет использоваться для отображения пользовательской процентной ставки, вычисления чаевых и общей суммы при перемещении ползунка `SeekBar`.

3.3.9. Материальные темы

Тема приводит внешний вид приложения в соответствие с визуальным стилем Android. В проектах для Android 5 и выше используются темы, отвечающие правилам материального дизайна Google.

Существуют несколько предопределенных тем материального дизайна.

- Тема «light» с белой панелью приложения, белым фоном и текстом черного или темно-серого цвета.

- Тема «light» с темной панелью приложения отличается от предыдущей тем, что панель приложения окрашена в черный цвет, а цвет по умолчанию выводится белым шрифтом.
- Тема «dark» с черной панелью приложения, темно-серым фоном и текстом белого или светло-серого цвета.

Каждая из этих тем существует в двух версиях:

- `Theme.Material` (например, `Theme.Material.Light`) для приложений, которые не используют библиотеки `AppCompat` и работают в Android 5 и выше, и
- `Theme.AppCompat` (например, `Theme.AppCompat.Light`) для приложений, которые используют библиотеки `AppCompat` и работают в Android 2.1 и выше.

При построении графического интерфейса можно выбрать нужную тему из готового набора или даже создать собственную тему. В этой главе будет использоваться `Theme.AppCompat.Light.DarkActionBar` — тема по умолчанию для шаблонов приложений Android Studio. Приложения, использующие библиотеки `AppCompat`, должны использовать одну из тем `AppCompat`; в противном случае некоторые представления могут отображаться некорректно. Дополнительную информацию по каждой теме и снимки экранов можно найти по адресам

http://www.google.com/design/spec/style/color.html#color-themes

http://developer.android.com/training/material/theme.html

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ 3.1

На многих современных телефонах на базе Android используются AMOLED-экраны. На таких экранах черные пикселы отключаются и не потребляют энергию. Приложения, использующие преимущественно черные темы, могут сократить энергозатраты приблизительно на 40% (http://bit.ly/AndroidAMOLEDDisplay).

3.3.10. Материальный дизайн: рельеф и тени

В рекомендациях материального дизайна Google говорится, что объекты интерфейса пользователя должны отбрасывать тени, как и объекты реального мира. Когда для представления задается свойство `elevation`, Android автоматически генерирует тень от этого представления. Чем больше значение `elevation`, тем четче выражена тень. В приложении `Tip Calculator` мы зададим свойство `elevation` для синего и оранжевого компонентов `TextView`, отображающих денежные суммы.

В рекомендациях материального дизайна приведены желательные значения `elevation` для разных экранных компонентов — например, для диалоговых окон

рекомендуемое значение `elevation` равно `24dp`, а для меню — `8dp`. За информацией о других рекомендуемых значениях обращайтесь по адресу

http://www.google.com/design/spec/what-is-material/elevation-shadows.html

3.3.11. Материальный дизайн: цвета

Разработчики приложений часто приводят цвета темы в соответствие с фирменным стилем компании. Если вам потребуется изменить цвета темы, рекомендации материального дизайна Google по использованию цвета[1] советуют выбрать цветовую палитру, состоящую из основного цвета (не более чем с тремя оттенками) и акцентного цвета. Основные цвета обычно используются для окрашивания строки состояния и панели приложения в верхней части экрана; кроме того, они могут использоваться в графическом интерфейсе. Акцентный цвет предназначен для выделения оттенков в различных представлениях — таких, как `SeekBar`, `CheckBox` и `RadioButton`. После выбора палитры вы сможете воспользоваться редактором Android Studio Theme Editor (раздел 3.5.2) для изменения цветов темы.

Образцы рекомендуемых цветов из палитры материального дизайна можно найти по адресу

http://www.google.com/design/spec/style/color.html#color-color-palette

Рекомендации по выбору цветов палитры доступны по адресу

http://www.materialpalette.com/

Сайт позволяет выбрать два цвета из палитры материального дизайна Google, после чего рекомендует три оттенка основного цвета, один дополнительный цвет и цвета текста и значков приложения.

В нашем приложении образцы цвета, отображаемые в Android Studio Theme Editor, будут использоваться для выбора:

- синего основного цвета фона для панели приложения;
- темно-синего основного цвета для строки состояния, отображаемой над панелью приложения;
- оранжевого акцентного цвета для придания оттенка `SeekBar`.

Для светло-синего компонента `TextView` со счетом, а также светло-оранжевых компонентов `TextView` с чаевыми и общей суммой мы воспользуемся палитрой Google для выбора более светлых оттенков основного и акцентного цветов.

[1] *http://www.google.com/design/spec/style/color.html.*

3.3.12. AndroidManifest.xml

Файл `AndroidManifest.xml` генерируется средой разработки при создании нового проекта приложения. В файле хранятся многие настройки, вводимые в диалоговом окне `Create New Project`: имя приложения, имя пакета, имя класса активности(-ей) и т. д. Мы отредактируем разметку XML этого файла и добавим в нее новый параметр, включающий постоянное отображение виртуальной клавиатуры на экране. Также будет указано, что приложение поддерживает только *портретную ориентацию* (то есть вертикальной является более длинная сторона).

3.3.13. Поиск в окне свойств

Окно свойств позволяет проводить поиск свойств по именам (полным или частичным), чтобы вам было проще находить и задавать значения свойств. Щелкните на заголовке окна свойств и начинайте вводить символы. В начале списка свойств появляется подсказка с введенным текстом, а Android Studio выделяет в списке части имен, полностью или частично совпадающие с введенным текстом. Вы можете прокрутить список и найти в нем имена свойств, содержащие выделенные фрагменты.

Окно также прокручивается до свойства, которое лучше всего соответствует введенному тексту. Например, если при поиске свойств `TextView` будет введен текст «text co» или «textco», в окне свойств также будут подсвечены части многих свойств, но окно прокрутится и выделит свойство `textColor`.

3.4. Построение графического интерфейса приложения

В этом разделе будет описан процесс построения графического интерфейса `Tip Calculator`, включая настройку основного и акцентного цветов.

3.4.1. Основы GridLayout

В приложении используется объект `GridLayout` (пакет `android.widget`) для расположения компонентов GUI в четырех *строках* и двух *столбцах*. Нумерация строк и столбцов начинается с нуля, как и нумерация элементов массива. Количество строк и столбцов в `GridLayout` задается в окне свойств. Каждая ячейка `GridLayout` либо остается пустой, либо содержит одно или несколько

представлений, которыми могут быть макеты, *содержащие* другие компоненты. *Высота* каждой строки определяется максимальной высотой представлений в этой строке. Аналогичным образом ширина каждого *столбца* определяется максимальной шириной представлений в этом столбце. На рис. 3.2 изображен компонент `GridLayout` в приложении `Tip Calculator` с пометкой строк и столбцов: столбцы разделяются вертикальными линиями, а строки — горизонтальными. Представления могут занимать *несколько* строк и/или столбцов — например, поле `Enter Amount` на рис. 3.2 охватывает два столбца в строке 0.

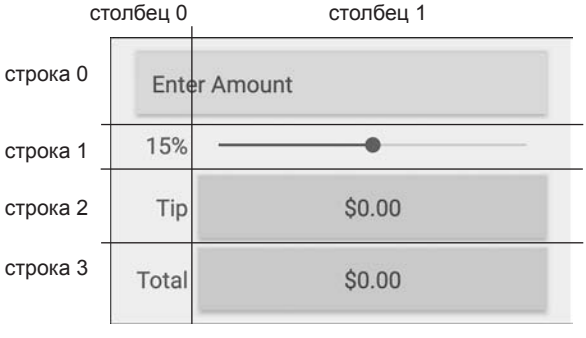


Рис. 3.2. Компонент GridLayout в приложении Tip Calculator

При перетаскивании представления на `GridLayout` в окне `Component Tree` представление занимает следующую доступную ячейку таблицы — компонент `GridLayout` заполняется слева направо, пока не закончится вся строка, после чего следующее представление добавляется в первый столбец следующей строки. Как вы увидите, существует возможность задать строку и столбец для размещения представления. Другие возможности `GridLayout` будут продемонстрированы при описании действий по построению графического интерфейса.

Значения свойства Id компонентов данного приложения

На рис. 3.3 приведены значения свойств `Id` компонентов данного приложения. Для наглядности в нашей схеме имя класса компонента используется в свойстве `Id` компонента и имени переменной Java. В первой строке в действительности находятся *два* компонента — `amountTextView` (изначально содержит текст «Enter Amount») закрывает компонент `amountEditText`, который получает ввод пользователя. Как будет вскоре показано, пользовательский ввод ограничивается цифрами, поэтому значение $34,56 должно вводиться в виде 3456. Это гарантирует, что пользователь не сможет ввести некорректное значение. Однако при этом пользователю должна быть показана сумма счета в виде денежной величины. При вводе каждой цифры значение делится на 100, а результат в формате денежной суммы отображается в `amountTextView`.

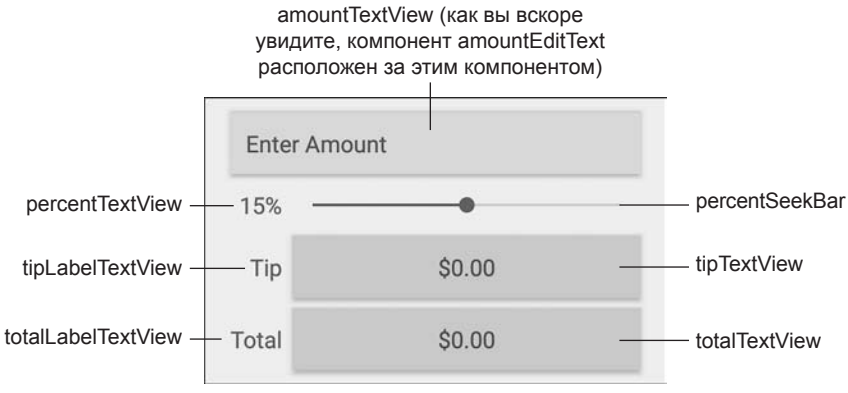


Рис. 3.3. Компоненты графического интерфейса Tip Calculator со значениями их свойств Id

3.4.2. Создание проекта TipCalculator

Выполните действия, описанные в разделе 2.3, чтобы создать новый проект на базе шаблона `Empty Activity`. Введите следующие значения на шаге `New Project` диалогового окна `Create New Project`:

- `Application name`: `Tip Calculator`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

Для остальных шагов диалогового окна `Create New Project` используйте те же значения, что и в разделе 2.3. Также выполните действия из раздела 2.5.2, чтобы добавить значок приложения в проект.

Когда проект откроется в Android Studio, в макетном редакторе выберите в списке виртуальных устройств Nexus 6 (см. рис. 2.8) — мы снова возьмем это устройство за основу для построения приложения. Также удалите компонент `TextView` с текстом «Hello World!».

3.4.3. Переключение на GridLayout

Напомним, что по умолчанию в шаблоне `Empty Activity` используется компонент `RelativeLayout`. В нашем примере он будет заменен компонентом `GridLayout`:

1. Щелкните на вкладке `Text` в нижней части макетного редактора, чтобы перейти из режима `Design` в режим просмотра разметки.
2. В верхней части разметки XML замените `RelativeLayout` на `GridLayout`.
3. Вернитесь в режим `Design` макетного редактора.

Настройка компонента GridLayout с двумя столбцами и отступами по умолчанию

Итак, графический интерфейс на рис. 3.2 состоит из двух столбцов. Чтобы назначить этот параметр, выделите компонент `GridLayout` в окне `Component Tree`, затем задайте его свойству `columnCount` значение 2 (это свойство находится в начале списка вместе с другими макетными свойствами). Задавать свойство `rowCount` не нужно — оно будет изменяться в процессе построения графического интерфейса.

По умолчанию ячейки `GridLayout` не имеют отступов (пространство, разделяющее компоненты). В рекомендациях материального дизайна предлагается разделять представления минимальным промежутком `8dp`:

http://developer.android.com/design/style/metrics-grids.html

Компонент `GridLayout` может автоматически включить рекомендуемые отступы. Выделите `GridLayout` в окне `Component Tree`, а затем в окне свойств установите флажок свойства `useDefaultMargins` компонента, чтобы ячейки макета окружались стандартными отступами.

3.4.4. Добавление компонентов TextView, EditText, SeekBar и LinearLayout

Перейдем к построению графического интерфейса на рис. 3.2. Мы начнем с создания базового макета и компонентов. В разделе 3.4.5 интерфейс будет завершен настройкой свойств компонентов. Затем в разделе 3.5 мы сменим тему по умолчанию и настроим два ее цвета. При добавлении каждого компонента в интерфейс немедленно задайте его свойство `Id` в соответствии с рис. 3.3. Для добавления новых компонентов в `GridLayout` используется окно `Component Tree`. Если представление будет размещено неточно, перетащите его в нужное место.

ЗАЩИТА ОТ ОШИБОК 3.1

Ячейки в зеленой сетке макетного редактора малы. Если вы случайно допустите ошибку при размещении представления, макетный редактор может изменить свойства rowCount и columnCount компонента GridLayout и неправильно задать свойства layout:row и layout:column представления, что приведет к нарушению структуры графического интерфейса. В таком случае измените свойства rowCount и columnCount компонента GridLayout, а также свойства layout:row и layout:column представления в соответствии со своим макетом.

Также можно перетащить представления прямо в макетный редактор. Для компонента `GridLayout` в макетном редакторе отображаются зеленые линейки,

которые помогают представить позицию представления. Во время перетаскивания макетный редактор отображает подсказку со строкой и столбцом, в которых будет размещено это представление, если отпустить его в текущей позиции мыши.

Шаг 1: Добавление компонентов в первую строку

Первая строка состоит из компонентов `amountTextView` и `amountEditText` — оба компонента занимают одну ячейку и распространяются на два столбца. Каждый раз, когда вы перетаскиваете компонент в `GridLayout` из окна `Component Tree`, он размещается в *следующей открытой* ячейке макета, если только его позиция не задается явно при помощи свойств `layput:row` и `layout:column` представления. Мы воспользуемся этой возможностью, чтобы компоненты `amountEditText` и `amountTextView` размещались в одной ячейке (`amountTextView` на переднем плане).

Все компоненты `TextView` в этом приложении используют *средний* шрифт из темы приложения. Палитра макетного редактора предоставляет готовые варианты `TextView` с именами `Plain Text`, `Large Text`, `Medium Text` и `Small Text` (в разделе `Widgets`) для соответствующих размеров текста. Вариант `Plain Text` использует шрифт по умолчанию для текущей темы. В остальных случаях среда разработки настраивает свойство `textAppearance` компонента `TextView`, используя стили материальной темы для соответствующего размера шрифта.

Выполните следующие действия, чтобы добавить в `GridLayout` два компонента, `TextView` и `EditText` для отображения и ввода денежной суммы.

1. Приложение позволяет вводить только неотрицательные целые числа, которые делятся на 100, для вывода суммы счета. В разделе `Text Fields` палитры находится много готовых разновидностей `EditText` для ввода разных данных (имена, пароли, адреса электронной почты, телефонные номера, время, даты, числа и т. д.). Когда пользователь взаимодействует с `EditText`, на экране отображается клавиатура, соответствующая типу данных `EditText`. Найдите в палитре раздел `Text Fields` и перетащите компонент `Number EditText` на узел `GridLayout` в окне `Component Tree` — в `GridLayout` размещается компонент `EditText` со свойством `id editText`. Этот компонент размещается в первом столбце первой строки таблицы `GridLayout`. Задайте свойству `layout:column` компонента `EditText` значение 0, а свойству `layout:columnSpan` значение 2 — с этими значениями поле занимает оба столбца строки 0.
2. Перетащите другой компонент `Medium Text` из раздела `Widgets` палитры на узел `amountEditText` в окне `Component Tree`. Под `amountEditText` появляется черная горизонтальная линия, которая показывает, что компонент `TextView` будет размещен после `amountEditText`. Среда разработки создает новый компонент `TextView` с именем `textView` и помещает его в узел `GridLayout`.

В макетном редакторе выводится стандартный текст "Medium Text", он будет изменен на шаге 5 (раздел 3.4.5). Задайте свойству id компонента TextView значение amountTextView, затем задайте свойствам layout:row и layout:column значение 0, а свойству layout:columnSpan значение 2 — с этими значениями поле занимает оба столбца строки 0. Вы убедитесь в этом, когда мы изменим цвет фона TextView.

Шаг 2: Добавление компонентов во вторую строку

Затем в GridLayout добавляются компоненты percentTextView и percentSeekBar для вывода и выбора процента чаевых (проследите за тем, чтобы значения свойства id каждого представления были заданы именно так, как указано в тексте).

1. Перетащите компонент Medium Text (percentTextView) из раздела Widgets палитры на компонент amountTextView узла GridLayout в окне Component Tree. Новое представление становится первым в строке 1 (вторая строка).
2. Перетащите компонент SeekBar (percentSeekBar) из раздела Widgets палитры на компонент percentTextView узла GridLayout в окне Component Tree. Новое представление становится вторым в строке 1.

Шаг 3: Добавление компонентов в третью строку

На следующем шаге в GridLayout добавляются компоненты tipLabelTextView и tipTextView для вывода суммы чаевых.

1. Перетащите компонент Medium Text (tipLabelTextView) на percentSeekBar в узле GridLayout. В результате новый компонент становится первым в строке 2 (третья строка).
2. Перетащите компонент Medium Text (tipTextView) на tipLabelTextView в узле GridLayout. В результате новый компонент становится вторым в строке 2.

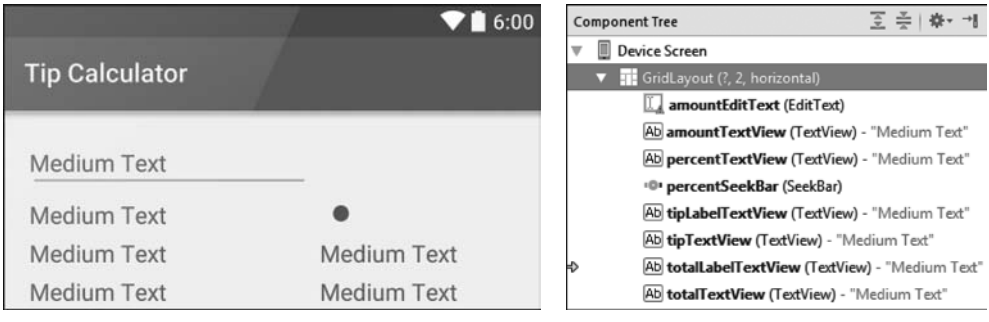
Шаг 4: Добавление компонентов в четвертую строку

Теперь в GridLayout следует добавить компоненты totalLabelTextView и totalTextView для вывода общей суммы.

1. Перетащите компонент Medium Text (totalLabelTextView) на tipTextView в узле GridLayout. В результате новый компонент становится первым в строке 3 (четвертая строка).
2. Перетащите компонент Medium Text (totalTextView) на totalLabelTextView в узле GridLayout. В результате новый компонент становится вторым в строке 3.

Текущее состояние приложения

Примерный вид графического интерфейса и окна `Component Tree` показан на рис. 3.4. Предупреждающие значки в макетном редакторе и окне `Component Tree` исчезнут после завершения макета в разделе 3.4.5.



а *б*

Рис. 3.4. Графический интерфейс и окно Component Tree после добавления компонентов в GridLayout: *а* — текущее состояние графического интерфейса; *б* — окно Component Tree с компонентами Tip Calculator

О виртуальной клавиатуре EditText

Когда виртуальная клавиатура находится на экране, кнопка `Back` (◁) заменяется направленной вниз стрелкой (▽), нажатие которой убирает клавиатуру с экрана. В этом случае стрелка снова заменяется кнопкой `Back` (◁), которая возвращается к предыдущей активности — обычно к предыдущему приложению или главному экрану устройства.

Обычно виртуальная клавиатура возвращается на экран прикосновением к `EditText`. В этом приложении `EditText` скрывается за `TextView`. Если закрыть клавиатуру, то для ее повторного отображения придется выйти из приложения и вернуться к нему. Можно включить режим принудительного отображения клавиатуры на экране, но тогда кнопка `Back` вообще не будет отображаться в приложении. А это сделает невозможным возврат к предыдущей активности — пользователь обычно ожидает, что эта возможность существует в любом приложении.

Мы использовали виртуальную клавиатуру Android, чтобы показать, как выбрать клавиатуру, отображаемую для конкретного компонента `EditText`. Другое возможное решение — вывести кнопки с цифрами 0–9, которые всегда остаются на экране. Приложение обрабатывает события щелчков и операции со строками, вместо того чтобы отслеживать пользовательский ввод в `EditText`.

3.4.5. Настройка компонентов

Чтобы завершить построение приложения, мы настроим свойства компонентов и создадим несколько ресурсов для строк и метрик. Также мы создадим несколько ресурсов со строками, размерами и цветами (по аналогии с тем, как было сделано в разделе 2.5).

Шаг 5: Ввод текстовых литералов

На следующем шаге будут заданы текстовые литералы для компонентов `amountTextView`, `percentTextView`, `tipLabelTextView` и `totalLabelTextView`. Если свойство `text` компонента `TextView` пусто, в поле выводится значение свойства `hint` (если оно задано) — это свойство часто используется для компонентов `EditText` (субкласс `TextView`), чтобы помочь пользователю понять назначение компонента. Мы используем это свойство с `amountTextView` и подсказываем пользователю, что в поле должна вводиться величина счета.

1. Выделите компонент `amountTextView` в окне `Component Tree` и найдите его свойство `hint` в окне свойств.
2. В окне свойств щелкните на кнопке с многоточием рядом со значением свойства, чтобы вызывать диалоговое окно `Resources`.
3. В диалоговом окне `New Resource` нажмите кнопку `New String Value...`, чтобы вызвать диалоговое окно `New String Value Resource`. Введите в поле `Resource name` значение `enter_amount`, а в поле `Resource value` — значение `"Enter Amount"`. Оставьте остальные значения без изменений и щелкните на кнопке `OK`, чтобы создать новый строковый ресурс и связать его со свойством `hint` компонента `amountTextView`.

Повторите описанные операции, чтобы задать свойство `Text` компонентов `percentTextView`, `tipLabelTextView` и `totalLabelTextView`, используя значения из табл. 3.1.

Таблица 3.1. Значения и идентификаторы строковых ресурсов

Компонент	Имя ресурса	Значение ресурса
percentTextView	tip_percentage	15%
tipLabelTextView	tip	Tip
totalLabelTextView	total	Total

Шаг 6: Выравнивание компонентов TextView по правому краю левого столбца

На рис. 3.2 компоненты `percentTextView`, `tipLabelTextView` и `totalLabelTextView` выровнены по правому краю. Их можно задать сразу для всех компонентов `TextView`.

1. Выделите компонент `percentTextView`.
2. Удерживая нажатой клавишу Ctrl (Windows/Linux) или Command (Mac), щелкните на компонентах `tipLabelTextView` и `totalLabelTextView`. В результате выделяются все три компонента `TextView`.
3. Раскройте узел свойства `layout:gravity` и установите флажок `right`.

Шаг 7: Настройка компонента amountEditText

В приложении компонент `amountEditText` скрывается за `amountTextView` и настраивается так, чтобы пользователь мог вводить только цифры. Выделите `amountEditText` и задайте следующие свойства.

1. Задайте свойству `digits` значение `0123456789` — оно разрешает вводить *только* цифры, хотя цифровая клавиатура также содержит кнопки для ввода минуса, запятой и точки.
2. Задайте свойству `maxLength` значение 6. Максимальная сумма счета ограничивается шестью цифрами, так что наибольшая сумма, поддерживаемая в приложении, равна 9999,99.

Шаг 8: Настройка компонента amountTextView

1. Удалите значение по умолчанию свойства `text` (`"Medium Text"`) — выводимый текст будет задаваться на программном уровне в зависимости от данных, вводимых пользователем.
2. Раскройте узел `layout:gravity` и задайте свойству `fill` значение `horizontal`. Это означает, что свойство `TextView` займет все остающееся горизонтальное пространство в строке `GridLayout`.
3. Задайте свойству `background` (которое задает цвет фона представления) новый цветовой ресурс с именем `amount_background` и значением `#BBDEFB` — светло-синим цветом, выбранным из палитры материального дизайна Google.
4. Добавьте отступы вокруг `TextView`. Свойство `Padding` компонента задает величину отступов вокруг содержимого компонента. Значение `all` указывает, что отступы должны быть применены к верхней, правой, нижней и левой стороне содержимого. Отступы также можно задать для каждой из сторон по отдельности. Раскройте узел свойства `padding`, щелкните на свойстве `all`,

а затем на кнопке с многоточием. Создайте новый ресурс метрики с именем `textview_padding` и значением `12dp`. Вскоре мы снова используем этот ресурс.

5. Остается добавить эффект тени. Задайте свойству `elevation` новый ресурс метрики с именем `elevation` и значением `4dp`. Мы выбрали это значение для демонстрационных целей, чтобы эффект тени был более выразительным.

Шаг 9: Настройка компонента percentTextView

Обратите внимание: компонент `percentTextView` расположен выше ползунка `percentSeekBar`. Макет будет смотреться лучше, если выровнять компонент вертикально по центру. Для этого раскройте узел `layout:gravity` и выберите в разделе `center` значение `vertical`. Вспомните, что ранее `layout:gravity` было присвоено значение `right`. Комбинация этих значений в разметке XML имеет вид

```
android:layout_gravity="center_vertical|right"
```

Вертикальная черта (`|`) используется для разделения нескольких значений `layout:gravity` — в данном случае этот символ означает, что компонент `TextView` должен выравниваться в ячейке по правому краю, а по вертикали — по центру.

Шаг 10: Настройка компонента percentSeekBar

Выделите компонент `percentSeekBar` и задайте следующие свойства.

1. По умолчанию `SeekBar` использует диапазон допустимых значений от 0 до 100, а его текущее значение обозначается свойством `progress`. Приложение поддерживает проценты по шкале от 0 до 30, а значение по умолчанию равно 15. Задайте свойству `max` компонента `SeekBar` значение 30, а свойству `progress` — значение 15.
2. Раскройте узел `layout:gravity` и выберите в разделе `fill` значение `horizontal`, чтобы компонент `SeekBar` занимал все горизонтальное пространство в столбце компонента `SeekBar`.
3. Задайте свойству `layout:height` новый ресурс метрики (`seekbar_height`) со значением `40dp`, чтобы увеличить вертикальное пространство для отображения `SeekBar`.

Шаг 11: Настройка компонентов tipTextView и totalTextView

Чтобы завершить форматирование `tipTextView` и `totalTextView`, выделите оба компонента и задайте следующие свойства.

1. Удалите значение по умолчанию свойства `text` (`"Medium Text"`) — выводимый текст будет задаваться на программном уровне в зависимости от данных, вводимых пользователем.

2. Раскройте узел `layout:gravity` и выберите в разделе `fill` значение `horizontal`, чтобы каждый компонент `TextView` занимал все горизонтальное пространство в столбце компонентов `TextView`.
3. Задайте свойству `background` новый цветовой ресурс с именем `result_background` и значением `#FFE0B2` — светло-оранжевым цветом, выбранным из палитры материального дизайна Google.
4. Задайте свойству `gravity` значение `center`, чтобы вычисленные значения чаевых и общей суммы выравнивались по центру в компонентах `TextView`.
5. Раскройте узел свойства `padding`, щелкните на свойстве `all`, а затем на кнопке с многоточием. Выберите ресурс метрики с именем `textview_padding`, который был создан ранее для `amountTextView`.
6. Примените эффект тени к каждому компоненту. Для этого задайте свойству `elevation` ресурс метрики с именем `elevation`, созданный ранее.

3.5. Тема по умолчанию и настройка цветов темы

У каждого приложения существует тема, определяющая оформление стандартных компонентов, которые вы используете. Тема приложения указывается в файле `AndroidManifest.xml` приложения (раздел 3.7). Вы можете настроить различные аспекты темы (например, составляющие цветовой схемы), определяя ресурсы в файле `styles.xml`, находящемся в папке `res/values` приложения.

3.5.1. Родительские темы

Ресурсный файл `style.xml` содержит стиль с именем `"AppTheme"`, ссылка на который включается в файл `AndroidManifest.xml` приложения для назначения темы. Этот стиль также определяет *родительскую тему*, которая может рассматриваться как аналог суперкласса в Java — новый стиль наследует атрибуты родительской темы и их значения по умолчанию. Как и субкласс Java, стиль может переопределить атрибуты родительской темы значениями, адаптированными для конкретных приложений (например, для использования в приложении фирменной цветовой гаммы компании). Мы используем эту концепцию в разделе 3.5.2 для настройки трех цветов, используемых в теме приложения.

Как упоминалось ранее, шаблоны приложений Android Studio теперь включают поддержку библиотек `AppCompat`, позволяющих использовать новые возможности Android в старых версиях платформы. По умолчанию Android Studio выбирает родительскую тему `Theme.AppCompat.Light.DarkActionBar`, одну из нескольких

стандартных тем в библиотеке `AppCompat` — приложения, использующие эту тему, отображаются на светлом фоне, а в верхней части приложения располагается темная панель приложения. Все темы `AppCompat` используют рекомендации материального дизайна Google для оформления графических интерфейсов.

3.5.2. Настройка цветов темы

В разделе 3.3.11 мы упоминали о том, где в экранных элементах используются цвета темы: основной, темный основной и акцентный. В этом разделе вы научитесь использовать новый редактор темы Android Studio для изменения этих цветов приложения; при этом переопределяются значения атрибутов темы `android:colorPrimary`, `android:colorPrimaryDark` и `android:colorAccent` (рис. 3.5). Это всего лишь три из многочисленных атрибутов темы, которые вы можете переопределять. Полный список доступен по адресу

http://developer.android.com/reference/android/R.attr.html

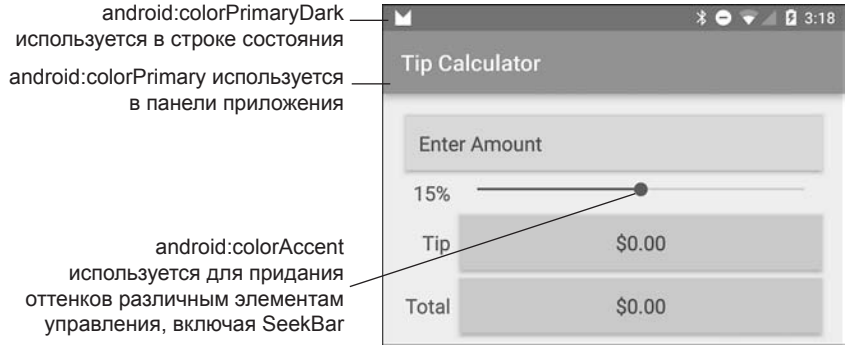


Рис. 3.5. Атрибуты темы для основного, основного черного и акцентного цвета

Изменение основного, темного основного и акцентного цвета

Настройка цветов осуществляется следующим образом.

1. Откройте файл `styles.xml`. В правом верхнем углу редактора щелкните на ссылке `Open editor`, чтобы открыть редактор Theme Editor (рис. 3.6) с текущими цветами `colorPrimary` (темно-синий), `colorPrimaryDark` (более темный оттенок `colorPrimary`) и `colorAccent` (ярко-розовый) — эти цвета выбраны по умолчанию в шаблоне Android Studio `Empty Activity`. В этом приложении мы заменим цвета `colorPrimary` и `colorPrimaryDark` более светлыми оттенками синего, а цвет `colorAccent` будет заменен оранжевым.
2. Чтобы задать значение `colorPrimary`, щелкните на образце цвета (рис. 3.6). На экране появляется диалоговое окно `Resources` (рис. 3.7). В этом диалоговом

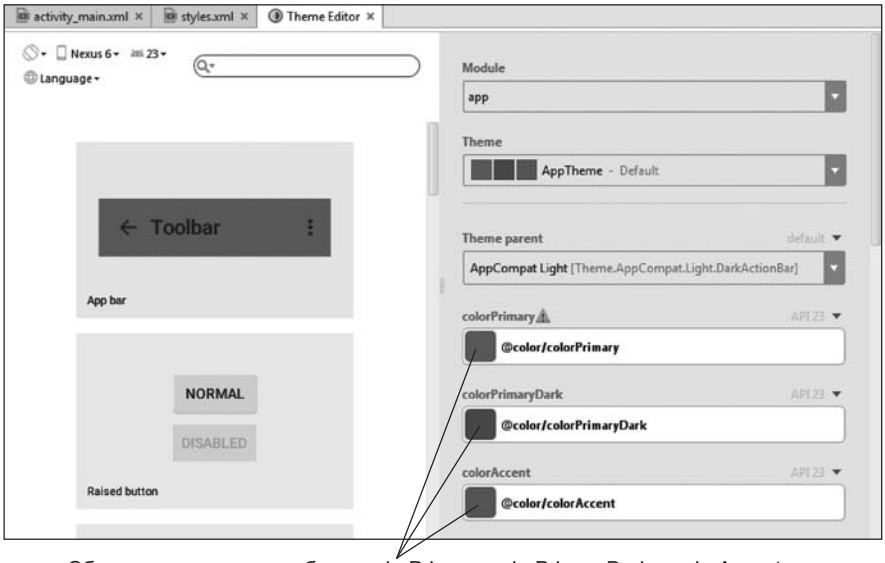


Рис. 3.6. В редакторе Theme Editor слева выводятся предварительные изображения компонентов, а справа — атрибуты темы

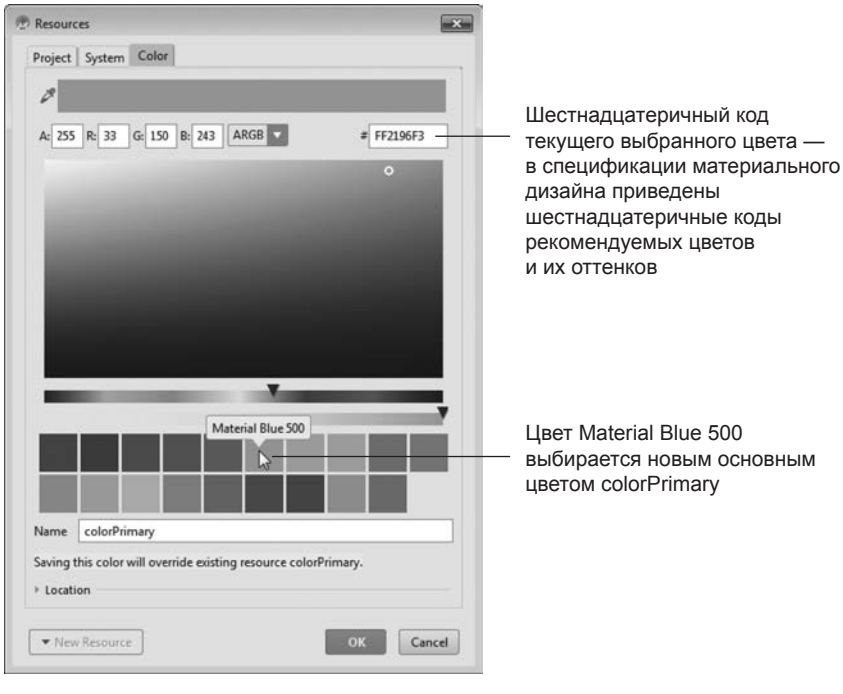


Рис. 3.7. Выбор образца Material Blue 500 для цвета colorPrimary

окне щелкните на образце `Material Blue 500`, а затем на кнопке `OK` для изменения значения `colorPrimary` — если навести указатель мыши на образец цвета, появляется подсказка с названием цвета. Число 500 обозначает конкретный оттенок цвета Material Blue. Оттенки цветов лежат в диапазоне от 50 (светлый) до 900 (темный) — с примерами можно ознакомиться по адресу

https://www.google.com/design/spec/style/color.html#color-color-palette

3. Щелкните на образце `colorPrimaryDark` в редакторе Theme Editor, чтобы открыть диалоговое окно `Resources`. Theme Editor распознает новое значение `colorPrimary` и автоматически выводит образец с рекомендуемым темным оттенком `colorPrimary`, который следует использовать для colorPrimaryDark — в данном случае `Material Blue 700`. Щелкните на образце (рис. 3.8), затем на кнопке `OK`.

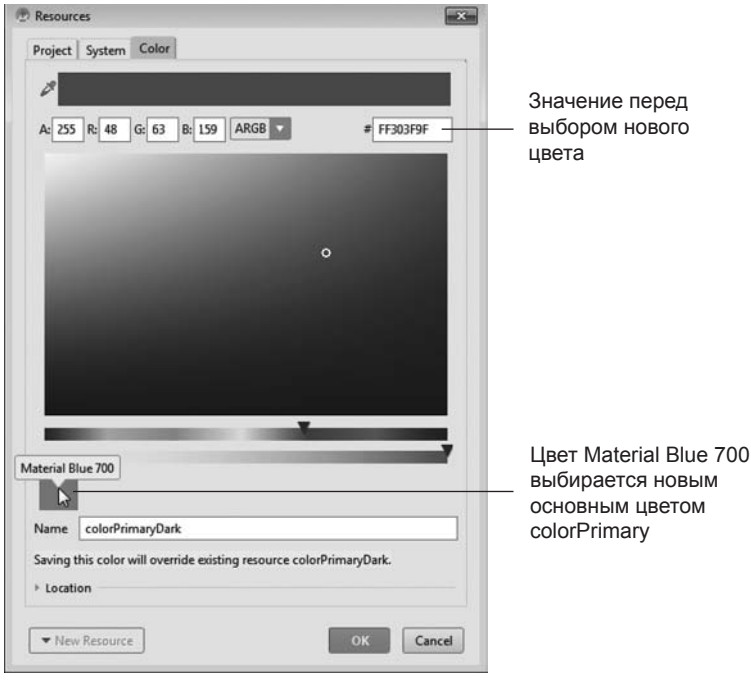


Рис. 3.8. Выбор образца Material Blue 700 для цвета colorPrimaryDark

4. Щелкните на образце цвета `colorAccent` в `Theme Editor`, чтобы открыть диалоговое окно `Resources`. Снова `Theme Editor` распознает изменение цвета `colorPrimary` и выводит образцы для разных дополнительных акцентных цветов. В диалоговом окне щелкните на образце `Orange accent 400`, а затем

на кнопке OK, чтобы изменить значение colorAccent (рис. 3.9). Щелкните на кнопке OK.

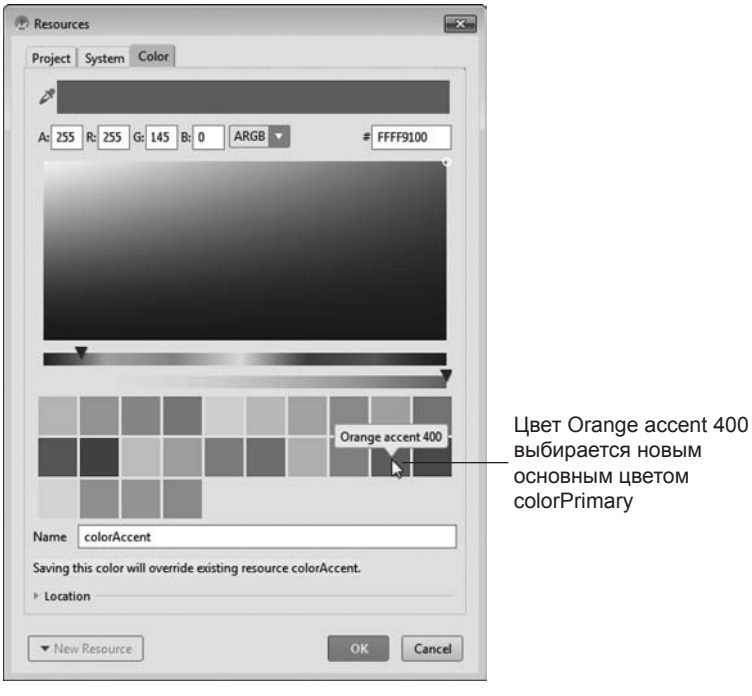


Рис. 3.9. Выбор образца Orange accent 400 для цвета colorAccent

Построенный графический интерфейс приложения должен выглядеть так, как показано на рис. 3.10.

3.5.3. Стили и часто используемые значения свойств

Как будет показано в дальнейших приложениях, стилевые ресурсы (style) определяют значения свойств, которые должны применяться к разным представлениям. Чтобы назначить стилевой ресурс некоторому представлению, вы задаете его свойство style. Все последующие изменения, вносимые в стиль, автоматически применяются ко всем представлениям, которые этот стиль используют. Например, возьмем компоненты tipTextView и totalTextView, которые одинаково настраивались на шаге 11 раздела 3.4.5. Вместо этого можно было определить ресурс style со значениями свойств layout:gravity, background, gravity, padding и elevation, а затем задать этот ресурс свойствам style обоих компонентов TextView.

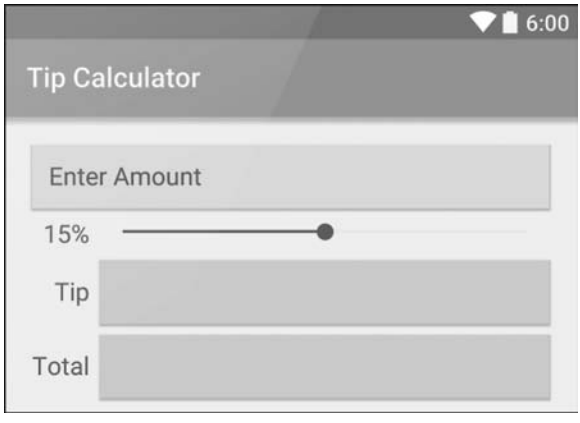


Рис. 3.10. Завершенный графический интерфейс приложения

3.6. Логика приложения

Класс `MainActivity` (листинги 3.1–3.7) реализует функциональность приложения `Tip Calculator`. Он вычисляет чаевые и общую сумму и выводит результаты в формате денежной суммы для действующего локального контекста. Чтобы просмотреть файл, в окне `Project` откройте папку `app/Java/com.deitel/tipcalculator` и сделайте двойной щелчок на файле `MainActivity.java`. Вам придется ввести большую часть кода в листингах 3.1–3.7.

3.6.1. Команды package и import

В листинге 3.1 приведены команды `package` и `import` из файла `MainActivity.java`. Команда `package` в строке 3 была вставлена при создании проекта. Когда файл Java открывается в среде разработки, команды `import` сворачиваются — отображается только одна команда со значком ⊕ слева. Щелкнув на ⊕, вы сможете просмотреть список команд `import`.

Листинг 3.1. Команды package и import из файла MainActivity

```
1  // MainActivity.java
2  // Вычисление общей суммы счета с заданным процентом чаевых
3  package com.deitel.tipcalculator;
4
5  import android.os.Bundle; // Для хранения информации состояния
6  import android.support.v7.app.AppCompatActivity; // Базовый класс
7  import android.text.Editable; // Для обработки событий EditText
8  import android.text.TextWatcher; // Слушатель EditText
9  import android.widget.EditText; // Для ввода счета
```

```
10 import android.widget.SeekBar; // Для изменения процента чаевых
11 import android.widget.SeekBar.OnSeekBarChangeListener; // Слушатель SeekBar
12 import android.widget.TextView; // Для вывода текста
13
14 import java.text.NumberFormat; // Для форматирования денежных сумм
15
```

В строках 5–14 импортируются классы и интерфейсы, используемые приложением.

- Класс `Bundle` из пакета `android.os` (строка 5) хранит набор пар «ключ—значение» — обычно эти пары представляют информацию состояния приложения или данные, передаваемые между активностями. Android дает приложению возможность сохранить свое состояние в `Bundle` перед тем, как другое приложение появится на экране (например, когда пользователь запустит другое приложение или примет телефонный звонок). После этого исполнительная среда Android может закрыть приложение, то есть освободить занимаемую им память. Когда приложение возвращается на экран, исполнительная среда Android передает объект `Bundle` с ранее сохраненным состоянием методу `onCreate` активности (раздел 3.6.4). После этого приложение может использовать сохраненные данные и вернуться к тому состоянию, в котором оно находилось перед активизацией другого приложения. Объекты `Bundle` будут использоваться в главе 8 для передачи данных между активностями.
- Класс `AppCompatActivity` из пакета `android.support.v7.app` (строка 6) предоставляет основные методы жизненного цикла приложения (вскоре они будут рассмотрены подробнее). Класс `AppCompatActivity` (непрямой субкласс класса `Activity` из пакета `android.app`) обеспечивает поддержку новой функциональности в приложениях, работающих на старых платформах Android.
- Интерфейс `Editable` из пакета `android.text` (строка 7) позволяет изменять содержимое и разметку текста в графическом интерфейсе.
- Интерфейс `TextWatcher` из пакета `android.text` (строка 8) реализуется для обработки событий при изменении пользователем текста в `EditText`.
- Пакет `android.widget` (строки 9, 10 и 12) содержит *виджеты* (то есть визуальные компоненты) и макеты, используемые в графическом интерфейсе Android. В приложении используются виджеты `EditText` (строка 9), `SeekBar` (строка 10) и `TextView` (строка 12).
- Интерфейс `SeekBar.OnSeekBarChangeListener` из пакета `android.widget` (строка 11) реализуется для обработки событий, возникающих при перемещении ползунка `SeekBar`.
- Класс `NumberFormat` из пакета `java.text` (строка 14) предоставляет средства числового форматирования (например, форматы денежных сумм и процентов по правилам локального контекста).

3.6.2. Класс MainActivity

Класс `MainActivity` (листинги 3.2–3.7) является основным классом активности приложения `Tip Calculator`. При создании проекта `Tip Calculator` среда разработки сгенерировала этот класс как субкласс `AppCompatActivity` (непрямого субкласса `Activity`) и предоставила переопределенную версию унаследованного от `Activity` метода `onCreate` (листинг 3.4). Каждый субкласс `Activity` *должен* переопределять этот метод. Метод `onCreate` будет рассмотрен далее.

Листинг 3.2. Класс MainActivity является субклассом Activity

```
16 // Класс MainActivity приложения Tip Calculator
17 public class MainActivity extends Activity {
18
```

3.6.3. Переменные класса

В листинге 3.3 объявляются переменные класса `MainActivity`. Объекты `NumberFormat` (строки 20–23) используются для форматирования денежных сумм и процентов соответственно. Статический метод `getCurrencyInstance` класса `NumberFormat` возвращает объект `NumberFormat`, который форматирует значения как денежные суммы с использованием локального контекста устройства. Аналогичный статический метод `getPercentInstance` форматирует значения как проценты с использованием локального контекста устройства.

Листинг 3.3. Переменные экземпляров класса MainActivity

```
19  // Форматировщики денежных сумм и процентов
20  private static final NumberFormat currencyFormat =
21     NumberFormat.getCurrencyInstance();
22  private static final NumberFormat percentFormat =
23     NumberFormat.getPercentInstance();
24
25  private double billAmount = 0.0; // Сумма счета, введенная пользователем
26  private double percent = 0.15; // Исходный процент чаевых
27  private TextView amountTextView; // Для отформатированной суммы счета
28  private TextView percentTextView; // Для вывода процента чаевых
29  private TextView tipTextView; // Для вывода вычисленных чаевых
30  private TextView totalTextView; // Для вычисленной общей суммы
31
```

Величина счета, введенная пользователем в поле `amountEditText`, считывается и хранится в формате `double` в переменной `billAmount` (строка 25). Процент чаевых (целое число в диапазоне 0–30), введенный посредством перемещения ползунка `SeekBar`, делится на 100; полученное значение `double` используется в вычислениях и сохраняется в переменной `customPercent` (строка 26). Например,

если выбрать с помощью компонента `SeekBar` значение 25, то в переменной `percent` будет сохранено значение 0,25, а приложение будет умножать величину счета на 0,25 для вычисления 25% чаевых.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 3.3

В точных финансовых вычислениях для представления денежных сумм используйте класс BigDecimal (пакет java.math) вместо типа double.

В строке 27 объявляется компонент `TextView` для отображения выставленного счета, отформатированного в виде денежной суммы. В строке 28 объявляется компонент `TextView` для вывода процента чаевых, полученного на основании положения ползунка `SeekBar` (15% на рис. 3.1, *а*). Переменные в строках 29–30 ссылаются на компоненты `TextView`, в которых приложение выводит вычисленные чаевые и сумму счета.

3.6.4. Переопределение метода onCreate класса Activity

Метод `onCreate` (см. листинг 3.4, строки 33–36) *генерируется автоматически* при создании проекта приложения и вызывается системой после *запуска* класса активности. Метод `onCreate` обычно инициализирует переменные экземпляра `Activity` и компоненты GUI. Этот метод должен быть предельно упрощен, чтобы приложение загружалось быстро. Фактически в случае, если загрузка приложения занимает более пяти секунд, операционная система отображает диалоговое окно ANR (Application Not Responding, приложение не отвечает). В этом окне пользователю предоставляется возможность принудительно завершить приложение. О том, как устранить эту проблему, рассказано в главе 9.

Листинг 3.4. Переопределение метода onCreate класса Activity

```
32  // Вызывается при первом создании активности
33  @Override
34  protected void onCreate(Bundle savedInstanceState) {
35     super.onCreate(savedInstanceState); // Вызов версии суперкласса
36     setContentView(R.layout.activity_main); // Заполнение GUI
37     // Получение ссылок на компоненты TextView, с которыми
38     // MainActivity взаимодействует на программном уровне
39     amountTextView = (TextView) findViewById(R.id.amountTextView);
40     percentTextView = (TextView) findViewById(R.id.percentTextView);
41     tipTextView = (TextView) findViewById(R.id.tipTextView);
42     totalTextView = (TextView) findViewById(R.id.totalTextView);
43     tipTextView.setText(currencyFormat.format(0)); // Заполнить 0
44     totalTextView.setText(currencyFormat.format(0)); // Заполнить 0
45
46     // Назначение слушателя TextWatcher для amountEditText
```

```
47      EditText amountEditText =
48         (EditText) findViewById(R.id.amountEditText);
49      amountEditText.addTextChangedListener(amountEditTextWatcher);
50
51      // Назначение слушателя OnSeekBarChangeListener для percentSeekBar
52      SeekBar percentSeekBar =
53         (SeekBar) findViewById(R.id.percentSeekBar);
54      percentSeekBar.setOnSeekBarChangeListener(seekBarListener);
55   }
56
```

Параметр Bundle метода onCreate

Во время выполнения пользователь может изменить конфигурацию устройства, повернув его, подключив клавиатуру Bluetooth или выдвинув аппаратную клавиатуру. Качественное приложение должно легко справляться с такими изменениями конфигурации. При вызове системой метода `onCreate` передается аргумент `Bundle` с сохраненным состоянием `Activity` (если оно имеется). Как правило, состояние сохраняется в методах `onPause` или `onSaveInstanceState` активности (эта возможность продемонстрирована в последующих приложениях). Строка 35 вызывает метод `onCreate` суперкласса; этот вызов обязателен при переопределении `onCreate`.

Сгенерированный класс R содержит идентификаторы ресурсов

При построении графического интерфейса приложения и добавлении в приложение ресурсов (например, строк в файле `strings.xml` или компонентов в файле `activity_main.xml`) среда разработки генерирует класс с именем `R`, который содержит вложенные классы, представляющие каждый тип ресурсов из папки `res` вашего проекта. Вложенные классы объявляются статическими, так что вы можете обращаться к ним в коде с использованием синтаксиса `R.`*ИмяКласса*. В классах, вложенных в класс `R`, создаются константы `static final int`, с помощью которых можно обращаться к ресурсам из кода приложения (вскоре вы увидите, как это делается). Некоторые классы, вложенные в класс R:

- Класс `R.drawable` содержит константы всех визуальных элементов (например, изображений), которые находятся в различных папках `drawable` в папке приложения `res`.
- Класс `R.id` содержит константы для *представлений* (визуальных компонентов), используемые в *файлах XML-разметки*.
- Класс `R.layout` содержит константы, которые представляют *макетные файлы* проекта (например, `activity_main.xml`).
- Класс `R.string` содержит константы, используемые для определения строк в файле `strings.xml`.

Заполнение графического интерфейса

При вызове `setContentView` (строка 36) передается константа `R.layout.activity_main`, определяющая XML-файл с графическим интерфейсом `MainActivity`, — в данном случае она определяет файл `activity_main.xml`. Метод `setContentView` использует эту константу для загрузки соответствующего документа XML, который разбирается и преобразуется в графический интерфейс приложения. Этот процесс называется *заполнением* (inflating) графического интерфейса.

Получение ссылок на виджеты

После того как заполнение графического интерфейса будет завершено, вы сможете получать ссылки на конкретные виджеты для выполнения программных операций с ними. Для этого используется метод `findViewById` класса `Activity`. Метод получает константу `int`, представляющую идентификатор конкретного виджета, и возвращает ссылку на него. Имя константы `R.id` каждого виджета определяется свойством `Id`, заданным при проектировании графического интерфейса. Например, компоненту `amountEditText` соответствует константа `R.id.amountEditText`.

Строки 39–42 получают ссылки на компоненты `TextView`, изменяемые приложением. Строка 39 получает ссылку на компонент `amountTextView`, обновляемый при вводе выставленного счета. Строка 40 получает ссылку на компонент `percentTextView`, который обновляется при изменении пользовательского процента чаевых. В строках 41–42 получаются ссылки на компоненты `TextView`, в которых отображаются вычисленные чаевые и итоговая сумма.

Вывод исходных значений в TextView

В строках 43–44 свойства `text` компонентов `tipTextView` и `totalTextView` заполняются исходным значением 0, отформатированным по правилам денежных сумм для текущего локального контекста (для этого вызывается метод `format` объекта `currencyFormat`). Текст в обоих компонентах будет изменяться в процессе ввода суммы счета пользователем.

Регистрация слушателей событий

В строках 47–49 мы получаем ссылку на компонент `amountEditText` и вызываем его метод `addTextChangedListener` для регистрации слушателя `TextWatcher`, который будет реагировать на события, генерируемые при изменении текста в `EditText`. В нашей программе этот слушатель (листинг 3.6) определяется как объект анонимного внутреннего класса, который присваивается переменной экземпляра `amountEditTextWatcher`. Хотя вместо `amountEditTextWatcher` в строке 49 можно было определить анонимный внутренний класс, мы решили привести определение позднее в классе, чтобы код лучше читался.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 3.4

Вместо того чтобы определять анонимные внутренние классы в больших методах, определяйте их как переменные экземпляров со спецификаторами private final, чтобы упростить чтение, правку и сопровождение кода.

В строках 52–53 мы получаем ссылку на компонент `percentSeekBar`, а в строке 54 вызывается его метод `setOnSeekBarChangeListener` для регистрации слушателя `OnSeekBarChangeListener`, который будет реагировать на события, генерируемые при перемещении ползунка `SeekBar` для изменения пользовательского процента чаевых. Слушатель (листинг 3.7) определяется как объект анонимного внутреннего класса, который присваивается переменной экземпляра `SeekBarListener`.

О связывании данных в Android 6

В Android теперь существует библиотека поддержки связывания данных Data Binding, которая может использоваться с приложениями Android для Android 2.1 (API уровня 7) и выше. Теперь в разметку XML макета можно включать выражения связывания с данными, которые работают с объектами Java и динамически обновляют данные в пользовательских интерфейсах ваших приложений.

Кроме того, у каждого файла с разметкой XML макета, содержащего компоненты с идентификаторами (`id`), также имеется соответствующий автоматически сгенерированный класс. Для каждого компонента с идентификатором этот класс содержит `public final`-переменную экземпляра, которая ссылается на этот компонент. Вы можете создать экземпляр этого класса `Binding`, чтобы заменить все вызовы `findViewById`; это сильно упростит методы `onCreate` в классах активностей и фрагментов со сложными интерфейсами пользователя. Имя каждой переменной экземпляра соответствует идентификатору, заданному в макете для соответствующего компонента. Фактическое имя класса `Binding` строится на базе имени макета — так, для файла `activity_main.xml` будет сгенерирован класс с именем `ActivityMainBinding`.

На момент написания книги библиотека Data Binding существовала в ранней бета-версии, которую неизбежно ожидали существенные изменения (как в синтаксисе выражений связывания данных, так и в поддержке инструментария Android Studio). За дополнительной информацией о связывании данных в Android обращайтесь по адресу

https://developer.android.com/tools/data-binding/guide.html

3.6.5. Метод calculate класса MainActivity

Метод `calculate` (листинг 3.5) вызывается слушателями `EditText` и `SeekBar` для обновления компонентов `TextView` с чаевыми и общей суммой счета каждый раз, когда пользователь *изменяет* выставленный счет. В строке 60 процент чаевых

выводится в `percentTextView`. В строках 63–64 вычисляются чаевые и общая сумма на основании введенного значения `billAmount`. В строках 67–68 значения выводятся в формате денежной суммы.

Листинг 3.5. Метод calculate класса MainActivity

```
57  // Вычисление и вывод чаевых и общей суммы
58  private void calculate() {
59     // Форматирование процентов и вывод в percentTextView
60     percentTextView.setText(percentFormat.format(percent));
61
62     // Вычисление чаевых и общей суммы
63     double tip = billAmount * percent;
64     double total = billAmount + tip;
65
66     // Вывод чаевых и общей суммы в формате денежной величины
67     tipTextView.setText(currencyFormat.format(tip));
68     totalTextView.setText(currencyFormat.format(total));
69  }
70
```

3.6.6. Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener

В строках 72–87 листинга 3.6 создается объект анонимного внутреннего класса, реагирующий на события `percentTipSeekBar`. Объект присваивается переменной экземпляра `seekBarListener`. В строке 54 (листинг 3.4) `seekBarListener` регистрируется как объект обработки события `OnSeekBarChangeListener` для компонента `percentSeekBar`. Для наглядности мы определяем так все объекты обработки событий, кроме самых простых, чтобы не загромождать метод `onCreate` этим кодом.

Листинг 3.6. Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener

```
71  // Объект слушателя для событий изменения состояния SeekBar
72  private final OnSeekBarChangeListener seekBarListener =
73     new OnSeekBarChangeListener() {
74        // Обновление процента чаевых и вызов calculate
75        @Override
76        public void onProgressChanged(SeekBar seekBar, int progress,
77           boolean fromUser) {
78           percent = progress / 100.0; // Назначение процента чаевых
79           calculate(); // Вычисление и вывод чаевых и суммы
80        }
81
82        @Override
83        public void onStartTrackingTouch(SeekBar seekBar) { }
84
```

```
85         @Override
86         public void onStopTrackingTouch(SeekBar seekBar) { }
87     };
88
```

Переопределение метода onProgressChanged интерфейса OnSeekBarChangeListener

В строках 75–86 реализуются методы интерфейса `OnSeekBarChangeListener`. Метод `onProgressChanged` вызывается при каждом изменении позиции ползунка `SeekBar`. Строка 78 вычисляет `percent` на основании параметра `progress` метода — значения типа `int`, представляющего позицию ползунка `SeekBar`. Пользовательский процент чаевых вычисляется делением значения на 100. В строке 79 вызывается метод `calculate` для пересчета и отображения пользовательских чаевых и общей суммы.

Переопределение методов onStartTrackingTouch и onStopTrackingTouch интерфейса OnSeekBarChangeListener

В языке Java *каждый* метод реализуемого интерфейса должен переопределяться в программе. Так как нашему приложению не нужно знать, когда пользователь начинает перемещать ползунок (`onStartTrackingTouch`) или прекращает перемещать его (`onStopTrackingTouch`), мы просто предоставляем пустое тело для каждого метода (строки 82–86), чтобы выполнить контракт интерфейса.

Инструменты переопределения методов в Android Studio

Android Studio может создавать за вас пустые методы, переопределяющие методы, унаследованные от суперклассов, или реализующие методы интерфейса. Если поместить курсор в тело класса, а затем выполнить команду меню `Code` ▸ `Override Methods...`, IDE выведет диалоговое окно со списком всех методов, которые могут переопределяться в текущем классе. Список включает все унаследованные методы в иерархии класса, а также методы всех интерфейсов, реализуемых в иерархии класса.

ЗАЩИТА ОТ ОШИБОК 3.2

Команда меню Android Studio Code ▸ Override Methods... ускоряет написание кода и снижает вероятность ошибок.

3.6.7. Анонимный внутренний класс, реализующий интерфейс TextWatcher

В строках 90–114 листинга 3.7 создается объект анонимного внутреннего класса, реагирующий на события `amountEditText`. Этот объект присваивается переменной

экземпляра `amountEditTextWatcher`. В строке 49 этот объект регистрируется для прослушивания событий `amountEditText`, происходящих при изменении текста.

Листинг 3.7. Анонимный внутренний класс, реализующий интерфейс TextWatcher

```
 89  // Объект слушателя для событий изменения текста в EditText
 90  private final TextWatcher amountEditTextWatcher = new TextWatcher() {
 91     // Вызывается при изменении пользователем величины счета
 92     @Override
 93     public void onTextChanged(CharSequence s, int start,
 94        int before, int count) {
 95
 96        try { // Получить счет и вывести в формате денежной суммы
 97           billAmount = Double.parseDouble(s.toString()) / 100.0;
 98           amountTextView.setText(currencyFormat.format(billAmount));
 99        }
100        catch (NumberFormatException e) { // Если s пусто или не число
101           amountTextView.setText("");
102           billAmount = 0.0;
103        }
104
105        calculate(); // Обновление полей с чаевыми и общей суммой
106     }
107
108     @Override
109     public void afterTextChanged(Editable s) { }
110
111     @Override
112     public void beforeTextChanged(
113        CharSequence s, int start, int count, int after) { }
114  };
115 }
```

Переопределение метода onTextChanged интерфейса TextWatcher

Метод `onTextChanged` (строки 92–106) вызывается при каждом изменении текста, отображаемого в компоненте `amountEditText`. Метод получает четыре параметра. В рассматриваемом примере используется только параметр `CharSequence s`, который содержит копию текста `amountEditText`. Другие параметры сообщают о том, что текст, количество символов которого определено параметром `count`, начинающийся в позиции `start`, заменил фрагмент прежнего текста длины `before`.

В строке 97 текст, введенный пользователем в компоненте `amountEditText`, преобразуется в тип `double`. Наше приложение разрешает пользователю вводить только целые числа, поэтому для получения фактического выставленного счета преобразованное значение делится на 100. Например, если пользователь ввел 2495, то величина счета составит 24,95. В строке 98 выводится обновленная величина счета. При возникновении исключения строки 101–102 сбрасывают содержимое `amountTextView` и присваивают `billAmount` значение 0. Строка 105 вызывает `calculate` для пересчета и отображения чаевых и общей суммы.

Другие методы amountEditTextWatcher

Нашему приложению не нужно знать, когда начинается внесение изменений в текст (`beforeTextChanged`) или что текст уже был изменен (`afterTextChanged`), поэтому мы просто переопределяем каждый из этих методов интерфейса `TextWatcher` пустым телом (строки 108–113), чтобы выполнить контракт интерфейса.

3.7. Файл AndroidManifest.xml

В этом разделе мы отредактируем файл `AndroidManifest.xml`, чтобы указать, что активность приложения поддерживает только портретную ориентацию устройства, а виртуальная клавиатура должна отображаться постоянно после появления активности на экране или возвращения к ней. Чтобы открыть манифест, сделайте двойной щелчок на файле `AndroidManifest.xml` приложения в папке `manifests` окна `Project`. В листинге 3.8 приведен полный манифест с выделенными изменениями — остальная разметка была автоматически сгенерирована Android Studio при создании проекта. Мы рассмотрим некоторые аспекты манифеста, а за полным списком элементов, которые могут содержаться в манифесте, атрибутов и их значений обращайтесь по адресу

http://developer.android.com/guide/topics/manifest/manifest-intro.html

Листинг 3.8. Содержимое манифеста AndroidManifest.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
 3    package="com.deitel.tipcalculator" >
 4
 5    <application
 6       android:allowBackup="true"
 7       android:icon="@mipmap/ic_launcher"
 8       android:label="@string/app_name"
 9       android:supportsRtl="true"
10       android:theme="@style/AppTheme" >
11       <activity
12          android:name=".MainActivity"
13          android:label="@string/app_name"
14          android:screenOrientation="portrait"
15          android:windowSoftInputMode="stateAlwaysVisible">
16          <intent-filter>
17             <action android:name="android.intent.action.MAIN" />
18
19             <category android:name="android.intent.category.LAUNCHER" />
20          </intent-filter>
21       </activity>
22    </application>
23
24 </manifest>
```

3.7.1. Элемент manifest

Элемент `manifest` (строки 2–24) указывает, что содержимое файла XML представляет манифест приложения. Атрибут `package` этого элемента задает имя пакета Java, введенное при создании проекта приложения (раздел 3.4.2). Напомним, что для приложений, отправляемых в магазин Google Play, имя пакета используется в качестве уникального идентификатора приложения.

3.7.2. Элемент application

Элемент `application`, вложенный в элемент `manifest` (строки 5–21), задает атрибуты приложения, в числе которых:

- `android:allowBackup` — флаг автоматического резервного копирования данных приложения для последующего восстановления на этом или другом устройстве.
- `android:icon` — значок, используемый в лаунчере для запуска приложения.
- `android:label` — название приложения, которое обычно отображается под значком в лаунчере и часто отображается на панели приложения во время его выполнения.
- `android:supportsRtl` — флаг возможности горизонтального отражения интерфейса приложения для поддержки языков, в которых буквы пишутся справа налево (арабский, иврит).
- `android:theme` — тема, описывающая оформление компонентов приложения по умолчанию.

Элементы, вложенные в элемент `application`, определяют основные составляющие приложения (например, его активности).

3.7.3. Элемент activity

Элемент `activity`, вложенный в элемент `application` (строки 10–20), описывает *активность* приложения. Приложение может иметь много активностей, одна из которых назначается активностью, которая должна отображаться при запуске приложения значком в лаунчере. Каждый элемент `activity` задает как минимум следующие атрибуты.

- `android:name` — имя класса активности. Запись `".MainActivity"` является сокращением для `"com.deitel.MainActivity"` (где `com.deitel` — доменное имя, указанное при создании проекта приложения, записанное в обратном порядке).

- `android:label` — имя активности, часто отображаемое на панели приложения во время нахождения активности на экране. Для приложений с одной активностью это имя обычно совпадает с именем приложения.

Для `MainActivity` были добавлены следующие атрибуты.

- `android:screenOrientation` — большинство приложений поддерживает как портретную, так и альбомную ориентацию. В портретной ориентации высота устройства больше его ширины. В альбомной ориентации ширина устройства больше высоты. В приложении `Tip Calculator` переход устройства в альбомную ориентацию на типичном телефоне приведет к тому, что цифровая клавиатура закроет большую часть графического интерфейса. По этой причине мы задаем этому свойству значение `"portrait"`, чтобы приложение поддерживало только портретную ориентацию.
- `android:windowSoftInputMode` — в приложении `Tip Calculator` виртуальная клавиатура должна появиться сразу же после запуска приложения и снова появляться каждый раз, когда пользователь возвращается к приложению. По этой причине свойству задается значение `"stateAlwaysVisible"`. При наличии аппаратной клавиатуры в этом режиме виртуальная клавиатура отображаться не будет.

3.7.4. Элемент intent-filter

Механизм *интентов* (intent) используется в Android для организации взаимодействия между выполняемыми компонентами — активностями, фоновыми службами и операционной системой. Приложение формирует интент, а Android использует механизм передачи интентов для координации работы исполняемых компонентов, выполняющих нужную операцию. Нежесткое связывание такого рода упрощает использование частей других приложений. Вы сообщаете Android, что нужно сделать, а Android находит установленные приложения с активностями, способными выполнить нужную операцию.

Взаимодействия между приложениями

Один из примеров использования интентов — координация работы разных приложений. Для примера можно рассмотреть возможную организацию обмена фотографиями в Android.

- Многие приложения Android для социальных сетей предоставляют собственные возможности обмена фотографиями. Каждое приложение в своем манифесте объявляет конкретную активность, которая отправляет фотографию на учетную запись пользователя.

- Другие приложения могут использовать готовые средства обмена фотографиями, вместо того чтобы реализовать их самостоятельно. Например, фоторедактор может предоставить команду для публикации фотографии. Приложение реагирует на запрос пользователя, создавая интент для передачи фотографии и передавая его Android.
- Android обращается к интенту и определяет, какие из установленных приложений предоставляют интенты, способные публиковать фотографии.
- Если такое приложение только *одно*, Android выполняет соответствующую активность этого приложения.
- Если таких приложений *несколько*, то Android выводит список приложений и предлагает пользователю решить, какая активность приложения должна выполняться.

Важнейшее преимущество этого механизма заключается в том, что разработчику фоторедактора не нужно встраивать в свое приложение поддержку всех существующих социальных сетей. Выдача интента публикации фотографии обеспечивает автоматическую поддержку всех приложений, объявляющих в своем манифесте активность публикации фотографий — как уже установленных, так и тех, которые пользователь захочет установить в будущем. За списком значений, которые могут использоваться с интентами, обращайтесь по адресу

http://developer.android.com/reference/android/content/Intent.html#constants

Выполнение приложений

Интенты также часто используются при запуске активностей. Касаясь значка приложения в лаунчере устройства, вы намереваетесь выполнить приложение. При этом лаунчер выдает интент для выполнения главной активности этого приложения (см. ниже). Android реагирует на интент запуском приложения и выполнением конкретной активности, указанной в манифесте приложения как главной активности.

Определение выполняемой активности

Android использует информацию в манифесте для определения активностей, которые могут реагировать на интенты, и интентов, которые могут обрабатываться приложением. В манифесте элемент `intent-filter`, вложенный в элемент `activity` (листинг 3.8, строки 16–20), определяет типы интентов, которые могут запускать данную активность. Если интент соответствует содержимому `intent-filter` только одной активности, то Android выполняет эту активность. При наличии нескольких совпадений Android выводит список, из которого пользователь может выбрать приложение, после чего выполняет соответствующую активность этого приложения.

Android также передает интент активности, потому что интент часто содержит данные, необходимые активности для выполнения ее операции. Например, для фоторедактора в интент публикации фотографии может быть включена фотография, которую нужно опубликовать.

Элемент `intent-filter` должен содержать один или несколько элементов `action`. Действие `"android.intent.action.MAIN"` в строке 17 листинга 3.8 означает, что `MainActivity` является главной активностью, которая должна выполняться при запуске приложения. Необязательный элемент `category` в строке 19 указывает, что является источником интента — для `"android.intent.category.LAUNCHER"` это лаунчер устройства. Данное значение также указывает, что активность должна отображаться в виде значка в лаунчере устройства вместе со значками других установленных приложений пользователя.

В следующей главе интенты будут рассмотрены более подробно. За дополнительной информацией об интентах и фильтрах интентов обращайтесь по адресу

http://developer.android.com/guide/components/intents-filters.html

3.8. Резюме

В этой главе вы создали свое первое интерактивное приложение Android — `Tip Calculator`. Сначала были вкратце рассмотрены функции этого приложения, потом мы опробовали его на практике для вычисления чаевых и общей суммы. Графический интерфейс приложения был построен в макетном редакторе среды разработки Android Studio, окне `Component Tree` и окне свойств. Мы также отредактировали разметку XML макета и воспользовались редактором `Theme Editor` для настройки заданных при создании проекта цветов (основного, темного основного и акцентного) темы приложения `Theme.AppCompat.Light.DarkActionBar`. Также был рассмотрен код `MainActivity` — субкласса `AppCompatActivity` (и потомка `Activity`), определяющего логику приложения.

При разработке графического интерфейса приложения компонент `GridLayout` использовался для распределения компонентов GUI по строкам и столбцам. Текст выводился в компонентах `TextView`, а для ввода данных использовались компоненты `EditText` и `SeekBar`.

В классе `MainActivity` задействованы многие средства объектно-ориентированного программирования на языке Java, включая классы, объекты, интерфейсы, анонимные внутренние классы и наследование. Также была представлена концепция заполнения графического интерфейса, то есть преобразования содержимого файла XML в его экранное представление. Вы познакомились с классом Android `Activity` и жизненным циклом активности. В частности,

мы переопределили метод `onCreate` для инициализации приложения при запуске. В методе `onCreate` метод `findViewById` класса `Activity` использовался для получения ссылок на визуальные компоненты, с которыми приложение взаимодействует на программном уровне. Мы определили анонимный внутренний класс, реализующий интерфейс `TextWatcher`, чтобы приложение могло вычислять новые чаевые и общую сумму при изменении текста в `EditText`. Также был определен анонимный внутренний класс, реализующий интерфейс `OnSeekBarChangeListener`, чтобы приложение могло вычислить новые чаевые и общую сумму при изменении пользовательского процента чаевых перемещением ползунка `SeekBar`.

Наконец, мы отредактировали файл `AndroidManifest.xml`, чтобы указать, что `MainActivity` поддерживает только портретную ориентацию, а класс `MainActivity` всегда должен отображать виртуальную клавиатуру. Заодно были представлены другие элементы, включенные Android Studio в манифест при создании проекта.

В главе 4 будет создано приложение `Flag Quiz`, на экране которого отображается флаг определенной страны, название которой предстоит выбрать пользователю из двух, четырех, шести или восьми вариантов. С помощью меню и флажков вы сможете настроить приложение, ограничив количество отображаемых флагов и стран определенными регионами мира.

4 Приложение Flag Quiz

Фрагменты, меню, настройки, явные интенты, обработчики, AssetManager, анимация с переходами, уведомления Toast, списки цветов состояний, макеты для разных ориентаций устройства, регистрация сообщений об ошибках

В этой главе…

- Использование фрагментов для эффективной организации пространства в графическом интерфейсе Activity на телефонах и планшетах
- Отображение меню на панели действий для настройки параметров приложения
- Использование объектов PreferenceFragment для автоматического управления настройками пользователя
- Использование SharedPreferences.Editor для изменения пар «ключ—значение», связанных с приложением
- Использование папок assets для организации графических ресурсов и работы с ними средствами AssetManager
- Определение анимаций и их применение к View
- Планирование выполняемых в будущем действий с помощью Handler
- Использование объектов Toast для кратковременного отображения сообщений
- Запуск конкретной активности с помощью явного интента
- Коллекции из пакета java.util
- Определение макетов для разных ориентаций устройства
- Использование механизма регистрации Android для регистрации сообщений об ошибках

4.1. Введение

Приложение `Flag Quiz` проверяет знание пользователем 10 флагов различных стран (рис. 4.1). После запуска приложения на экране появляется изображение флага и четыре кнопки с вариантами возможного ответа. Один из вариантов правильный, а остальные (неправильные) выбираются случайным образом без повторений. Приложение выводит информацию о ходе игры, отображая номер вопроса (из десяти возможных) в компоненте `TextView`, находящемся над изображением текущего флага.

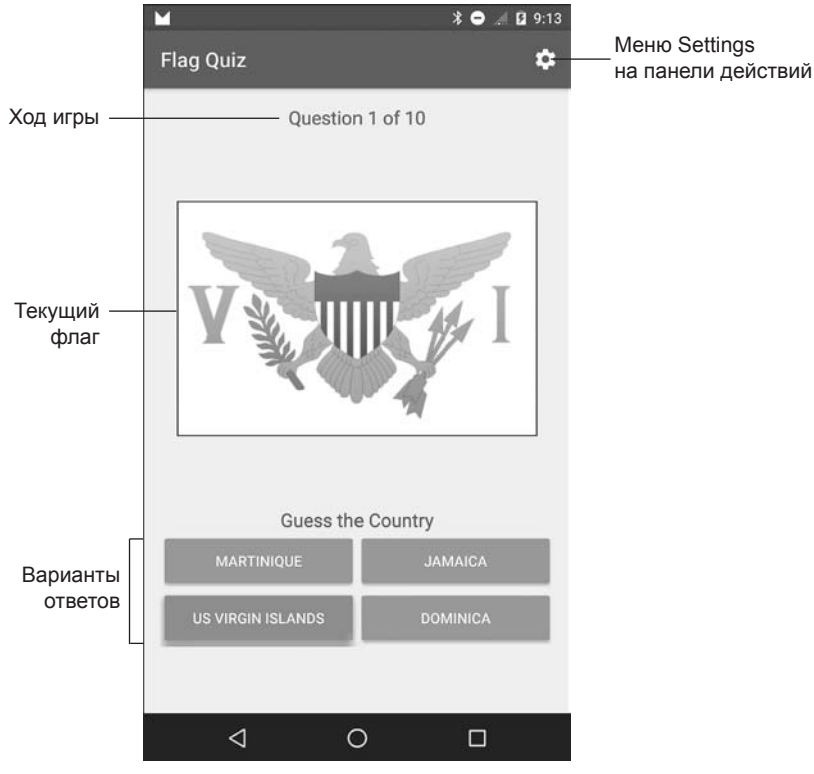


Рис. 4.1. Приложение Flag Quiz на смартфоне в портретной ориентации

Как будет показано ниже, приложение также позволяет управлять сложностью игры: пользователь может выбрать между двумя, четырьмя, шестью или восемью вариантами ответа, а также выбором регионов, которые должны быть включены в опрос. Формат отображения вариантов зависит от устройства, на котором выполняется приложение, и ориентации экрана — приложение поддерживает портретную ориентацию на любом устройстве, но альбомная ориентация поддерживается только на планшетах.

В портретной ориентации приложение выводит на панели действий меню со значком `Settings` (⚙). Когда пользователь выбирает эту команду, приложение открывает отдельный экран (другую активность) для определения количества вариантов ответа и регионов. На планшете в альбомной ориентации (рис. 4.2) используется другой макет, при котором настройки приложения отображаются в левой части экрана, а флаги — справа.

Начнем с тестового запуска, после чего будет приведен обзор технологий, задействованных при его создании. Затем мы построим графический интерфейс приложения. Глава завершается анализом полного исходного кода приложения с более подробным анализом новых возможностей.

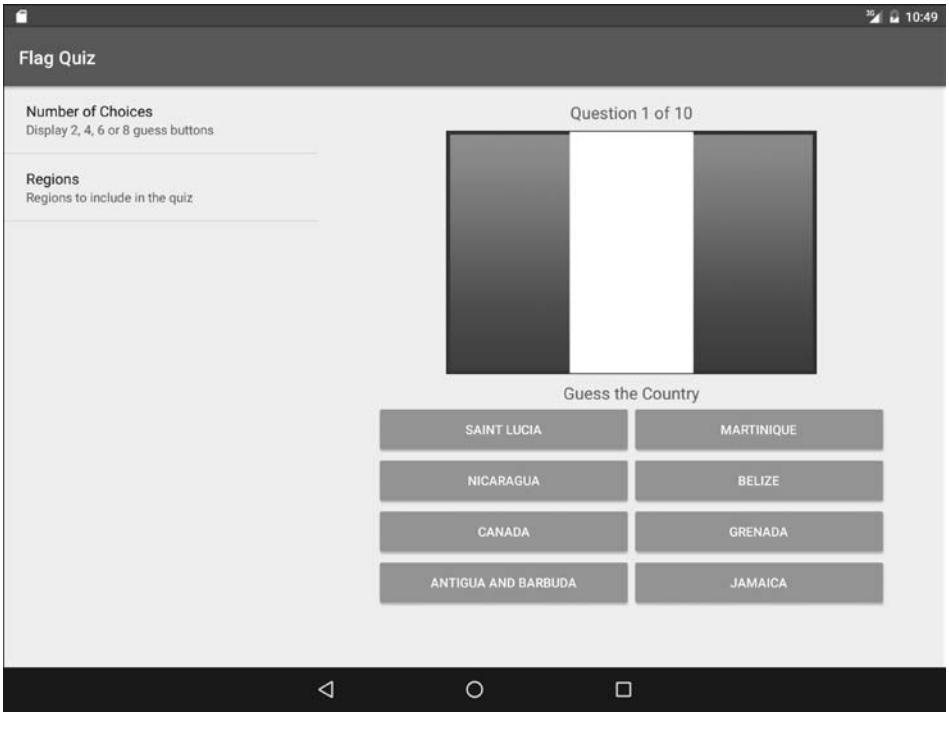


Рис. 4.2. Приложение Flag Quiz на планшете в альбомной ориентации

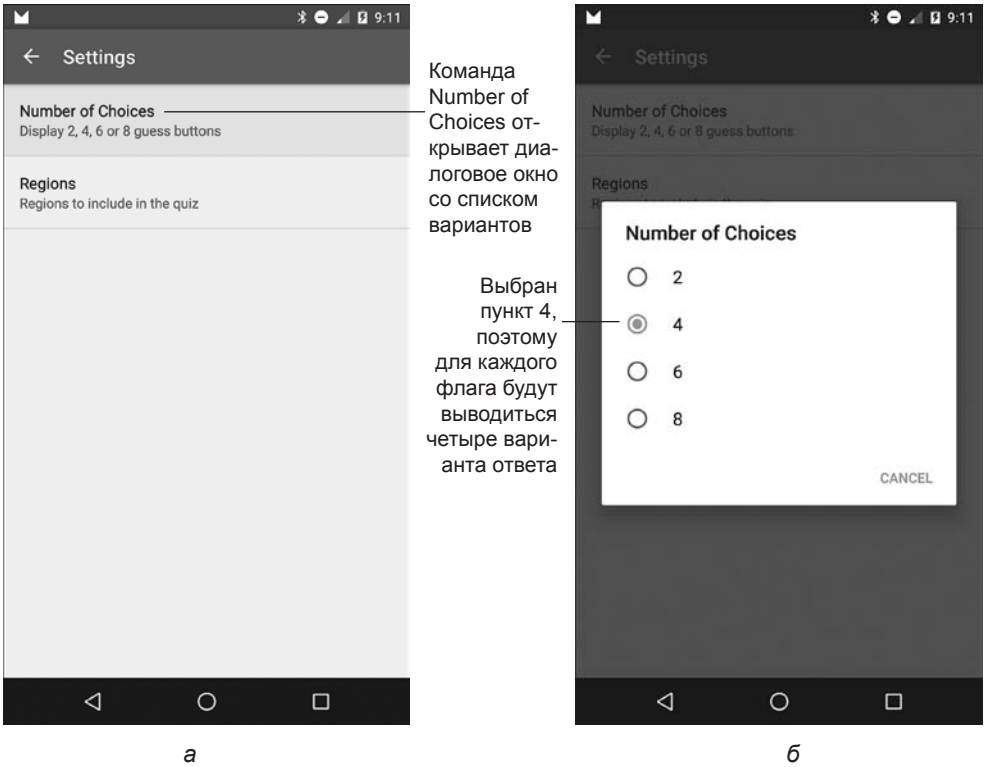
4.2. Тестирование приложения Flag Quiz

Откройте Android Studio, откройте приложение `Flag Quiz` из папки `Flag Quiz` в папке примеров и импортируйте проект приложения `Flag Quiz`. Приложение можно протестировать на виртуальном (AVD) или физическом устройстве. Среда строит проект и запускает приложение (см. рис. 4.1 и 4.2).

4.2.1. Настройка викторины

При установке и первом запуске приложение настраивается так, чтобы для каждого вопроса выводились четыре варианта ответов и флаги из *всех* регионов. В этом тестовом запуске мы изменим конфигурацию так, чтобы выводились флаги только из Северной Америки; для количества вариантов будет оставлено значение по умолчанию (четыре).

На телефоне, планшете или AVD в портретной ориентации коснитесь кнопки меню настроек на панели приложения (⚙) (см. рис. 4.1); на экране появляется диалоговое окно `Settings` (рис. 4.3, *а*). На планшетном устройстве или AVD в *альбомной* ориентации команды меню настроек отображаются в левой части экрана (рис. 4.2). Чтобы открыть диалоговое окно для выбора количества вариантов ответа, отображаемых для каждого флага (рис. 4.3, *б*), коснитесь кнопки `Number of Choices`. (На планшетном устройстве или AVD в альбомной ориентации весь экран окрашивается в серый цвет, а диалоговое окно отображается в центре

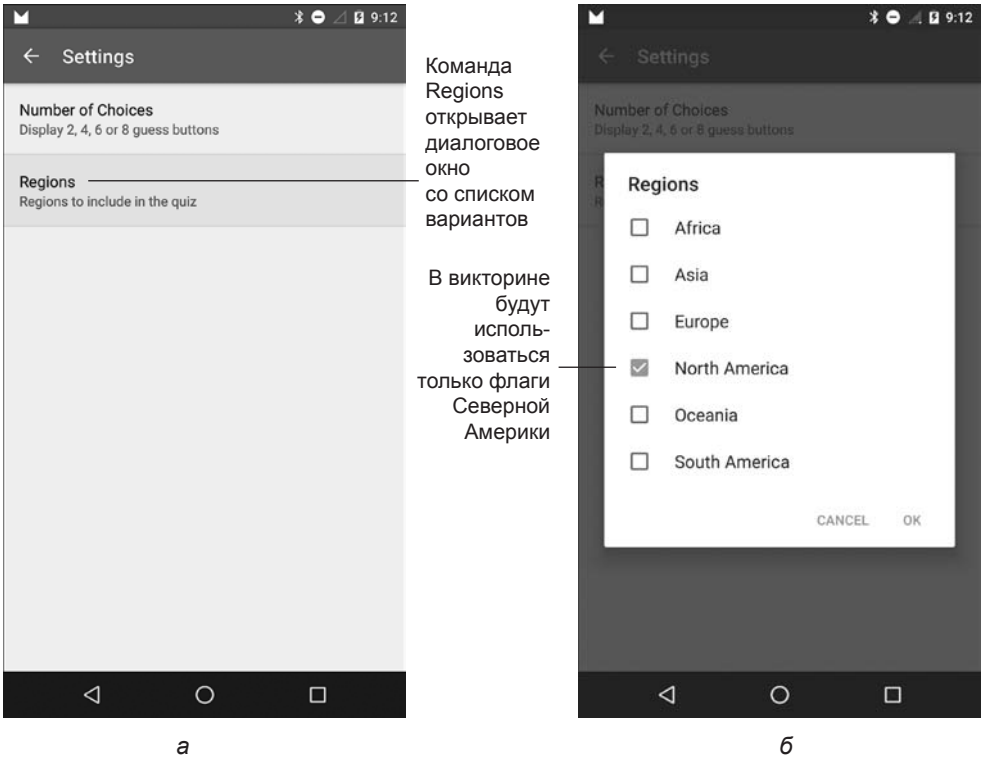


а *б*

Рис. 4.3. Экран настроек Flag Quiz и диалоговое окно Number of Choices: *а* — пользователь выбирает в меню команду Number of Choices; *б* — диалоговое окно для выбора количества вариантов ответа

экрана.) По умолчанию для каждого флага отображаются четыре варианта ответа — это значение используется по умолчанию. Чтобы повысить сложность викторины, выберите вариант `6` или `8` и коснитесь кнопки `OK`; если вы не хотите изменять сложность, коснитесь кнопки `CANCEL`, чтобы вернуться к окну `Settings`.

Коснитесь кнопки `Regions` (рис. 4.4, *а*), чтобы отобразить набор флажков, представляющих регионы мира (рис. 4.4, *б*). По умолчанию после первого запуска приложения отображаются все регионы, поэтому для викторины случайным образом выбираются флаги любых стран мира. Коснитесь флажков `Africa`, `Asia`, `Europe`, `Oceania` и `South America`, чтобы исключить эти регионы из викторины. Коснитесь кнопки `OK`, чтобы сохранить настройки. На телефонах, планшетах или AVD в портретной ориентации коснитесь кнопки `Back` (◁), чтобы вернуться к викторине и начать новую игру с изменившимися настройками. На планшетном устройстве или AVD в альбомной ориентации в правой части экрана немедленно отображается очередной вопрос с обновленной конфигурацией.



а *б*

Рис. 4.4. Экран настроек Flag Quiz и диалоговое окно Regions: *а* — пользователь выбирает в меню команду Regions; *б* — диалоговое окно для выбора регионов

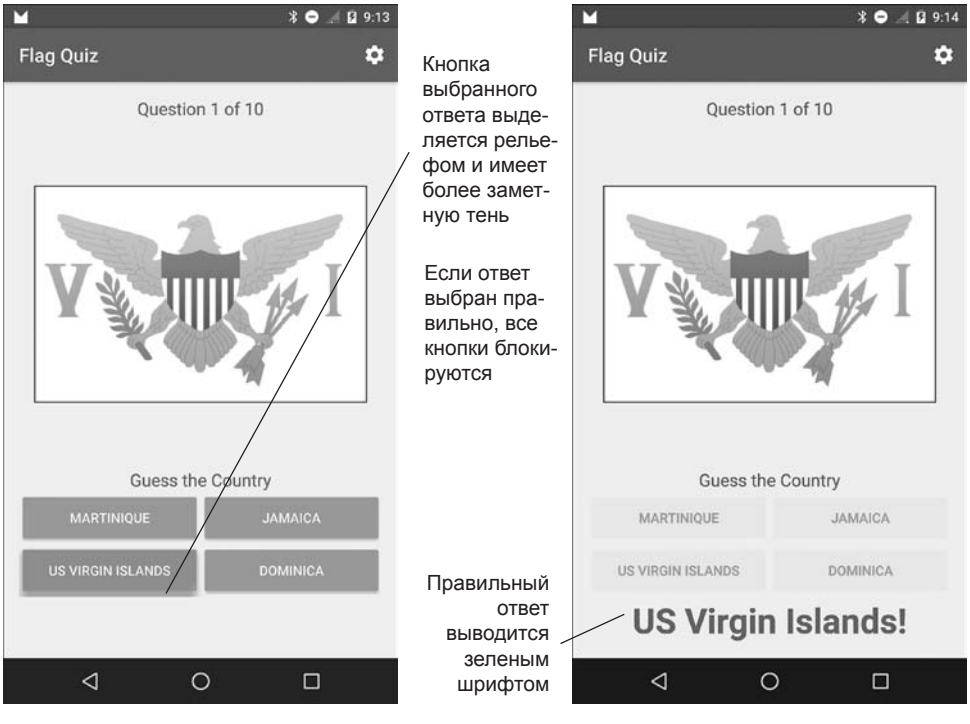
4.2.2. Ответы на вопросы викторины

Начинается новая викторина с выбранным количеством вариантов и флагами, ограниченными североамериканским регионом. Попробуйте ответить на вопрос: коснитесь кнопки той страны, которой (по вашему мнению) принадлежит этот флаг.

Выбирается правильный ответ

Если вариант был выбран правильно (рис. 4.5, *а*), приложение блокирует все кнопки ответов и выводит название страны зеленым шрифтом в нижней части экрана (рис. 4.5, *б*). После непродолжительной задержки приложение загружает следующий флаг с новым вариантом ответов. Переход приложения от текущего вопроса к следующему сопровождается анимацией.

- Сначала большой круг уменьшается на экране до тех пор, пока его диаметр не уменьшится до нуля. При этом скрывается флаг текущего вопроса и кнопки вариантов.



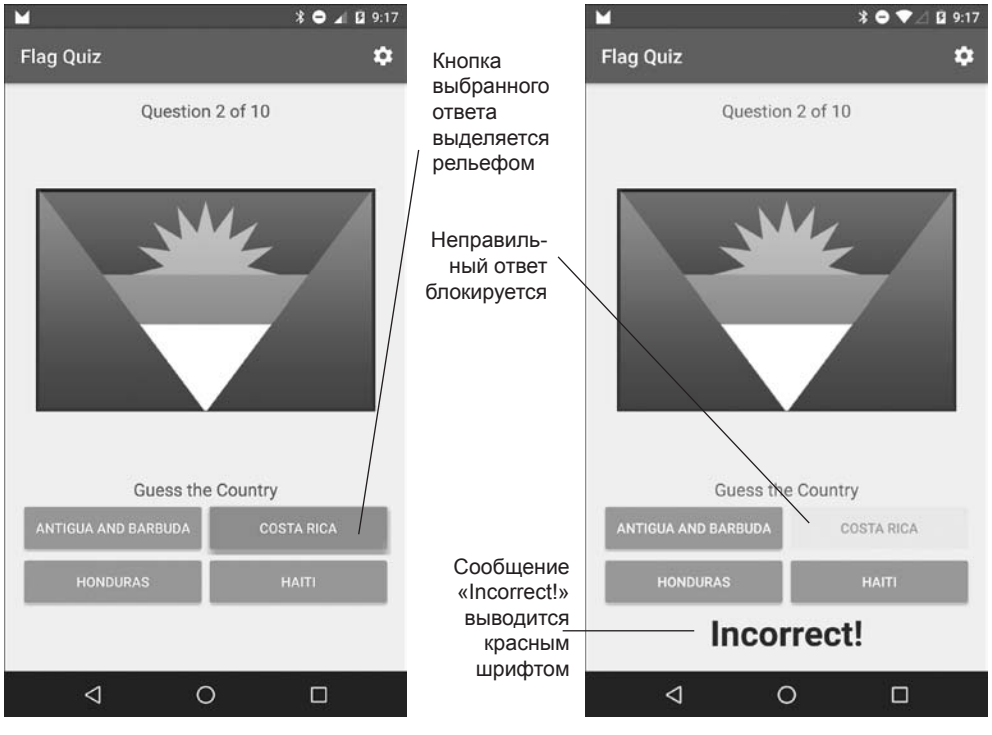
а *б*

Рис. 4.5. Пользователь выбрал правильный ответ: *а* — выбор правильного ответа; *б* — правильный ответ выводится на экране

- Затем круг снова увеличивается в размерах, пока на экране не будет виден флаг нового вопроса и кнопки вариантов.

Выбирается неправильный ответ

Если вариант выбран неправильно (рис. 4.6, *а*), приложение блокирует кнопку с названием страны, использует анимацию, чтобы «покачать» флагом, и выводит в нижней части экрана сообщение `Incorrect!` красным шрифтом (рис. 4.6, *б*). Продолжайте попытки, пока не найдете правильный ответ для этого флага.



а *б*

Рис. 4.6. Блокировка неправильного ответа в приложении Flag Quiz: *а* — выбор неправильного ответа; *б* — выводится сообщение Incorrect

Завершение викторины

После успешного отгадывания стран, которым принадлежат десять отображаемых флагов, на экране появляется окно `AlertDialog`, в котором будет указано количество попыток отгадывания и процент правильных ответов (рис. 4.7). Диалоговое окно является модальным, и чтобы закрыть его, с ним необходимо взаимодействовать — немодальное окно можно закрыть кнопкой `Back` (◁) на

AVD или устройстве. Кнопка RESET QUIZ закрывает диалоговое окно и начинает новую игру с текущими настройками.

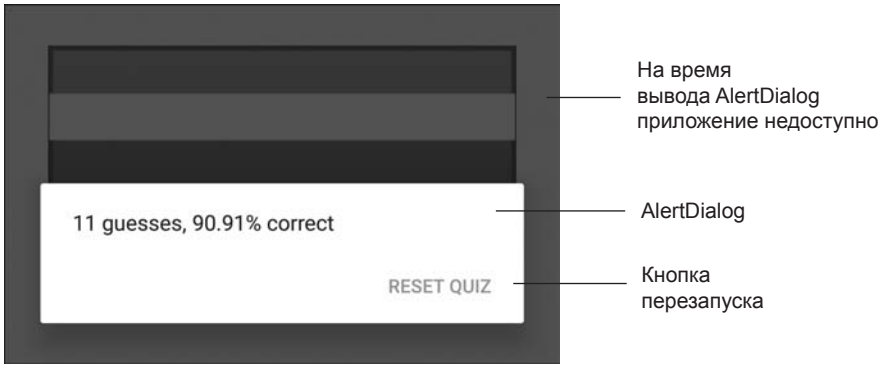


Рис. 4.7. Вывод результатов после завершения викторины

4.3. Обзор применяемых технологий

В этом разделе представлены новые возможности, которые будут использоваться при построении приложения Flag Quiz.

4.3.1. Меню

При создании проекта приложения в среде разработки класс MainActivity настраивается для отображения меню (⋮) в правой части панели действий. В этом приложении меню будет отображаться только в портретной ориентации. Меню, открываемое кнопкой ⋮, по умолчанию содержит только команду Settings, которая обычно используется для отображения настроек приложения. В этом приложении мы отредактируем файл с разметкой XML меню, определим значок (⚙) для команды меню Settings и укажем, что значок должен отображаться непосредственно на панели приложения. Пользователь сможет вызвать настройки приложения, коснувшись значка один раз — вместо того чтобы сначала открывать меню, а затем выбирать команду Settings. Для добавления значка настроек в проект будет использоваться редактор Vector Asset Studio среды Android Studio. В приложениях следующих глав вы узнаете, как создать новые команды меню.

Меню представляется объектом класса Menu (пакет android.view). Чтобы назначить команды Menu, следует переопределить метод onCreateOptionsMenu класса Activity (раздел 4.6.5) и добавить нужные команды в аргумент Menu

метода — либо на программном уровне, либо заполнением документа XML, описывающего команды меню. Когда пользователь выбирает команду меню, метод `onOptionsItemSelected` класса `Activity` (раздел 4.6.6) реагирует на выбор.

4.3.2. Фрагменты

Фрагмент (fragment) обычно представляет повторно используемую часть пользовательского интерфейса активности, но он также может представлять повторно используемый блок программной логики. Приложение использует фрагменты для создания частей графического интерфейса и управления ими. Фрагменты можно объединять для создания интерфейсов, эффективно использующих размер экрана планшета. Кроме того, простой механизм замены фрагментов сделает интерфейс приложения более динамичным (см. главу 9).

Базовым классом всех фрагментов является класс `Fragment` (пакет `android.app`). При использовании субклассов `AppCompatActivity` с фрагментами необходимо использовать версию этого класса из библиотеки Android Support Studio (пакет `android.support.v4.app`). Приложение `Flag Quiz` определяет следующие субклассы `Fragment` (с прямым или косвенным наследованием).

- Класс `MainActivityFragment` (раздел 4.7) — прямой субкласс `Fragment`; он формирует графический интерфейс и определяет логику викторины. По аналогии с `Activity` каждый объект `Fragment` имеет собственный макет, который обычно определяется как ресурс макета, но может создаваться динамически. Графический интерфейс `MainActivityFragment` строится в разделе 4.5.2. Мы создадим *два* макета для `MainActivity` — для устройств в портретной ориентации и для планшетных устройств в альбомной ориентации, а затем используем `MainActivityFragment` в обоих макетах.
- Класс `SettingsActivityFragment` (раздел 4.9) является субклассом `PreferenceFragment` (пакет `android.preference`), способного автоматически поддерживать пользовательские настройки в файле, связанном с приложением. Как вы увидите, разработчик может создать XML-файл с описанием настроек, который используется классом `PreferenceFragment` для построения соответствующего интерфейса (см. рис. 4.3–4.4). Работа с настройками более подробно рассматривается в разделе 4.3.5.
- Когда пользователь ответит на все вопросы, приложение создает анонимный внутренний класс, расширяющий `DialogFragment` (пакет `android.support.v4.app`), и использует его для отображения окна `AlertDialog` с результатами (раздел 4.7.10).

Фрагменты *должны* находиться под управлением активностей — они не могут выполняться автономно. Когда приложение работает на планшете в альбомной

ориентации, MainActivity управляет всеми фрагментами. В портретной ориентации (на любом устройстве) класс SettingsActivity (раздел 4.8) управляет SettingsActivityFragment, а MainActivity — всеми остальными фрагментами.

4.3.3. Методы жизненного цикла фрагментов

Каждый фрагмент, как и активность, имеет свой жизненный цикл и предоставляет методы, которые можно переопределять для обработки событий жизненного цикла. В этом приложении будут переопределены следующие методы:

- onCreate — этот метод (переопределяется в классе SettingsActivityFragment) вызывается при создании фрагмента. Объекты MainActivityFragment и SettingsActivityFragment создаются при заполнении макетов их родительских активностей, а объект DialogFragment, выводящий результаты, создается после завершения викторины;
- onCreateView — этот метод (переопределяется в классе MainActivityFragment) вызывается после onCreate; он должен построить и вернуть объект View с графическим интерфейсом фрагмента. Как вы вскоре увидите, он получает объект LayoutInflater, который используется для программного заполнения графического интерфейса фрагмента по компонентам, заданным в заранее определенном макете в формате XML.

Фрагменты могут добавлять свои команды в меню управляющей активности. Как класс Activity, класс Fragment содержит метод жизненного цикла onCreateOptionsMenu и метод обработки события onOptionsItemSelected.

Другие методы жизненного цикла фрагментов будут рассмотрены далее в книге. За полной информацией о подробностях жизненного цикла обращайтесь по адресу

http://developer.android.com/guide/components/fragments.html

4.3.4. Управление фрагментами

Родительская активность использует для управления своими фрагментами объект FragmentManager (пакет android.app), возвращаемый методом getFragmentManager класса Activity. Если активности потребуется взаимодействовать с фрагментом, который объявлен в макете активности и обладает идентификатором id, то для получения ссылки на заданный фрагмент активность может вызвать метод findFragmentById класса FragmentManager. Как будет показано в главе 9, FragmentManager может использовать объекты FragmentTransaction для динамического добавления, удаления и переключения между фрагментами.

4.3.5. Объекты Preference

В разделе 4.2.1 мы изменили настройки приложения. Эти настройки хранятся в файле в формате пар «ключ—значение»; ключ используется для быстрой выборки соответствующего значения. Ключи относятся к типу `String`, а значения могут быть объектами `String` или значениями примитивных типов. Для работы с файлом используется объект класса `SharedPreferences` (пакет `android.content`). Этот файл доступен только для приложения, создавшего файл.

`PreferenceFragment` использует объекты `Preference` (пакет `android.preference`) для управления настройками приложения и сохраняет эти настройки в файле при помощи объекта `SharedPreferences`. Приложение использует субкласс `Preference` с именем `ListPreference` для управления количеством вариантов ответов, отображаемых для каждого флага, а субкласс `Preference` с именем `MultiSelectListPreference` — для управления регионами, включаемыми в викторину. Варианты `ListPreference` являются взаимоисключающими (рис. 4.3, *б*), тогда как в `MultiSelectListPreference` можно выбрать любое количество вариантов (рис. 4.4, *б*). Для обращения к настройкам и работы с файлом `SharedPreference` по умолчанию используется объект `PreferenceManager` (пакет `android.preference`).

Также мы будем работать с файлом `SharedPreferences` приложения напрямую.

- При запуске викторины из файла настроек приложения читаются количество кнопок с вариантами и регионы, в которых должны находиться страны.
- Когда пользователь изменяет настройки регионов, приложение проверяет, что выбран хотя бы один регион; в противном случае не останется ни одного флага, который можно было бы включить в игру. Если ни один регион не выбран, приложение изменяет настройки и выбирает Северную Америку.

Для изменения содержимого `SharedPreferences` будет использоваться объект `SharedPreferences.Editor` (раздел 4.6.7).

4.3.6. Папка assets

Изображения флагов в приложении загружаются приложением только тогда, когда в них возникнет необходимость. Они находятся в папке `assets` приложения[1]. Чтобы добавить изображения в проект, мы скопировали в `assets` папки всех регионов из нашей файловой системы (раздел 4.4.4). Изображения можно найти в папке `images/FlagQuizImages` из папки примеров книги.

В отличие от папок `drawable`, в которых графический контент должен находиться на корневом уровне в каждой папке, папка `assets` может включать файлы

[1] Изображения флагов были взяты на веб-сайте *www.free-country-flags.com*.

произвольного типа, которые могут быть упорядочены по вложенным папкам (изображения флагов стран каждого региона находятся в отдельной подпапке). Доступ к файлам, находящимся в папках `assets`, обеспечивается с помощью диспетчера `AssetManager` (пакет `android.content.res`), который может предоставить список всех имен файлов в заданной папке, вложенной в `assets`, а также может применяться для обращения к каждому отдельному изображению.

4.3.7. Папки ресурсов

В разделе 2.4.4 вы узнали о папках `drawable`, `layout` и `values`, находящихся в папке `res` приложения. В этом приложении также будут использоваться папки ресурсов `menu`, `anim`, `color` и `xml`. В табл. 4.1 приведены краткие описания этих папок (а также папок `animator` и `raw`).

Таблица 4.1. Другие папки, находящиеся в папке res проекта

Вложенная папка	Описание
anim	Папки с именами, начинающимися с anim, содержат XML-файлы с определениями анимаций с переходами, способных изменять прозрачность, размеры и позицию объектов, а также поворачивать их. Мы определим такую анимацию в разделе 4.4.10, а затем воспроизведем ее в разделе 4.7.10, чтобы создать визуальный эффект «дрожания» при неправильном ответе
animator	Папки с именами, начинающимися с animator, содержат XML-файлы с определениями анимаций свойств, изменяющими свойство объекта с течением времени. В Java свойства обычно реализуются в классах переменными экземпляров, имеющими set- и get-методы доступа
color	Папки с именами, начинающимися с color, содержат XML-файлы с определениями списков цветов для различных состояний (например, состояний кнопки — свободное, нажатое, заблокированное и т. д.). Мы используем список цветов состояний для определения разных цветов заблокированных и незаблокированных кнопок в разделе 4.4.8
raw	Папки с именами, начинающимися с raw, содержат файлы ресурсов (например, аудиоролики), которые читаются приложением как поток байтов. Такие ресурсы будут использоваться в главе 6 для воспроизведения звука
menu	Папки с именами, начинающимися с menu, содержат XML-файлы с описаниями содержимого меню. При создании проекта IDE автоматически определяет меню с командой Settings
xml	Папки с именами, начинающимися с xml, содержат XML-файлы, не относящиеся ни к одной другой категории ресурсов. Часто это низкоуровневые файлы данных в формате XML, используемые приложением. В разделе 4.4.11 мы создадим файл XML для настроек приложения, отображаемых классом SettingsActivityFragment

4.3.8. Поддержка разных размеров экранов и разрешений

В разделе 2.5.1 вы узнали, что экраны устройств Android могут иметь разные размеры, разрешения и плотности пикселов (DPI, Dots Per Inch). Также вы узнали, что графика и другие визуальные ресурсы обычно предоставляются в нескольких разрешениях, чтобы система Android могла выбрать оптимальный ресурс для плотности пикселов устройства. Кроме того, в разделе 2.8 было рассказано о том, как определять строковые ресурсы для разных языков и регионов. Android использует схему *уточнения имен* папок ресурсов, чтобы выбирать подходящие изображения на основании плотности пикселов устройства и строки правильных языков — на основании настроек локального контекста и региональных стандартов. Этот механизм также может использоваться для выбора ресурсов из папок, упомянутых в разделе 4.3.7.

Для класса `MainActivity` этого приложения используемый макет определяется квалификаторами размера и ориентации — один вариант для портретной ориентации на телефонах и планшетах, другой — для альбомной ориентации только на планшетах. Соответственно, для `MainActivity` определяются два макета:

- `content_main.xml` — макет по умолчанию, отображающий только `MainActivityFragment`;
- `content_main.xml` (`sw700dp-land`) используется *только* на больших устройствах (например, планшетах), находящихся в альбомной (land) ориентации.

Уточненные имена папок ресурсов имеют формат:

имя-квалификаторы

где часть *квалификаторы* состоит из одного или нескольких квалификаторов, разделенных дефисами (`-`). Существует 19 видов квалификаторов, которые могут добавляться к именам папок ресурсов. Смысл других квалификаторов будет объясняться далее в книге. За полным списком всех квалификаторов вложенных папок `res` и правилами их определения в именах папок обращайтесь по адресу

http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources

4.3.9. Определение размера экрана

В этом приложении меню отображается только в том случае, если приложение выполняется на телефоне или планшете в портретной ориентации (раздел 4.6.5). Чтобы получить эту информацию, мы воспользуемся объектом класса `Configuration` (пакет `android.content.res`), который содержит открытую

переменную экземпляра `orientation`, содержащую `ORIENTATION_PORTRAIT` или `ORIENTATION_LANDSCAPE` для текущей ориентации устройства.

4.3.10. Вывод временных сообщений

Класс `Toast` (пакет `android.widget`) ненадолго выводит сообщение, которое вскоре исчезает. Объекты `Toast` часто используются для отображения второстепенных сообщений об ошибках или информации:

- что викторина будет перезапущена после изменения настроек;
- что необходимо выбрать хотя бы один регион (если пользователь исключил все регионы) — в этом случае приложение назначает Северную Америку регионом по умолчанию для викторины.

4.3.11. Использование обработчика для планирования будущих операций

Если пользователь выбрал правильный вариант, приложение выводит правильный ответ на две секунды, прежде чем вывести следующий флаг. Для этой цели используется объект `Handler` (пакет `android.os`). Метод `postDelayed` класса `Handler` в аргументах получает выполняемую задачу `Runnable` и величину задержки в миллисекундах. По истечении задержки задача `Runnable` выполняется *в том же программном потоке*, который создал `Handler`.

ЗАЩИТА ОТ ОШИБОК 4.1

Операции, взаимодействующие с графическим интерфейсом или изменяющие его, должны выполняться в GUI-потоке, потому что компоненты GUI не являются потоково-безопасными.

4.3.12. Применение анимации к компонентам

Если пользователь выбирает неверный вариант, флаг покачивается, для чего к компоненту `ImageView` применяется анимация `Animation` (пакет `android.view.animation`). Мы используем статический метод `loadAnimation` класса `AnimationUtils` для загрузки анимации из XML-файла, задающего параметры анимации. Также указывается количество повторений анимации (метод `setRepeatCount` класса `Animation`). Выполнение же самой анимации в `ImageView` реализуется путем вызова метода `startAnimation` класса `View` (с аргументом `Animation`) для компонента `ImageView`.

4.3.13. Использование ViewAnimationUtils для создания анимации

Анимация делает приложение более привлекательным. В приложении `Flag Quiz` вскоре после того, как пользователь выберет правильный вариант, приложение воспроизводит анимацию: флаг и кнопки ответов исчезают с экрана, а затем на экране появляется следующий флаг и следующие ответы. В разделе 4.7.9 мы воспользуемся классом `ViewAnimationUtils` для создания объекта `Animator` вызовом метода `createCircularReveal`. После этого задается продолжительность анимации и запускается сама анимация, для чего вызываются методы `setDuration` и `start` объекта `Animator` соответственно. Анимация выглядит как уменьшающееся или увеличивающееся круглое окно, в котором отображается часть интерфейса.

4.3.14. Список цветов состояний

Ресурсный файл со списком цветов состояний определяет цветовой ресурс, который изменяет цвета в зависимости от состояния компонента. Например, можно определить для цвета фона кнопки список цветов состояний, который будет определять разные цвета для кнопки в нажатом, ненажатом, доступном и заблокированном состоянии. Аналогичным образом для компонента `CheckBox` можно задать разные цвета для установленного и снятого состояния.

В нашем приложении при выборе неправильного ответа соответствующая кнопка блокируется, а при правильном ответе блокируются все кнопки. На заблокированной кнопке белый текст читается плохо. Для решения этой проблемы мы определим список цветов состояний, который задает цвет текста кнопки в зависимости от того, находится ли кнопка в разблокированном и заблокированном состоянии (раздел 4.4.8). За дополнительной информацией о списках цветов состояний обращайтесь по адресу

http://developer.android.com/guide/topics/resources/color-list-resource.html

4.3.15. Окна AlertDialog

Объекты `AlertDialog` (пакет `android.app`) используются для вывода сообщений, вариантов действий и их подтверждения. Пока диалоговое окно находится на экране, пользователь не может работать с приложением — такие диалоговые окна называются *модальными*. Как вы вскоре увидите, приложение сначала задает параметры диалогового окна с помощью объекта `AlertDialog.Builder`, а затем использует этот объект для создания `AlertDialog`.

Объекты `AlertDialog` могут отображать кнопки, флажки, переключатели и списки, при помощи которых пользователь реагирует на сообщение в диалоговом

окне. Также в окнах может отображаться графический интерфейс, заданный разработчиком. Стандартный объект `AlertDialog` может содержать до трех кнопок, представляющих следующие действия.

- **Негативное действие** — отменяет операцию диалогового окна (часто эта кнопка помечается надписью `Отмена` или `Нет`). Если диалоговое окно содержит несколько кнопок, эта кнопка находится в крайней левой позиции.
- **Позитивное действие** — подтверждает операцию диалогового окна (часто кнопка помечается надписью `OK` или `Да`). Если диалоговое окно содержит несколько кнопок, эта кнопка находится в крайней правой позиции.
- **Нейтральное действие** — кнопка указывает, что пользователь не хочет ни отменять, ни подтверждать операцию. Например, приложение, которое предлагает пользователю зарегистрироваться для получения доступа к расширенной функциональности, может предоставить нейтральную кнопку `Напомнить позднее`.

В нашем приложении объект `AlertDialog` используется для вывода результатов в конце викторины (раздел 4.7.10). В окне также присутствует кнопка для перезапуска викторины. Мы реализуем интерфейс `DialogInterface.OnClickListener` (пакет `android.content`) для обработки события кнопки. Дополнительную информацию о диалоговых окнах Android можно найти по адресу

http://developer.android.com/guide/topics/ui/dialogs.html

4.3.16. Регистрация сообщений

Когда в программе возникают исключительные ситуации или вы хотите сохранить важные аспекты выполнения своего кода, воспользуйтесь встроенным механизмом *регистрации* сообщений. В Android существует класс `Log` (пакет `android.util`) с набором статических методов, которые представляют различные аспекты сообщений об исключениях. Зарегистрированные сообщения можно просмотреть на вкладке `LogCat` в нижней части IDE, а также воспользоваться утилитой Android *logcat*. Также можно открыть окно Android Device Monitor из Android Studio командой `View` ▸ `Tool Windows` ▸ `Android Monitor`. Дополнительную информацию о регистрации сообщений можно найти на веб-сайте

http://developer.android.com/tools/debugging/debugging-log.html

4.3.17. Запуск другой активности с использованием явного интента

Как упоминалось в разделе 3.7.4, для передачи информации между активностями (внутри одного приложения или разных приложений) в Android

используются *интенты* (intents). Каждая активность может определять *фильтры интентов*, обозначающие *действия*, которые могут выполняться этой активностью. Фильтры интентов определяются в файле AndroidManifest.xml. Собственно, во всех приложениях, которые мы создавали ранее, среда разработки генерировала фильтр интентов для единственной активности приложения; этот фильтр сообщает, что он может реагировать на предопределенное действие с именем `android.intent.action.MAIN`, указывающее, что активность может использоваться для запуска приложения. Активность запускается при помощи интента, обозначающего выполняемое действие, и данных, с которыми это действие выполняется.

Явные и неявные интенты

В этом приложении используется явный интент. Когда приложение работает в портретной ориентации, настройки приложения отображаются в `SettingsActivity` (раздел 4.8) — конкретной активности, предназначенной для управления настройками приложения. В разделе 4.6.6 вы узнаете, как использовать явный интент для запуска конкретной активности в том же приложении.

Android также поддерживает *неявные* (implicit) интенты — разработчик *не указывает* компонент для обработки интента. Например, можно создать интент для отображения содержимого URL-адреса и поручить Android запустить наиболее подходящую активность (браузер) в зависимости от типа данных. Если действие и информация, переданная `startActivity`, могут быть обработаны несколькими активностями, система выводит диалоговое окно, в котором пользователь выбирает активность. Если система не может найти активность для обработки действия, метод `startActivity` инициирует исключение `ActivityNotFoundException`. В общем случае рекомендуется предусмотреть обработку этого исключения в программах. Также можно предотвратить возникновение самого исключения — используйте метод `resolveActivity` класса `Intent` для определения того, существует ли активность для обработки интента. Дополнительную информацию об интентах можно найти по адресу

http://developer.android.com/guide/components/intents-filters.html

4.3.18. Структуры данных Java

В этом приложении использованы различные структуры данных из пакета `java.util`. Приложение динамически загружает имена графических файлов и сохраняет их в массиве `ArrayList<String>`. Метод `shuffle` класса `Collections` случайным образом изменяет порядок имен файлов в массиве `ArrayList<String>` для каждой новой игры (раздел 4.7.7). Второй массив `ArrayList<String>` используется для

хранения имен файлов изображений, соответствующих 10 странам для текущей викторины. Объект `Set<String>` используется для хранения регионов текущей игры. Для обращения к объекту `ArrayList<String>` используется переменная интерфейсного типа `List<String>`.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 4.1

Обращение к объектам коллекций через переменные интерфейсного типа — полезный прием программирования на Java, который позволяет легко изменять структуры данных, не затрагивая при этом остальной код приложения.

4.3.19. Особенности Java SE 7

В Android реализована полноценная поддержка Java SE 7. Полный список возможностей, появившихся в Java SE 7, находится по адресу

http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html

В приложении используются следующие возможности Java SE 7.

- Выведение типов для создания обобщенных экземпляров — если компилятор может вывести тип обобщенного объекта на основании контекста, конструкцию <*mun*> при создании объекта можно заменить на <>. Например, в коде `MainActivityFragment` приложения `Flag Quiz` переменная экземпляра `quizCountriesList` объявляется с типом `List<String>`, и компилятор знает, что коллекция должна содержать `String`. А значит, при создании соответствующего объекта `ArrayList` можно использовать оператор Java SE 7 <>, как в следующей команде; на основании объявления `quizCountriesList` компилятор заключает, что под <> подразумевается `<String>`:

```
quizCountriesList = new ArrayList<>();
```

- Конструкция `try` с ресурсами — вместо того чтобы объявлять ресурс, использовать его в блоке `try` и закрывать в блоке `finally`, можно использовать команду `try` с ресурсами. Ресурс объявляется в круглых скобках блока `try` и используется в блоке `try`. Далее ресурс неявно закрывается, когда программа пытается выйти из блока `try`. Например, в коде `MainActivityFragment` приложения `Flag Quiz` мы используем `InputStream` для чтения байтов изображений флагов и используем их для создания объектов `Drawable` (раздел 4.7.7):

```
try (InputStream stream =
   assets.open(region + "/" + nextImage + ".png")) {
   // Код, который может выдать исключение
}
```

4.3.20. AndroidManifest.xml

Как упоминалось в главе 3, файл `AndroidManifest.xml` создается автоматически при создании приложения. Все активности приложения должны быть объявлены в файле манифеста. Мы покажем, как добавить дополнительные активности в проект. При добавлении в проект активности `SettingsActivity` (раздел 4.4.12) среда разработки также добавит ее в файл манифеста. За полным описанием `AndroidManifest.xml` обращайтесь по адресу

http://developer.android.com/guide/topics/manifest/manifest-intro.html

Другие аспекты файла `AndroidManifest.xml` будут рассмотрены в последующих приложениях.

4.4. Создание проекта, файлов ресурсов и дополнительных классов

В этом разделе мы создадим проект и настроим ресурсы (строки, массивы, цвета и анимации) для приложения `Flag Quiz`. Также будет создан дополнительный класс для второй активности, предназначенной для изменения настроек приложения.

4.4.1. Создание проекта

Создайте новый проект так, как описано в разделе 2.3. Введите следующие значения на шаге `New Project` мастера `Create New Project`:

- `Application Name`: `Flag Quiz`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

На остальных шагах диалогового окна `Create New Project` используйте те же настройки, что и в разделе 2.3, но на этот раз на шаге `Add an activity to Mobile` выберите шаблон `Blank Activity` вместо `Empty Activity` и установите флажок `Use a Fragment`. Оставьте имена по умолчанию для имени активности (`Activity Name`), имени макета (`Layout Name`), заголовка (`Title`) и имени ресурса меню (`Menu Resource Name`). Нажмите кнопку `Finish`, чтобы создать проект. IDE создает различные файлы с кодом Java и ресурсами, включая:

- класс `MainActivity`;
- класс фрагмента `MainActivityFragment` (субкласс `Fragment`), отображаемого `MainActivity`;

- файлы макетов для `MainActivity` и `MainActivityFragment` и
- файл `menu_main.xml`, определяющий меню `MainActivity`.

Также добавьте в проект значок приложения так, как описано в разделе 2.5.2.

Когда проект откроется в Android Studio, IDE выводит файл `content_main.xml` в макетном редакторе. Выберите Nexus 6 в раскрывающемся списке виртуальных устройств (см. рис. 2.8) — как и прежде, это устройство будет взято за основу для построения интерфейса.

4.4.2. Шаблон Blank Activity

`Blank Activity` — шаблон, обеспечивающий обратную совместимость (для Android 2.1 и выше) и использующий возможности библиотеки Android Design Support Library. Этот шаблон может использоваться с фрагментом или без него. Когда вы устанавливаете флажок `Use a Fragment`, IDE создает макеты с именами `activity_main.xml`, `content_main.xml` и `fragment_main.xml`.

activity_main.xml

Макет в файле `activity_main.xml` содержит компонент `CoordinatorLayout` (из пакета `android.support.design.widget` в библиотеке Android Design Support Library). Шаблоны приложений Android Studio, использующие `CoordinatorLayout`, обычно содержат панель приложения, которая определяется как `Toolbar` (пакет `android.support.v7.widget`). Шаблоны явно определяют панель приложения для обеспечения обратной совместимости с ранними версиями Android, в которых панели приложений не поддерживались. Компонент `CoordinatorLayout` также упрощает взаимодействия с вложенными представлениями — например, перемещение части графического интерфейса при появлении представления на экране или восстановление графического интерфейса в исходной позиции при выходе с экрана.

Макет `activity_main.xml` включает (с использованием элемента `<include>` в XML) графический интерфейс, определяемый в `content_main.xml`. Макет по умолчанию также содержит `FloatingActionButton` — круглую кнопку из Android Design Support Library, «плавающую» над графическим интерфейсом приложения. Как правило, кнопка `FloatingActionButton` выделяет важное действие, которое пользователь может выполнить, прикоснувшись к кнопке. Каждое приложение, построенное на основе `Blank Activity`, включает `FloatingActionButton` и другие особенности материального дизайна. Мы начнем использовать `FloatingActionButton` начиная с главы 7.

content_main.xml

Макет `content_main.xml` определяет часть графического интерфейса `MainActivity`, которая отображается между панелью приложения (наверху) и системной панелью (внизу). Когда вы устанавливаете флажок использования фрагмента для шаблона `Blank Activity`, этот файл содержит только элемент `<fragment>` для отображения графического интерфейса `MainActivityFragment`, определенного в `fragment_main.xml`. Если флажок не устанавливается, то файл определяет компонент `RelativeLayout` с `TextView`, а вы самостоятельно определяете в нем графический интерфейс `MainActivity`.

fragment_main.xml

Макет `fragment_main.xml` определяется только в том случае, если вы устанавливаете флажок `Use a Fragment` для шаблона `Blank Activity`. При использовании фрагмента основной графический интерфейс определяется именно здесь.

Подготовка к построению графического интерфейса

Кнопка `FloatingActionButton` в этом приложении не понадобится; откройте макет `activity_main.xml` и удалите ярко-розовую кнопку в правом нижнем углу. Также выделите компонент `CoodinatorLayout` в окне `Component Tree` и задайте свойству `id` макета `coordinatorLayout`. Откройте макет `fragment_main.xml` и удалите компонент `TextView` с текстом `Hello World!` из шаблона приложения.

4.4.3. Настройка поддержки Java SE 7

В этом приложении используются новые возможности Java SE 7. По умолчанию новые проекты Android Studio используют Java SE 6. Чтобы включить поддержку Java SE 7:

1. Щелкните правой кнопкой мыши на папке `app` проекта и выберите команду `Open Module Settings`. Открывается окно `Project Structure`.
2. Убедитесь в том, что в верхней части окна выбрана вкладка `Properties`.
3. Выберите в списках `Source Compatibility` и `Target Compatibility` значение `1.7` и щелкните на кнопке `OK`.

4.4.4. Добавление изображений флагов в проект

Выполните следующие действия, чтобы создать папку `assets` и добавить флаги в проект.

1. Щелкните правой кнопкой мыши на папке app проекта и выберите команду New ▸ Folder ▸ Assets Folder. В открывшемся диалоговом окне Customize the Activity нажмите кнопку Finish.
2. Перейдите к папке на диске, содержащей примеры книги, и скопируйте все содержимое папки images/FlagQuizImages.
3. Щелкните на папке assets в окне Project и вставьте папки, скопированные на предыдущем шаге. В открывшемся диалоговом окне Copy нажмите кнопку OK, чтобы подтвердить копирование папок и их содержимого в проект.

4.4.5. Файл strings.xml и ресурсы форматных строк

В разделе 3.4.5 вы узнали, как создать строковый ресурс при помощи диалогового окна Resources. В этом приложении мы создадим строковые (и многие другие) ресурсы заранее, а затем используем их при построении графического интерфейса и в коде приложения. Для создания строковых ресурсов будет использоваться редактор Translations Editor, представленный в разделе 2.8.

1. В окне Project раскройте узел res/values и откройте файл strings.xml.
2. Щелкните на ссылке Open editor в правом верхнем углу, чтобы открыть редактор Translations Editor.
3. В левом верхнем углу Translations Editor щелкните на кнопке Add Key (+).
4. В открывшемся диалоговом окне введите в поле Key значение number_of_choices, а в поле Default Value — Number of Choices. Нажмите кнопку OK, чтобы создать новый ресурс.
5. Повторите шаг 4 для всех оставшихся строковых ресурсов из табл. 4.2.

СОВЕТ ПО ОФОРМЛЕНИЮ 4.1

В рекомендациях по дизайну приложений Android сказано, что текст, отображаемый в графическом интерфейсе, должен быть коротким, простым и понятным, а важные слова должны находиться в начале. За подробностями о рекомендуемом стиле обращайтесь по адресу http://developer.android.com/design/style/writing.html.

Таблица 4.2. Строковые ресурсы, используемые в приложении Flag Quiz

Имя ресурса	Значение
number_of_choices_description	Display 2, 4, 6 or 8 guess buttons
world_regions	Regions
world_regions_description	Regions to include in the quiz

Имя ресурса	Значение
guess_country	Guess the Country
results	%1$d guesses, %2$.02f%% correct
incorrect_answer	Incorrect!
default_region_message	One region must be selected. Setting North America as the default region
restarting_quiz	Quiz will restart with your new settings
question	Question %1$d of %2$d
reset_quiz	Reset Quiz
image_description	Image of the current flag in the quiz
default_region	North_America

Форматные строки как строковые ресурсы

Ресурсы `results` и `question` содержат *форматные строки*. Если ресурс `String` содержит несколько спецификаторов формата, их необходимо пронумеровать. В ресурсе `results` запись `1$` в `%1$d` обозначает *первое* значение, которое должно заменить спецификатор `%1$d`. Аналогичным образом `2$` в `%2$.02f` обозначает *второе* значение, которое должно заменить спецификатор `%2$.02f`. Символ `d` в первом спецификаторе означает, что подставляемое значение должно форматироваться как целое число, а `f` во втором спецификаторе представляет значение в формате с плавающей точкой. В локализованных версиях `strings.xml` порядок спецификаторов формата `%1$d` и `%2$.02f` может измениться. Первое значение заменяет спецификатор `%1$d`, а второе значение заменяет `%2$.02f` *независимо от их положения в форматной строке*.

4.4.6. arrays.xml

Формально все ресурсы приложения из папки `res/values` могут определяться в одном файле. Тем не менее для удобства управления разными типами ресурсов для каждого типа обычно используется отдельный файл. Например, ресурсы массивов обычно определяются в `arrays.xml`, цветовые ресурсы — в `colors.xml`, строковые — в `strings.xml` и числовые — в `values.xml`. В этом приложении используются три ресурса строковых массивов, определяемые в `arrays.xml`:

- `regions_list` определяет названия регионов, с разделением слов *символами подчеркивания* — эти значения используются для загрузки файлов

изображений из соответствующих папок, а также как выбираемые значения в `SettingsActivityFragment`;

- `regions_list_for_settings` определяет имена регионов с разделением слов *пробелами* — эти значения используются в `SettingsActivityFragment` для вывода названий регионов для пользователя;
- `guesses_list` определяет строки 2, 4, 6 и 8 — эти значения используются в `SettingsActivityFragment` для выбора количества предлагаемых вариантов ответа.

В табл. 4.3 представлены имена и значения элементов этих трех массивов.

Таблица 4.3. Ресурсы строковых массивов, определяемые в arrays.xml

Имя массива ресурса	Значение
regions_list	Africa, Asia, Europe, North_America, Oceania, South_America
regions_list_for_settings	Africa, Asia, Europe, North America, Oceania, South America
guesses_list	2, 4, 6, 8

Чтобы создать файл и настроить ресурсы массивов, выполните следующие действия.

1. В папке `res` проекта щелкните правой кнопкой мыши на папке `values` и выполните команду `New ▸ Values resource file`. На экране появляется диалоговое окно `New Resource File`. Так как щелчок был сделан на папке `values`, диалоговое окно заранее настраивается для добавления файла ресурсов `Values` в папке `values`.
2. Введите в поле `File` значение `arrays.xml` и нажмите `OK`, чтобы завершить создание файла.
3. Android Studio не предоставляет редактор ресурсов для строковых массивов, поэтому для создания ресурса строкового массива придется напрямую редактировать файл XML.

Ресурс строкового массива имеет следующий формат:

```
<string-array name="имя_ресурса">
   <item>значение первого элемента</item>
   <item>значение второго элемента</item>
   ...
</string-array>
```

В листинге 4.1 приведен завершенный файл XML.

Листинг 4.1. Массив arrays.xml определяет ресурсы строковых массивов, используемые в приложении Flag Quiz

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

<string-array name="regions_list">
  <item>Africa</item>
  <item>Asia</item>
  <item>Europe</item>
  <item>North_America</item>
  <item>Oceania</item>
  <item>South_America</item>
</string-array>

   <string-array name="regions_list_for_settings">
      <item>Africa</item>
      <item>Asia</item>
      <item>Europe</item>
      <item>North America</item>
      <item>Oceania</item>
      <item>South America</item>
   </string-array>

   <string-array name="guesses_list">
      <item>2</item>
      <item>4</item>
      <item>6</item>
      <item>8</item>
   </string-array>

</resources>
```

4.4.7. colors.xml

Приложение выводит правильные ответы зеленым шрифтом, а неправильные — красным. Цвета, как и все остальные ресурсы, следует определять в разметке XML, чтобы вы могли легко изменить цвета без модификации исходного кода Java и использовать средства выбора ресурсов Android для определения цветовых ресурсов для разных ситуаций (разные локальные контексты, день/ночь и т. д.). Обычно цвета определяются в файле с именем `colors.xml`, который создается многими шаблонами приложений Android Studio или при определении цветов средствами, описанными в разделе 2.5.7; в противном случае вам придется создать файл самостоятельно.

Шаблон приложения `Blank Activity` уже содержит файл `colors.xml`, в котором определяются ресурсы для основного цвета темы, основного темного и акцентного цвета. Мы добавим цветовые ресурсы для правильного и неправильного ответа, а также изменим акцентный цвет приложения. Для этого

отредактируем XML напрямую (вместо того чтобы использовать `Theme Editor`, как в разделе 3.5).

Откройте файл `colors.xml` (листинг 4.2) из папки `res/values` и добавьте в него строки 6 и 7. Также измените шестнадцатеричное значение цвета `colorAccent` (строка 5) с `#FF4081` (светло-розовый цвет, используемый по умолчанию шаблоном приложения) на `#448AFF` (более светлый оттенок синего по сравнению с `colorPrimary` и `colorPrimaryDark`). Обратите внимание: в редакторе XML рядом с каждым цветом отображается образец цвета.

Листинг 4.2. Файл colors.xml определяет ресурсы цветов приложения

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3    <color name="colorPrimary">#3F51B5</color>
4    <color name="colorPrimaryDark">#303F9F</color>
5    <color name="colorAccent">#448AFF</color>
6    <color name="correct_answer">#00CC00</color>
7    <color name="incorrect_answer">#FF0000</color>
8 </resources>
```

4.4.8. button_text_color.xml

Как упоминалось в разделе 4.3.14, если для цвета кнопки (основного или фонового) предоставлен ресурс цветов состояний, то Android выбирает цвет из списка в зависимости от состояния кнопки. Для таких приложений определяются цвета для заблокированного и доступного состояния. Чтобы создать список цветов состояний, выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `res` проекта и выполните команду `New ▸ Android resource file`. На экране появляется диалоговое окно `New Resource File`.
2. Введите в поле `File` значение `button_text_color.xml`.
3. В раскрывающемся списке `Resource type` выберите вариант `Color`. Значение `Root element` автоматически заменяется на `selector`, а значение `Directory name` автоматически заменяется на `color`.
4. Нажмите кнопку `OK`, чтобы создать файл. Файл `button_text_color.xml` создается в папке `res/color`, которая также будет автоматически создана IDE.
5. Добавьте в файл разметку, приведенную в листинге 4.3.

Элемент `<selector>` (строки 2–10) содержат элементы `<item>`, каждый из которых задает цвет для определенного состояния кнопки. В этом списке цветов состояний в каждом элементе `<item>` задается свойство `android:state_enabled` — для каждого доступного состояния (`true`; строки 3–5) и для каждого

заблокированного состояния (`false`; строки 7–9). Свойство `android:color` (строки 4 и 8) задает цвет для этого состояния.

Листинг 4.3. Файл button_text_color.xml определяет цвет текста на кнопке для доступного и заблокированного состояния

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <selector xmlns:android="http://schemas.android.com/apk/res/android">
 3    <item
 4       android:color="@android:color/primary_text_dark"
 5       android:state_enabled="true"/>
 6
 7    <item
 8       android:color="@android:color/darker_gray"
 9       android:state_enabled="false"/>
10 </selector>
```

4.4.9. Редактирование файла menu_main.xml

При тестовом запуске приложения вы коснулись кнопки (⚙) для обращения к настройкам приложения. Сейчас мы добавим значок в проект и отредактируем файл `menu_main.xml`, чтобы он отображался на панели приложения. Чтобы добавить значок в проект:

1. Запустите Vector Asset Studio командой `File` ▸ `New` ▸ `Vector Asset` — эта программа позволяет добавить в проект любые из рекомендуемых Google значков материального дизайна (*https://www.google.com/design/icons/*). Каждый значок определяется в виде векторного изображения, которое плавно масштабируется для любого размера.
2. Нажмите кнопку `Choose`. Прокрутите содержимое открывшегося диалогового окна, найдите значок (⚙), выделите его и нажмите кнопку `OK`. IDE обновляет значение `Resource name` в соответствии с выбранным значком — при желании имя можно отредактировать. Остальные значения в диалоговом окне оставьте без изменений.
3. Нажмите кнопку `Next`, а затем кнопку `Finish`, чтобы добавить в папку `res/drawable` масштабируемое представление значка — `ic_settings_24dp.xml`.
4. По умолчанию значки, добавляемые в проект таким образом, окрашены в черный цвет; их будет трудно разглядеть на темно-синем фоне панели приложения. Чтобы изменить цвет, откройте файл `ic_settings_24dp.xml` и задайте атрибуту `android:fillColor` элемента `<path>` значение `white`:

```
android:fillColor="@android:color/white"
```

Затем добавьте значок в `menu_main.xml`.

1. Откройте файл menu_main.xml в редакторе — этот файл находится в папке res/menu.
2. Добавьте в элемент <item> следующий атрибут android:icon (предварительное изображение значка отображается на сером поле слева от строки):

```
android:icon="@drawable/ic_settings_24dp"
```

3. Вы можете включить режим, при котором команда меню отображается прямо на панели приложения; в таком случае она называется *действием* (action). По умолчанию действие представляется значком команды меню — если он задан; в противном случае отображается текст команды. Чтобы команда меню отображалась как действие на панели приложения, задайте атрибуту app:showAsAction элемента <item> следующее значение:

```
app:showAsAction="always"
```

В следующей главе вы узнаете, как задать команды меню, отображающиеся на панели действий только при наличии свободного места.

4.4.10. Создание анимации «качающегося» флага

В этом разделе мы создадим анимацию «качающегося флага», используемую в случае неправильного выбора варианта. В разделе 4.7.10 будет показано, как использовать эту анимацию в приложении. Для создания этой анимации выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке res макета и выберите команду New ▸ Android resource file, чтобы открыть диалоговое окно New Resource file.
2. В текстовом поле File name введите имя incorrect_shake.xml.
3. В списке Resource Type выберите вариант Animation. Значение Root element автоматически заменяется на set, а Directory name на anim.
4. Нажмите кнопку OK, чтобы создать файл. Файл немедленно открывается в режиме разметки XML.

Среда разработки не предоставляет редактор для анимаций, поэтому вам придется изменить разметку XML в соответствии с листингом 4.4.

Листинг 4.4. Анимация «качающегося флага» (incorrect_shake.xml), применяемая к флагу в случае неправильного выбора варианта

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <set xmlns:android="http://schemas.android.com/apk/res/android"
```

```
 4    android:interpolator="@android:anim/decelerate_interpolator" >
 5
 6    <translate android:duration="100" android:fromXDelta="0"
 7       android:toXDelta="-5%p" />
 8
 9    <translate android:duration="100" android:fromXDelta="-5%p"
10       android:toXDelta="5%p" android:startOffset="100" />
11
12    <translate android:duration="100" android:fromXDelta="5%p"
13       android:toXDelta="-5%p" android:startOffset="200" />
14 </set>
```

В рассматриваемом примере используются *анимации View* для создания эффекта «качающегося флага». Эта анимация фактически состоит из трех анимаций, объединенных в *набор анимаций* (строки 3–14) — группу, образующих большую анимацию. Наборы анимаций могут включать произвольную комбинацию *анимаций с переходами* — `alpha` (прозрачность), `scale` (изменение размеров), `translate` (перемещение) и `rotate` (поворот). Анимация «качающегося флага» состоит из трех анимаций `translate`. Анимация `translate` перемещает компонент `View` внутри родительского компонента. Android также поддерживает *анимации свойств*, позволяя задавать анимацию любого свойства произвольного объекта.

Первая анимация `translate` (строки 6–7) перемещает компонент `View` из начальной позиции в конечную через заданный период времени. Атрибут `android:fromXDelta` определяет смещение компонента `View` (в начальной позиции анимации), а атрибут `android:toXDelta` — смещение `View` в конечной позиции анимации. Эти атрибуты могут включать:

- абсолютные значения (в пикселах);
- проценты от размера анимированного компонента `View`;
- проценты от размера *родительского* компонента `View`.

Атрибуту `android:fromXDelta` было присвоено абсолютное значение 0. Атрибуту `android:toXDelta` было задано значение `-5%p`, означающее, что компонент `View` должен быть перемещен *влево* (знак «минус») на 5% от ширины родительского компонента (на это указывает буква `p`). Если потребуется выполнить перемещение на величину, равную 5% от ширины компонента `View`, букву `p` не указывайте. Атрибут `android:duration` определяет продолжительность анимации (в миллисекундах). Таким образом, анимация, определенная в строках 6–7, переместит компонент `View` влево на 5% относительно ширины родительского компонента в течение 100 миллисекунд.

Вторая анимация (строки 9–10) продолжается с того места, где была завершена первая. Она перемещает компонент `View` с позиции, заданной смещением `-5%p`, в позицию, заданную смещением `%5p`, в течение 100 миллисекунд.

По умолчанию анимации, включенные в набор анимаций, применяются параллельно, но с помощью атрибута `android:startOffset` можно задать величину задержки (в миллисекундах) перед началом анимации. Задержка может использоваться для последовательного выполнения анимаций. В нашем примере вторая анимация начинается через 100 миллисекунд после завершения первой. Третья анимация (строки 12–13) совпадает со второй, но выполняется в обратном направлении и начинается через 200 миллисекунд после завершения первой анимации.

4.4.11. Определение конфигурации приложения в файле preferences.xml

В этом разделе мы создадим файл `preferences.xml`, используемый классом `SettingsActivityFragment` для отображения настроек приложения. Выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `res` проекта и выберите команду `New` ▸ `Android resource file`, чтобы открыть диалоговое окно `New Resource File`.
2. Введите в поле `File name` значение `preferences.xml`.
3. Выберите в списке `Resource type` вариант `XML`. Значение `Root element` автоматически заменяется на `PreferenceScreen` (экран с перечнем настроек), а значение `Directory name` автоматически заменяется на `color`. Значение в поле `Directory name` автоматически заменяется на `xml`.
4. Нажмите кнопку `OK`, чтобы создать файл. Файл `preferences.xml` создается в папке `xml`, которая также будет автоматически создана IDE.
5. Если файл `res/xml/preferences.xml` не откроется в IDE автоматически, откройте его двойным щелчком.

Теперь в файл нужно добавить два типа настроек, `ListPreference` и `MultiSelectListPreference`. Каждая настройка обладает рядом свойств, описанных в табл. 4.4 (для `ListPreference`) и табл. 4.5 (для `MultiSelectListPreference`). Чтобы добавить настройки и их свойства в файл, необходимо отредактировать разметку XML. Окончательный вид файла XML представлен в листинге 4.5.

Таблица 4.4. Значения свойств ListPreference

Свойство	Значение	Описание
entries	@array/guesses_list	Массив строк, которые будут отображаться в виде списка вариантов

Свойство	Значение	Описание
entryValues	@array/guesses_list	Массив значений, связанных с элементами из свойства Entries. Значение выбранного элемента будет храниться в объекте SharedPreferences
key	pref_numberOfChoices	Имя параметра, хранящегося в SharedPreferences
title	@string/number_of_choices	Заголовок параметра, отображаемый в GUI
summary	@string/number_of_choices_description	Краткое описание параметра, отображаемое под заголовком
persistent	true	Признак сохранения параметра после завершения приложения — если он равен true, класс PreferenceFragment немедленно сохраняет значение параметра при каждом его изменении
defaultValue	4	Элемент из свойства Entries, которое выбирается по умолчанию

Таблица 4.5. Значения свойств MultiSelectListPreference

Свойство	Значение	Описание
entries	@array/regions_list_for_settings	Массив строк, которые будут отображаться в виде списка вариантов
entryValues	@array/regions_list	Массив значений, связанных с элементами из свойства Entries. Значения всех выбранных элементов будут храниться в объекте SharedPreferences
key	pref_regionsToInclude	Имя параметра, хранящегося в SharedPreferences
title	@string/world_regions	Заголовок параметра, отображаемый в GUI
summary	@string/world_regions_description	Краткое описание параметра, отображаемое под заголовком
persistent	true	Признак сохранения параметра после завершения приложения
defaultValue	@array/regions_list	Массив значений по умолчанию этого параметра — в данном случае по умолчанию будут выбраны все элементы

Листинг 4.5. Файл preferences.xml определяет настройки, отображаемые в SettingsActivityFragment

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
   xmlns:android="http://schemas.android.com/apk/res/android">

   <ListPreference
      android:entries="@array/guesses_list"
      android:entryValues="@array/guesses_list"
      android:key="pref_numberOfChoices"
      android:title="@string/number_of_choices"
      android:summary="@string/number_of_choices_description"
      android:persistent="true"
      android:defaultValue="4" />

   <MultiSelectListPreference
      android:entries="@array/regions_list_for_settings"
      android:entryValues="@array/regions_list"
      android:key="pref_regionsToInclude"
      android:title="@string/world_regions"
      android:summary="@string/world_regions_description"
      android:persistent="true"
      android:defaultValue="@array/regions_list" />

</PreferenceScreen>
```

4.4.12. Добавление классов SettingsActivity и SettingsActivityFragment в проект

В этом разделе мы создадим класс `SettingsActivity` (раздел 4.8) и класс `SettingsActivityFragment` (раздел 4.9). Для этого в проект будет добавлен шаблон `Blank Activity`, использующий фрагмент. Чтобы добавить в проект классы `SettingsActivity` и `SettingsActivityFragment` (и их макеты), выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `app` и выберите команду `New` ▸ `Activity` ▸ `Blank Activity`, чтобы открыть диалоговое окно `New Android Activity`.
2. В поле `Activity Name` введите имя `SettingsActivity`. Содержимое полей `Layout Name` и `Title` автоматически обновляется в зависимости от того, какой текст был введен в поле `Activity Name`.
3. Введите в поле `Title` значение `Settings`, чтобы добавить в `strings.xml` новый строковый ресурс, который будет отображаться в панели приложения для активности `SettingsActivity`.
4. Установите флажок `Use a Fragment`, чтобы создать класс `SettingsActivityFragment` с соответствующим макетом.

5. Выберите MainActivity в поле Hierarchical Parent новой активности SettingsActivity (воспользуйтесь кнопкой … справа от раскрывающегося списка). При этом Android Studio генерирует код, который помещает на панель приложения кнопку для возврата к родительской активности (то есть MainActivity).
6. Нажмите кнопку Finish, чтобы создать новые классы и макеты.

Среда разработки создает файлы макетов activity_settings.xml, content_settings.xml и fragment_settings.xml в папке res/layout приложения, а также файлы с программным кодом SettingsActivity.java и SettingsActivityFragment.java в папке пакета Java. Откройте макет activity_settings.xml и удалите кнопку FloatingActionButton, как это было сделано в разделе 4.4.2 для activity_main.xml.

4.5. Построение графического интерфейса

В этом разделе мы построим пользовательский интерфейс приложения Flag Quiz. В двух предыдущих главах вы уже видели, как создать графический интерфейс и настроить свойства компонентов, поэтому в разделах 4.5.1–4.5.4 основное внимание будет уделяться новым возможностям. Многие свойства компонентов, которые необходимо задать, перечисляются в таблицах.

4.5.1. Макет activity_main.xml для устройств в портретной ориентации

В двух предыдущих приложениях мы определили графический интерфейс приложения в файле activity_main.xml. При использовании фрагментов графический интерфейс приложения обычно отображает интерфейсы одного или нескольких фрагментов. В нашем приложении макет MainActivity — activity_main.xml — использует элемент <include> в разметке XML для включения в макет MainActivity интерфейса, определенного в файле content_main.xml.

В свою очередь, макет content_main.xml отображает фрагмент MainActivityFragment, графический интерфейс которого определяется в файле fragment_main.xml. Все три файла макетов были созданы IDE при создании проекта в разделе 4.4.1.

В файле content_main.xml, определяемом IDE, элемент <fragment> является корневым. Во время выполнения графический интерфейс MainActivityFragment заполняет часть экрана, занимаемую элементом <fragment>.

СОВЕТ ПО ОФОРМЛЕНИЮ 4.2

В рекомендациях по дизайну приложений Android дается совет оставлять промежуток в 16dp между контентом приложения и краями всей области экрана; тем не менее многие приложения (например, игры) используют весь экран.

В коде этого приложения используются несколько фрагментов. Чтобы код получения ссылок на эти фрагменты лучше читался, мы изменим свойство `id` этого элемента `<fragment>`.

1. Откройте файл `content_main.xml` на вкладке `Design`.
2. В окне `Component Tree` выберите `fragment` — идентификатор по умолчанию, созданный IDE.
3. В окне свойств задайте свойству `id` значение `quizFragment`.
4. Сохраните файл `content_main.xml`.

4.5.2. Создание макета fragment_main.xml

Обычно в приложении определяется макет для каждого фрагмента, но для фрагмента `SettingsActivityFragment` это не нужно — его графический интерфейс будет автоматически сгенерирован функциональностью, унаследованной от суперкласса `PreferenceFragment`. В этом разделе представлен макет `MainActivityFragment` (`fragment_main.xml`). На рис. 4.8 представлены значения свойства `id` компонентов интерфейса `MainActivityFragment` — задайте их при добавлении компонентов в макет.

Для построения графического интерфейса, показанного на рис. 4.18, мы воспользуемся приемами, описанными в главе 3. Напомним, что для выбора конкретного компонента графического интерфейса часто оказывается проще всего воспользоваться окном `Component Tree`. Мы начнем с базового макета и элементов управления, а затем настроим свойства элементов управления для завершения проекта.

Шаг 1: Замена RelativeLayout на LinearLayout

Как и в макетах `activity_main.xml` двух предшествующих приложений, макет по умолчанию `fragment_main.xml` использует `RelativeLayout`. В этом приложении мы заменим его вертикальным компонентом `LinearLayout`.

1. Откройте файл `fragment_main.xml` и перейдите на вкладку `Text`.
2. В разметке XML замените `RelativeLayout` на `LinearLayout`.
3. Вернитесь на вкладку `Design`.

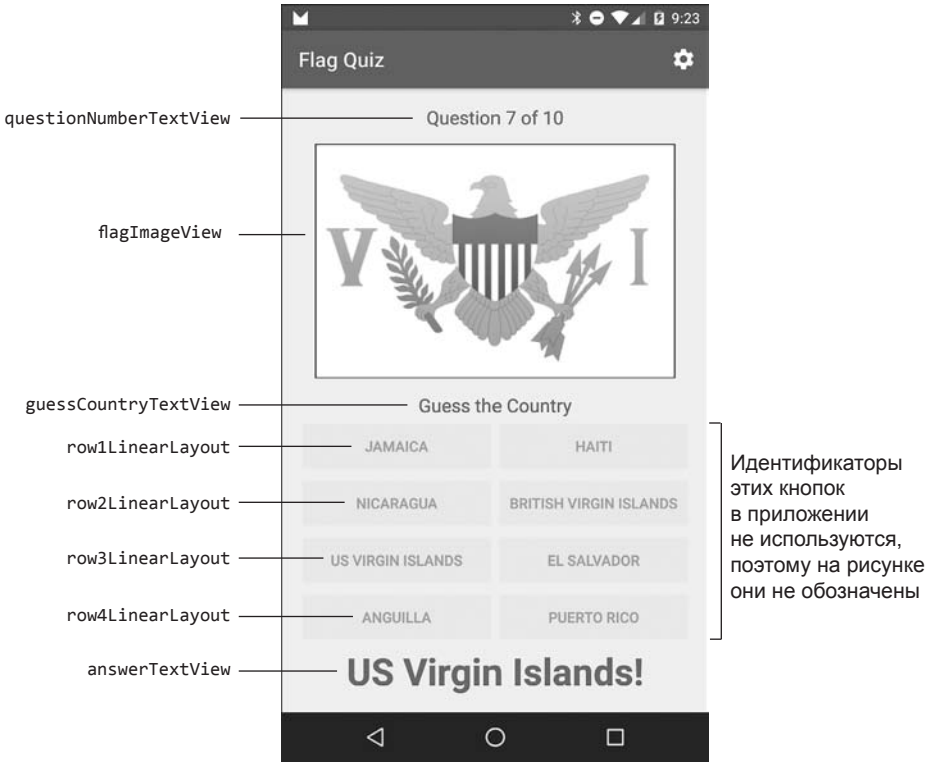


Рис. 4.8. Компоненты Flag Quiz со значениями свойств id

4. В окне `Component Tree` выберите компонент `LinearLayout`.
5. В окне свойств задайте свойству `orientation` компонента `LinearLayout` значение `vertical`.
6. Убедитесь в том, что свойствам `layout:width` и `layout:height` задано значение `match_parent`.
7. Задайте свойству `id` компонента `LinearLayout` значение `quizLinearLayout` для обращения к компоненту из программного кода.

По умолчанию IDE задает свойствам `Padding Left` и `Padding Right` макета предопределенный ресурс метрики с именем `@dimen/activity_horizontal_margin` — он находится в файле `dimens.xml` из папки `res/values` проекта. Значение ресурса равно `16dp`, поэтому левая и правая стороны макета будут снабжены отступом 16dp. Среда разработки создала этот ресурс при создании проекта приложения. Аналогичным образом IDE задает свойствам `Padding Top` и `Padding Bottom` значение `@dimen/activity_vertical_margin` — еще один предопределенный ресурс метрики со значением `16dp`. Следовательно, верхняя и нижняя стороны макета

тоже будут снабжены отступами 16dp. Итак, весь графический интерфейс `MainActivityFragment` будет отделен от остального интерфейса `MainActivity` отступами 16dp.

Шаг 2: Добавление компонента questionNumberTextView в LinearLayout

Перетащите компонент `Medium Text` из раздела `Widgets` палитры на компонент `quizLinearLayout` в окне `Component Tree`. Задайте свойству `id` значение `questionNumberTextView`. В окне свойств задайте следующие значения:

- `layout:gravity center: horizontal` — горизонтальное выравнивание компонента по центру макета;
- `layout:margin: @dimen/spacing` — задайте только для нижнего поля, чтобы добавить зазор 8dp под компонентом. За описанием создания ресурса обращайтесь к разделу 2.5.6;
- `text: @string/question` — чтобы задать это свойство, щелкните на поле свойства `text`, а затем щелкните на кнопке с многоточием (...). На вкладке `Project` диалогового окна `Resources` (рис. 4.9) выберите ресурс `question` и нажмите кнопку `OK`.

Рис. 4.9. Диалоговое окно Resource Chooser — выбор существующего строкового ресурса question

Шаг 3: Добавление компонента flagImageView в LinearLayout

Перетащите компонент `ImageView` из раздела `Widgets` палитры на компонент `quizLinearLayout` в окне `Component Tree`. Задайте свойству `id` значение `flagImageView`. В окне свойств задайте следующие значения.

- `layout:width: match_parent`;
- `layout:height: 0dp` — высота компонента может определяться свойством `layout:weight`;
- `layout:gravity center: both`;
- `layout:margin bottom: @dimen/spacing` — добавляет отступ `8dp` под компонентом;
- `layout:margin left and right: @dimen/activity_horizontal_margin` — добавляет отступ `16dp` у левой и правой стороны компонента, чтобы во время анимации «качания» флаг выводился полностью;
- `layout:weight: 1` — при задании этому свойству `layout:weight` значения 1 (по умолчанию это свойство равно 0 для всех компонентов) компонент `flagImageView` считается более важным по сравнению с другими компонентами `quizLinearLayout`. При построении макета эти компоненты занимают только то вертикальное пространство, которое ему необходимо, а `flagImageView` занимает все остальное вертикальное пространство. IDE рекомендует задать свойству `layout:height` компонента `flagImageView` значение `0dp`, чтобы ускорить построение графического интерфейса во время выполнения;
- `adjustViewBounds: true` — задавая свойству `Adjust View Bounds` компонента `ImageView` значение `true` (устанавливая его флажок), вы тем самым показываете, что `ImageView` сохраняет исходные пропорции изображения;
- `contentDescription: @string/image_description`;
- `scaleType: fitCenter` — означает, что компонент `ImageView` должен масштабировать изображение для заполнения по ширине или высоте без нарушения пропорций исходного изображения. Если ширина изображения меньше ширины `ImageView`, изображение выравнивается по центру по горизонтали. Аналогичным образом, если высота изображения превышает высоту `ImageView`, то изображение выравнивается по центру по вертикали.

СОВЕТ ПО ОФОРМЛЕНИЮ 4.3

Вспомните, что в Android рекомендуется строить приложения так, чтобы каждый компонент GUI мог использоваться с TalkBack. Для компонентов, не имеющих текстовых описаний (например, ImageView), текст включается в свойство contentDescription компонента.

Шаг 4: Добавление компонента guessCountryTextView в LinearLayout

Перетащите компонент `Medium Text` из раздела `Widgets` палитры на компонент `quizLinearLayout` в окне `Component Tree`. Задайте свойству `id` значение `guessCountryTextView`. В окне свойств задайте следующие значения:

- `layout:gravity center: horizontal`;
- `text: @string/guess_country`.

Шаг 5: Добавление кнопок в LinearLayout

В этом приложении кнопки будут добавляться в макет по строкам — каждая строка представляет собой компонент `LinearLayout` с двумя кнопками. Свойства восьми кнопок будут заданы на шаге 7.

1. Перетащите компонент `LinearLayout (Horizontal)` из раздела `Layouts` палитры на компонент `quizLinearLayout` в окне `Component Tree`. Задайте его свойству `id` значение `row1LinearLayout`, а свойству `layout:height` — значение `wrap_content`.
2. Перетащите кнопку из раздела `Widgets` палитры на компонент `row1LinearLayout` в окне `Component Tree`. Задавать его свойство `id` не нужно, так как мы не будем обращаться к кнопкам по идентификаторам в Java-коде приложения.
3. Повторите шаг 2 для другой кнопки в первой строке.
4. Повторите шаги 1–3 еще для трех компонентов `LinearLayout`. Задайте их свойствам `id` значения, приведенные на рис. 4.8, и создайте последние три строки кнопок.

Шаг 6: Добавление компонента answerTextView в LinearLayout

Перетащите компонент `Medium Text` из раздела `Widgets` палитры на компонент `quizLinearLayout` в окне `Component Tree`. Задайте его свойству `id` значение `answerTextView`. В окне свойств задайте следующие значения:

- `layout:gravity`: установите флажок `bottom` и задайте `center` значение `horizontal`;
- `gravity: center_horizontal` — текст `TextView` выравнивается по центру при выводе двух и более строк;
- `textSize: @dimen/answer_size` — размер текста меняется на `36sp`. Создайте ресурс метрики так, как было показано в разделе 2.5.6;
- `textStyle: bold`.

Свойство `text` компонента `TextView` будет задаваться на программном уровне. К этому моменту окно `Component Tree` должно выглядеть так, как показано на рис. 4.10.

Рис. 4.10. Окно Component Tree для fragment_main.xml

Шаг 7: Задание свойств кнопок

После завершения шага 6 задайте свойства кнопок в соответствии с табл. 4.6 — вы можете выделить все восемь кнопок в окне `Component Tree`, а затем задать их свойства для всех кнопок одновременно.

- Задайте свойству `layout:width` значение `0dp`, а свойству `layout:weight` значение 1, чтобы горизонтальное пространство равномерно распределялось между кнопками одного компонента `LinearLayout`.
- Задайте свойству `layout:height` каждой кнопки значение `match_parent`, чтобы высота кнопки соответствовала высоте `LinearLayout`.
- Задайте свойству `lines` каждой кнопки значение 2, чтобы все кнопки имели одинаковую высоту для названий стран, занимающих разное число строк, — если текст на кнопке окажется слишком длинным, то символы, не помещающиеся в двух строках, просто отсекаются.
- Задайте свойству `style` значение `@android:style/Widget.Material.Button.Colored`, чтобы цвет кнопок выбирался в зависимости от цветов темы приложения. В качестве цвета кнопок будет использоваться акцентный цвет приложения, который мы задали в разделе 4.4.7. Чтобы задать это свойство, щелкните на кнопке с многоточием (…), чтобы открыть диалоговое окно `Resources`, выберите `Widget.Material.Button.Colored` на вкладке `System` и нажмите кнопку `OK`.

- Задайте свойству `textColor` список цветов состояний `@color/button_text_color`, определенный в разделе 4.4.8, чтобы цвет текста изменялся в зависимости от состояния кнопки (доступное/заблокированное).

Таблица 4.6. Значения свойств компонентов Button в файле fragment_main.xml

Компонент GUI	Свойство	Значение
Button	Параметры группы layout	
	layout:width	0dp
	layout:height	match_parent
	layout:weight	1
	Другие свойства	
	lines	2
	textColor	@color/button_text_color
	style	@android:style/Widget.Material.Button.Colored

4.5.3. Панель инструментов макетного редактора

Графический интерфейс `MainActivityFragment` завершен. На панели инструментов макетного редактора (табл. 4.7) присутствуют различные кнопки, которые позволяют просмотреть макет для других размеров экрана и ориентаций устройства. В частности, можно просмотреть миниатюры для многих размеров и ориентаций экрана. Для этого откройте `content_main.xml`, щелкните на списке виртуального устройства в верхней части макетного редактора и выберите вариант `Preview All Screen Sizes`. На рис. 4.11 представлены некоторые кнопки на панели инструментов макетного редактора.

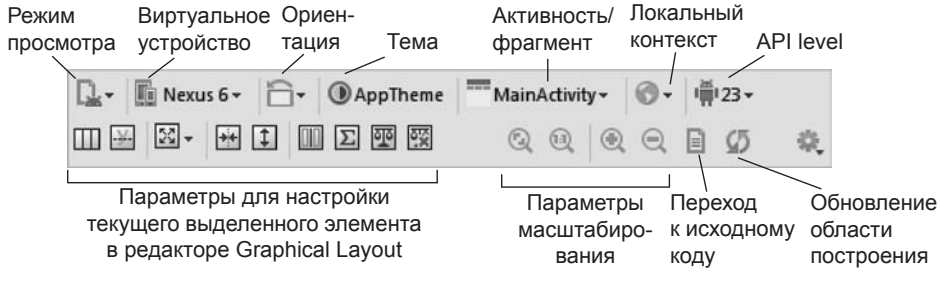


Рис. 4.11. Параметры конфигурации просмотра

Таблица 4.7. Объяснение параметров конфигурации просмотра

Параметр	Описание
Режим просмотра	Просмотр вариантов по одному или одновременный просмотр для разных размеров экрана
Виртуальное устройство	Система Android работает на множестве разных устройств, поэтому макетный редактор включает множество готовых конфигураций для разных размеров и разрешений экрана. В этой книге используются готовые конфигурации для Nexus 6 и Nexus 9 (в зависимости от приложения). На рис. 4.11 выбрано устройство Nexus 6
Ориентация	Переключение области построения интерфейса между портретной и альбомной ориентацией
Тема	Используется для выбора темы графического интерфейса
Активность/фрагмент	Класс Activity или Fragment для создаваемого графического интерфейса
Локальный контекст	Для интернационализированных приложений (раздел 2.8) позволяет выбрать конкретную локализацию — например, чтобы вы могли увидеть, как ваше приложение выглядит со строками другого языка
API level	Целевой уровень API. Как правило, в каждом новом уровне появляются новые средства графического интерфейса. В окне макетного редактора отображаются только те возможности, которые доступны для выбранного уровня API

4.5.4. Макет content_main.xml для планшетов в альбомной ориентации

Как упоминалось ранее, макет `content_main.xml` для `MainActivity` по умолчанию отображает графический интерфейс `MainActivityFragment`. Теперь мы определим макет `MainActivity` для планшетных устройств в альбомной ориентации, в которой оба фрагмента `SettingsActivityFragment` и `MainActivityFragment` должны отображаться одновременно. Для этого будет создан второй макет `content_main.xml`, который будет использоваться системой Android только на подходящих устройствах.

Создание файла content_main.xml для планшетов в альбомной ориентации

Чтобы создать макет, выполните следующие действия:

1. Щелкните правой кнопкой мыши на папке `res/layout` проекта и выберите команду `New ▸ Layout`.

2. Введите в поле File name диалогового окна New Resource File значение content_main.xml.
3. Убедитесь в том, что в поле Root element выбрано значение LinearLayout.
4. В списке Available qualifiers выберите квалификатор Smallest Screen Width и щелкните на кнопке >>, чтобы добавить квалификатор в список Chosen Qualifiers. Присвойте ему значение 700 — макет предназначен для экранов шириной не менее 700 пикселов.
5. В списке Available qualifiers выберите квалификатор Orientation и щелкните на кнопке >>, чтобы добавить квалификатор в список Chosen Qualifiers. Присвойте ему значение Landscape.
6. Нажмите кнопку OK.

Android Studio создает новый файл content_main.xml, который сохраняется в папке с именем res/layout-sw700dp-land. Этот макет будет использоваться только на устройствах с минимальной шириной экрана (sw, «screen width») 700dp и только в том случае, если устройство находится в альбомной (land) ориентации. Android использует квалификаторы sw и land для выбора подходящих ресурсов во время выполнения.

В окне Project не отображаются разные папки layout и layout-sw700dp-land, которые вы увидите при просмотре папок на диске. Вместо этого оба макета объединяются в один узел content_main.xml (2) в папке res/layout окна Project — суффикс (2) означает, что узел содержит два макета. Раскрыв этот узел, вы увидите два файла:

- content_main.xml и
- content_main.xml (sw700dp-land).

Макет без квалификаторов в круглых скобках содержит макет по умолчанию. Версия с квалификаторами используется только в том случае, если она соответствует текущей конфигурации. После создания файла Android Studio открывает макет в макетном редакторе. В режиме Design макет представляется в альбомной ориентации.

Создание графического интерфейса для планшетного макета

Перейдем к построению графического интерфейса для планшетного макета.

1. Выделите компонент LinearLayout (vertical) в окне Component Tree и задайте свойству orientation значение horizontal.
2. Щелкните на варианте <fragment> в разделе Custom палитры. В диалоговом окне Fragments выберите фрагмент SettingsActivityFragment и нажмите

кнопку OK. Затем щелкните на узле LinearLayout в окне Component Tree; <fragment> добавляется в макет. Задайте свойству id фрагмента значение settingsActivityFragment.

3. Повторите предыдущий шаг, но на этот раз выберите MainActivityFragment. Также задайте свойству id этого фрагмента значение quizFragment.
4. Выделите узел settingsActivityFragment в окне Component Tree. Задайте layout:width значение 0dp, layout:height — значение match_parent и layout:weight — значение 1.
5. Выберите узел quizFragment в окне Component Tree. Задайте layout:width значение 0dp, layout:height — значение match_parent и layout:weight — значение 2. Свойство layout:weight у MainActivityFragment равно 2, у SettingsActivityFragment оно равно 1; сумма весов равна 3, поэтому MainActivityFragment будет занимать две трети горизонтального пространства макета.
6. Перейдите на вкладку Text и добавьте следующие две строки в открывающий тег LinearLayout, чтобы верхний край выводился под панелью приложения, а не позади нее:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
app:layout_behavior="@string/appbar_scrolling_view_behavior"
```

Выбор фрагмента для предварительного просмотра в режиме Design макетного конструктора

В режиме Design макетного редактора может выводиться предварительное изображение фрагментов(-а), отображаемых в макете. Если включаемый макет не указан, то макетный редактор выводит сообщение "Rendering Problems". Чтобы задать фрагмент для предварительного просмотра, щелкните правой кнопкой мыши на фрагменте (в режиме Design или в окне Component Tree) и выберите команду Choose Preview Layout…. В диалоговом окне Resources выберите имя макета фрагмента.

4.6. Класс MainActivity

Класс MainActivity (разделы 4.6.1–4.6.7) управляет фрагментом MainActivityFragment при выполнении приложения в портретной ориентации и фрагментами SettingsActivityFragment и MainActivityFragment — когда приложение выполняется на планшете в альбомной ориентации.

4.6.1. Команды package, import и переменные экземпляров

В листинге 4.6 приведены команды `package` и `import`, а также поля класса `MainActivity`. В строках 6–19 импортируются различные классы и интерфейсы Java и Android, используемые приложением. Мы выделили новые команды `import`; соответствующие классы и интерфейсы рассматриваются в разделе 4.3, а также о них упоминается в разделах 4.6.2–4.6.7.

Листинг 4.6. Команда package, команды import и переменные экземпляров класса MainActivity

```
 1 // MainActivity.java
 2 // Управляет фрагментом MainActivityFragment на телефоне и фрагментами
 3 // MainActivityFragment и SettingsActivityFragment на планшете
 4 package com.deitel.flagquiz;
 5
 6 import android.content.Intent;
 7 import android.content.SharedPreferences;
 8 import android.content.SharedPreferences.OnSharedPreferenceChangeListener;
 9 import android.content.pm.ActivityInfo;
10 import android.content.res.Configuration;
11 import android.os.Bundle;
12 import android.preference.PreferenceManager;
13 import android.support.v7.app.AppCompatActivity;
14 import android.support.v7.widget.Toolbar;
15 import android.view.Menu;
16 import android.view.MenuItem;
17 import android.widget.Toast;
18
19 import java.util.Set;
20
```

4.6.2. Поля

В листинге 4.7 приведены поля класса `MainActivity`. В строках 23–24 определяются константы для ключей параметров, созданных в разделе 4.4.11. Ключи будут использоваться для обращения к соответствующим значениям параметров. Логическая переменная `phoneDevice` (строка 26) указывает, выполняется ли приложение на телефоне, — и если выполняется, то приложение разрешает только портретную ориентацию. Логическая переменная `preferencesChanged` (строка 30) указывает, изменились ли настройки приложения, — и если изменились, то метод жизненного цикла `onStart` класса `MainActivity` (раздел 4.6.4) вызывает методы `updateGuessRows` (раздел 4.7.4) и `updateRegions` (раздел 4.7.5) класса `MainActivityFragment` для изменения конфигурации викторины на основании новых настроек. Изначально этой переменной присваивается значение `true`, чтобы при первом запуске использовалась конфигурация с параметрами по умолчанию.

Листинг 4.7. Объявление и поля MainActivity

```
21 public class MainActivity extends Activity {
22    // Ключи для чтения данных из SharedPreferences
23    public static final String CHOICES = "pref_numberOfChoices";
24    public static final String REGIONS = "pref_regionsToInclude";
25
26    private boolean phoneDevice = true; // Включение портретного режима
27    private boolean preferencesChanged = true; // Настройки изменились?
28
```

4.6.3. Переопределение метода onCreate

В листинге 4.8 приведен переопределяемый метод `onCreate` класса `Activity` — мы удалили предопределенный обработчик события кнопки `FloatingActionButton`, который не используется в приложении. В строке 33 вызывается метод `setContentView` для назначения графического интерфейса `MainActivity`. Напомним, что `activity_main.xml` встраивает в свой макет содержимое файла `content_main.xml`, а приложение содержит две версии этого файла. При заполнении `activity_main.xml` Android встраивает файл `content_main.xml` из папки `res/layout`, если только приложение не выполняется на устройстве, имеющем ширину не менее 700 пикселов в альбомной ориентации, — тогда Android использует версию из папки `res/layout-sw700dp-land`. Строки 34–35 были сгенерированы IDE для назначения объекта `Toolbar`, определенного в макете `MainActivity`, как *панели приложения* (ранее называвшейся панелью действия), — еще один пример отображения панели приложения с обеспечением обратной совместимости.

Листинг 4.8. Переопределяемый в MainActivity метод onCreate

```
29    // Настройка MainActivity
30    @Override
31    protected void onCreate(Bundle savedInstanceState) {
32       super.onCreate(savedInstanceState);
33       setContentView(R.layout.activity_main);
34       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
35       setSupportActionBar(toolbar);
36
37       // Задание значений по умолчанию в файле SharedPreferences
38       PreferenceManager.setDefaultValues(this, R.xml.preferences, false);
39
40       // Регистрация слушателя для изменений SharedPreferences
41       PreferenceManager.getDefaultSharedPreferences(this).
42          registerOnSharedPreferenceChangeListener(
43             preferencesChangeListener);
44
45       // Определение размера экрана
46       int screenSize = getResources().getConfiguration().screenLayout &
47          Configuration.SCREENLAYOUT_SIZE_MASK;
48
49       // Для планшетного устройства phoneDevice присваивается false
```

```
50       if (screenSize == Configuration.SCREENLAYOUT_SIZE_LARGE ||
51          screenSize == Configuration.SCREENLAYOUT_SIZE_XLARGE)
52          phoneDevice = false; // Не соответствует размерам телефона
53
54       // На телефоне разрешена только портретная ориентация
55       if (phoneDevice)
56          setRequestedOrientation(
57             ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
58    }
59
```

Определение значений по умолчанию и регистрация слушателя изменений

При установке и первом запуске приложения в строке 38 назначаются параметры конфигурации приложения по умолчанию, для чего вызывается метод `setDefaultValues` класса `PreferenceManager` — он создает и инициализирует файл `SharedPreferences` приложения, используя значения по умолчанию, заданные в `preferences.xml`. Метод получает три аргумента.

- Объект `Context` (пакет `android.content`) предоставляет доступ к информации об окружении, в котором выполняется приложение, и позволяет использовать различные устройства Android — в данном случае это активность (`this`), для которой задаются настройки по умолчанию.
- Идентификатор ресурса для XML-файла настроек (`R.xml.preferences`), создаваемого в разделе 4.4.11.
- Логический признак, определяющий, должны ли значения по умолчанию сбрасываться при каждом вызове метода `setDefaultValues`, — значение `false` указывает, что значения настроек по умолчанию должны задаваться только при первом вызове этого метода.

Каждый раз, когда пользователь изменяет настройки приложения, объект `MainActivity` должен вызвать метод `updateGuessRows` или `updateRegions` класса `MainActivityFragment` (в зависимости от того, какой параметр был изменен). `MainActivity` регистрирует слушателя `OnSharedPreferenceChangedListener` (строки 41–43), чтобы получать оповещения обо всех изменениях настроек. Метод `getDefaultSharedPreferences` класса `PreferenceManager` возвращает ссылку на объект `SharedPreferences`, представляющий параметры конфигурации приложения, а метод `registerOnSharedPreferenceChangeListener` класса `SharedPreferences` регистрирует слушателя, определяемого в разделе 4.6.7.

Настройка портретной ориентации на телефонах

Строки 46–52 определяют, на каком устройстве выполняется приложение: на планшете или на телефоне. Унаследованный метод `getResources` возвращает объект `Resources` приложения (пакет `android.content.res`), который

может использоваться для обращения к ресурсам приложения и получения информации о среде выполнения. Метод `getConfiguration` возвращает объект `Configuration` (пакет `android.content.res`) с открытой (`public`) переменной экземпляра `screenLayout`, по которой можно определить категорию размера экрана устройства. Для этого значение `screenLayout` объединяется с маской `Configuration.SCREENLAYOUT_SIZE_MASK` при помощи поразрядного оператора И (`&`). Затем результат сравнивается с константами `SCREENLAYOUT_SIZE_LARGE` и `SCREENLAYOUT_SIZE_XLARGE` класса `Configuration` (строки 50–51). Если одна из проверок даст положительный результат, значит, приложение выполняется на планшетном устройстве. Наконец, если устройство является телефоном, в строках 56–57 вызывается унаследованный от `Activity` метод `setRequestedOrientation`, который заставляет приложение отображать `MainActivity` только в портретной ориентации.

4.6.4. Переопределение метода onStart

Переопределяемый метод жизненного цикла активности `onStart` (листинг 4.9) вызывается в следующих двух ситуациях.

- При первом запуске приложения метод `onStart` вызывается после `onCreate`. В этом случае вызов `onStart` гарантирует, что приложение будет правильно инициализировано в состоянии по умолчанию при установке и первом запуске или в соответствии с обновленной конфигурацией пользователя при последующих запусках.
- Если приложение выполняется в портретной ориентации, а пользователь открывает `SettingsActivity`, активность `MainActivity` приостанавливается на время отображения `SettingsActivity`. Когда пользователь возвращается к `MainActivity`, снова вызывается метод `onStart`. На этот раз вызов обеспечивает необходимое изменение конфигурации, если пользователь внес изменения в настройки.

В обоих случаях, если переменная `preferencesChanged` равна `true`, `onStart` вызывает методы `updateGuessRows` (раздел 4.7.5) и `updateRegions` (раздел 4.7.5) класса `MainActivityFragment` для изменения конфигурации. Чтобы получить ссылку на объект `MainActivityFragment` для вызова его методов, в строках 68–70 используется унаследованный от `AppCompatActivity` метод `getSupportFragmentManager` для получения объекта `FragmentManager`, после чего вызывается его метод `findFragmentById`. Затем в строках 71–74 вызываются методы `updateGuessRows` и `updateRegions` класса `MainActivityFragment`, при этом в аргументах передается объект `SharedPreferences` приложения, чтобы эти методы могли загрузить текущую конфигурацию. Строка 75 сбрасывает состояние игры, а строка 76 возвращает `preferencesChanged` значение `false`.

Листинг 4.9. Переопределенный метод onStart класса MainActivity

```
60    // Вызывается после завершения выполнения onCreate
61    @Override
62    protected void onStart() {
63       super.onStart();
64
65       if (preferencesChanged) {
66          // После задания настроек по умолчанию инициализировать
67          // MainActivityFragment и запустить викторину
68          MainActivityFragment quizFragment = (MainActivityFragment)
69             getSupportFragmentManager().findFragmentById(
70                R.id.quizFragment);
71          quizFragment.updateGuessRows(
72             PreferenceManager.getDefaultSharedPreferences(this));
73          quizFragment.updateRegions(
74             PreferenceManager.getDefaultSharedPreferences(this));
75          quizFragment.resetQuiz();
76          preferencesChanged = false;
77       }
78    }
79
```

4.6.5. Переопределение метода onCreateOptionsMenu

Метод `onCreateOptionsMenu` (листинг 4.10) вызывается для инициализации стандартного меню активности — этот метод вместе с методом `onOptionsItemSelected` автоматически генерируется шаблоном Android Studio `Blank Activity`. Система передает объект `Menu`, в котором должны отображаться команды. В нашем приложении меню должно отображаться только при запуске приложения в портретной ориентации. В строке 84 объект `Resources` класса `Activity` (возвращаемый унаследованным методом `getResources`) используется для получения объекта `Configuration` (возвращаемого методом `getConfiguration`), представляющего текущую конфигурацию устройства. Открытая переменная экземпляра `orientation` содержит признак `Configuration.ORIENTATION_PORTRAIT` или `Configuration.ORIENTATION_LANDSCAPE`. Если устройство находится в портретной ориентации (строка 87), строка 89 создает меню на базе файла `menu_main.xml` — ресурса меню по умолчанию, созданного средой разработки при создании проекта. Метод `getMenuInflater`, унаследованный от `Activity`, возвращает объект `MenuInflater`, для которого вызывается метод `inflate` с двумя аргументами — идентификатором ресурса меню, используемого для заполнения меню, и объектом `Menu`, в который будут помещены команды меню. Если метод `onCreateOptionsMenu` возвращает `true`, это означает, что меню должно отображаться на экране.

Листинг 4.10. Переопределенный метод onCreateOptionsMenu класса MainActivity

```
80    // Меню отображается на телефоне или планшете в портретной ориентации
81    @Override
82    public boolean onCreateOptionsMenu(Menu menu) {
```

```
83        // Получение текущей ориентации устройства
84        int orientation = getResources().getConfiguration().orientation;
85
86        // Отображение меню приложения только в портретной ориентации
87          if (orientation == Configuration.ORIENTATION_PORTRAIT) {
88             // Заполнение меню
89             getMenuInflater().inflate(R.menu.menu_main, menu);
90             return true;
91        }
92        else
93             return false;
94     }
95
```

4.6.6. Переопределение метода onOptionsItemSelected

Метод `onOptionsItemSelected` (листинг 4.11) вызывается при выборе команды меню. В нашем приложении меню по умолчанию, сгенерированное средой разработки при создании проекта, состоит из единственной команды `Settings`; следовательно, если этот метод вызывается, пользователь выбрал команду `Settings`. Строка 99 создает явный интент для запуска `SettingsActivity`. Используемый конструктор `Intent` получает объект контекста, из которого будет запускаться активность, и класс, представляющий запускаемую активность (`SettingsActivity.class`). Затем этот интент передается унаследованному методу `startActivity` для запуска активности (строка 100).

Листинг 4.11. Переопределенный метод onOptionsItemSelected класса MainActivity

```
 96    // Отображает SettingsActivity при запуске на телефоне
 97    @Override
 98    public boolean onOptionsItemSelected(MenuItem item) {
 99       Intent preferencesIntent = new Intent(this, SettingsActivity.class);
100       startActivity(preferencesIntent);
101       return super.onOptionsItemSelected(item);
102    }
103
```

4.6.7. Анонимный внутренний класс, реализующий интерфейс OnSharedPreferenceChangeListener

Объект `preferenceChangeListener` (листинг 4.12) является объектом анонимного внутреннего класса, реализующего интерфейс `OnSharedPreferenceChangeListener`. Этот объект был зарегистрирован в методе `onCreate` для прослушивания изменений в конфигурации `SharedPreferences` приложения. Когда происходит

изменение, метод onSharedPreferenceChanged задает preferencesChanged значение true (строка 111), после чего получает ссылку на MainActivityFragment (строки 113–115) для сброса викторины с новой конфигурацией. Если изменился параметр CHOICES, то в строках 118–119 вызываются методы updateGuessRows и resetQuiz класса MainActivityFragment.

Листинг 4.12. Анонимный внутренний класс, реализующий интерфейс onSharedPreferenceChangeListener

```
104    // Слушатель изменений в конфигурации SharedPreferences приложения
105    private OnSharedPreferenceChangeListener preferencesChangeListener =
106       new OnSharedPreferenceChangeListener() {
107          // Вызывается при изменении настроек приложения
108          @Override
109          public void onSharedPreferenceChanged(
110             SharedPreferences sharedPreferences, String key) {
111             preferencesChanged = true; // Пользователь изменил настройки
112
113             MainActivityFragment quizFragment = (MainActivityFragment)
114                getSupportFragmentManager().findFragmentById(
115                   R.id.quizFragment);
116
117             if (key.equals(CHOICES)) { // Изменилось число вариантов
118                quizFragment.updateGuessRows(sharedPreferences);
119                quizFragment.resetQuiz();
120             }
121             else if (key.equals(REGIONS)) { // Изменились регионы
122                Set<String> regions =
123                   sharedPreferences.getStringSet(REGIONS, null);
124
125                if (regions != null && regions.size() > 0) {
126                   quizFragment.updateRegions(sharedPreferences);
127                   quizFragment.resetQuiz();
128                }
129                else {
130                   // Хотя бы один регион - по умолчанию Северная Америка
131                   SharedPreferences.Editor editor =
132                      sharedPreferences.edit();
133                   regions.add(getString(R.string.default_region));
134                   editor.putStringSet(REGIONS, regions);
135                   editor.apply();
136
137                   Toast.makeText(MainActivity.this,
138                      R.string.default_region_message,
139                      Toast.LENGTH_SHORT).show();
140                }
141             }
142
143             Toast.makeText(MainActivity.this,
144                R.string.restarting_quiz,
145                Toast.LENGTH_SHORT).show();
146          }
147    };
148 }
```

Если изменился параметр `REGIONS`, то строки 122–123 получают коллекцию `Set<String>` с включенными регионами. Метод `getStringSet` класса `SharedPreferences` возвращает `Set<String>` для заданного ключа. В викторину должен быть включен хотя бы один регион, поэтому если множество `Set<String>` не пусто, строки 126–127 вызывают методы `updateRegions` и `resetQuiz` класса `MainActivityFragment`.

В противном случае в строках 131–135 в параметре `REGIONS` выбирается Северная Америка как регион по умолчанию. Для получения имени региона по умолчанию строка 133 вызывает унаследованный метод `getString` класса `Activity`, который возвращает ресурс `String` для заданного идентификатора ресурса (`R.string.default_region`).

Чтобы изменить содержимое объекта `SharedPreferences`, сначала мы вызываем его метод `edit` для получения объекта `SharedPreferences.Editor` (строки 131–132), который позволяет добавлять пары «ключ—значение», удалять пары «ключ—значение» и изменить значение, связанное с заданным ключом, в файле `SharedPreferences`. Строка 134 вызывает метод `putStringSet` класса `SharedPreferences.Editor` для сохранения содержимого `regions` (`Set<String>`). В строке 135 изменения закрепляются (сохраняются) вызовом метода `apply` класса `SharedPreferences.Editor`, который немедленно вносит изменения в представление `SharedPreferences` в памяти и асинхронно записывает изменения в файл в фоновом режиме. Также существует метод `commit`, который записывает изменения в файл синхронно (то есть немедленно).

В строках 137–139 выводится уведомление `Toast`, которое сообщает о задании региона по умолчанию. В аргументах метода `makeText` класса `Toast` передается контекст, в котором отображаются `Toast`, выводимое сообщение и продолжительность вывода. Временное окно отображается на экране вызовом метода `show` класса `Toast`. Независимо от того, какой параметр изменился, в строках 143–145 выводится предупреждение о том, что викторина будет перезапущена с новыми параметрами. Временное окно `Toast`, появляющееся при изменении конфигурации приложения пользователем, показано на рис. 4.12.

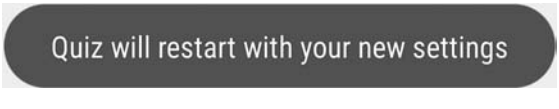


Рис. 4.12. Временное окно, появляющееся при изменении конфигурации

4.7. Класс MainActivityFragment

Класс `MainActivityFragment` (листинги 4.13–4.23) — субкласс класса `Fragment` из библиотеки Android Support Library (пакет `android.support.v4.app`) — строит графический интерфейс игры и реализует ее логику.

4.7.1. Команды package и import

В листинге 4.13 приведены команды `package` и `import` класса `MainActivityFragment`. В строках 5–36 импортируются различные классы и интерфейсы Java и Android, используемые приложением. Мы выделили важнейшие команды `import`; соответствующие классы и интерфейсы рассматриваются в разделе 4.3, а также там, где они упоминаются (в разделах 4.7.2–4.7.11).

Листинг 4.13. Команда package и команды import класса MainActivityFragment

```
 1 // MainActivityFragment.java
 2 // Класс содержит логику приложения Flag Quiz
 3 package com.deitel.flagquiz;
 4
 5 import java.io.IOException;
 6 import java.io.InputStream;
 7 import java.security.SecureRandom;
 8 import java.util.ArrayList;
 9 import java.util.Collections;
10 import java.util.List;
11 import java.util.Set;
12
13 import android.animation.Animator;
14 import android.animation.AnimatorListenerAdapter;
15 import android.app.AlertDialog;
16 import android.app.Dialog;
17 import android.content.DialogInterface;
18 import android.content.SharedPreferences;
19 import android.content.res.AssetManager;
20 import android.graphics.drawable.Drawable;
21 import android.os.Bundle;
22 import android.support.v4.app.DialogFragment;
23 import android.support.v4.app.Fragment;
24 import android.os.Handler;
25 import android.util.Log;
26 import android.view.LayoutInflater;
27 import android.view.View;
28 import android.view.View.OnClickListener;
29 import android.view.ViewAnimationUtils;
30 import android.view.ViewGroup;
31 import android.view.animation.Animation;
32 import android.view.animation.AnimationUtils;
33 import android.widget.Button;
34 import android.widget.ImageView;
35 import android.widget.LinearLayout;
36 import android.widget.TextView;
37
```

4.7.2. Поля

В листинге 4.14 перечислены статические переменные и переменные экземпляров класса `MainActivityFragment`. Константа `TAG` (строка 40) используется

при регистрации ошибок средствами класса `Log` (листинг 4.19), чтобы эти сообщения об ошибках отличались от других сообщений, записываемых в журнал устройства. Константа `FLAGS_IN_QUIZ` (строка 42) представляет количество флагов в викторине.

Листинг 4.14. Поля MainActivityFragment

```
38 public class MainActivityFragment extends Fragment {
39    // Строка, используемая при регистрации сообщений об ошибках
40    private static final String TAG = "FlagQuiz Activity";
41
42    private static final int FLAGS_IN_QUIZ = 10;
43
44    private List<String> fileNameList; // Имена файлов с флагами
45    private List<String> quizCountriesList; // Страны текущей викторины
46    private Set<String> regionsSet; // Регионы текущей викторины
47    private String correctAnswer; // Правильная страна для текущего флага
48    private int totalGuesses; // Количество попыток
49    private int correctAnswers; // Количество правильных ответов
50    private int guessRows; // Количество строк с кнопками вариантов
51    private SecureRandom random; // Генератор случайных чисел
52    private Handler handler; // Для задержки загрузки следующего флага
53    private Animation shakeAnimation; // Анимация неправильного ответа
54
55    private LinearLayout quizLinearLayout; // Макет с викториной
56    private TextView questionNumberTextView; // Номер текущего вопроса
57    private ImageView flagImageView; // Для вывода флага
58    private LinearLayout[] guessLinearLayouts; // Строки с кнопками
59    private TextView answerTextView; // Для правильного ответа
60
```

ПЕРЕДОВОЙ ОПЫТ ПРОГРАММИРОВАНИЯ 4.1

Ради удобства чтения и изменения кода используйте строковые константы для представления литералов с именами файлов. Эти строки не нуждаются в локализации и поэтому не определяются в strings.xml.

Переменная `fileNameList` (строка 44) содержит имена файлов с изображениями флагов для текущего набора выбранных регионов. Переменная `quizCountriesList` (строка 45) содержит имена файлов с флагами для стран, используемых в текущей игре. В переменной `regionsSet` (строка 46) хранится информация о наборе регионов.

Переменная `correctAnswer` (строка 47) содержит имя файла с флагом для текущего правильного ответа. В переменной `totalGuesses` (строка 48) хранится общее количество правильных и неправильных ответов игрока до настоящего момента. Переменная `correctAnswers` (строка 49) содержит количество правильных ответов; если пользователь завершит викторину, это значение будет равно `FLAGS_IN_QUIZ`. В переменной `guessRows` (строка 50) хранится количество

компонентов `LinearLayout` с двумя кнопками, используемыми для вывода вариантов ответов, — оно определяется настройками приложения (раздел 4.7.4).

Переменная `random` (строка 51) используется для случайного выбора флагов, включаемых в викторину, и местонахождения кнопки, представляющей правильный ответ в компонентах `LinearLayout`. Если пользователь выбрал правильный ответ, но викторина еще не закончена, объект `Handler` (строка 52) используется для загрузки следующего флага после непродолжительной задержки.

Объект анимации `shakeAnimation` (строка 53) содержит динамически заполняемую анимацию «встряхивания», которая применяется к изображению флага в том случае, если пользователь дал неправильный ответ. Строки 55–59 содержат переменные, используемые для выполнения операций с различными компонентами GUI из программного кода.

4.7.3. Переопределение метода onCreateView

Метод `onCreateView` класса `MainActivityFragment` (листинг 4.15) заполняет GUI и инициализирует большинство переменных экземпляров `MainActivityFragment` — `guessRows` и `regionsSet` инициализируются, когда `MainActivity` вызывает методы `updateGuessRows` и `updateRegions` класса `MainActivityFragment`. После вызова метода `onCreateView` суперкласса (строка 65) мы заполняем графический интерфейс `MainActivityFragment` (строки 64–65), используя объект `LayoutInflater`, передаваемый методу `onCreateView` в аргументе. Метод `inflate` класса `LayoutInflater` получает три аргумента:

- идентификатор ресурса макета, определяющий заполняемый макет;
- объект `ViewGroup` (объект макета), в котором будет отображаться макет; передается во втором аргументе `onCreateView`;
- логический признак, который указывает, должен ли заполненный графический интерфейс быть присоединен к объекту `ViewGroup` из второго аргумента. В методе `onCreateView` фрагмента он всегда должен быть равен `false` — система автоматически присоединяет фрагмент к соответствующему объекту `ViewGroup` управляющей активности. Передача `true` приводит к выдаче исключения, потому что графический интерфейс фрагмента уже присоединен.

Метод `inflate` возвращает ссылку на объект `View`, содержащий заполненный графический интерфейс. Ссылка сохраняется в локальной переменной `view`, чтобы ее можно было вернуть из `onCreateView` после инициализации других переменных экземпляров `MainActivityFragment`. (Примечание: мы удалили автоматически сгенерированный пустой конструктор без аргументов (он предшествует методу `onCreateView` в определении класса, созданном IDE), так как

компилятор предоставляет конструктор по умолчанию для любого класса, не имеющего конструктора.)

Листинг 4.15. Переопределенный метод onCreateView класса MainActivityFragment

```
61    // Настройка MainActivityFragment при создании представления
62    @Override
63    public View onCreateView(LayoutInflater inflater, ViewGroup container,
64       Bundle savedInstanceState) {
65       super.onCreateView(inflater, container, savedInstanceState);
66       View view =
67          inflater.inflate(R.layout.fragment_main, container, false);
68
69       fileNameList = new ArrayList<>(); // Оператор <>
70       quizCountriesList = new ArrayList<>();
71       random = new SecureRandom();
72       handler = new Handler();
73
74       // Загрузка анимации для неправильных ответов
75       shakeAnimation = AnimationUtils.loadAnimation(getActivity(),
76          R.anim.incorrect_shake);
77       shakeAnimation.setRepeatCount(3); // Анимация повторяется 3 раза
78
79       // Получение ссылок на компоненты графического интерфейса
80       quizLinearLayout =
81          (LinearLayout) view.findViewById(R.id.quizLinearLayout);
82       questionNumberTextView =
83          (TextView) view.findViewById(R.id.questionNumberTextView);
84       flagImageView = (ImageView) view.findViewById(R.id.flagImageView);
85       guessLinearLayouts = new LinearLayout[4];
86       guessLinearLayouts[0] =
87          (LinearLayout) view.findViewById(R.id.row1LinearLayout);
88       guessLinearLayouts[1] =
89          (LinearLayout) view.findViewById(R.id.row2LinearLayout);
90       guessLinearLayouts[2] =
91          (LinearLayout) view.findViewById(R.id.row3LinearLayout);
92       guessLinearLayouts[3] =
93          (LinearLayout) view.findViewById(R.id.row4LinearLayout);
94       answerTextView = (TextView) view.findViewById(R.id.answerTextView);
95
96       // Настройка слушателей для кнопок ответов
97       for (LinearLayout row : guessLinearLayouts) {
98          for (int column = 0; column < row.getChildCount(); column++) {
99             Button button = (Button) row.getChildAt(column);
100            button.setOnClickListener(guessButtonListener);
101         }
102      }
103
104      // Назначение текста questionNumberTextView
105      questionNumberTextView.setText(
106          getString(R.string.question, 1, FLAGS_IN_QUIZ));
107      return view; // Возвращает представление фрагмента для вывода
108   }
109
```

В строках 69–70 создаются объекты `ArrayList<String>`, в которых хранятся имена файлов с изображениями флагов выбранных регионов и названия стран для текущей серии флагов соответственно. В строке 71 создается объект `SecureRandom` для генерирования случайного набора флагов и кнопок с вариантами. В строке 72 создается объект-обработчик `Handler`, который обеспечивает двухсекундную задержку перед отображением следующего флага, после того как пользователь правильно выберет страну для текущего флага.

В строках 75–76 происходит динамическая загрузка анимации, применяемой к флагу при неправильном ответе. Статический метод `loadAnimation` класса `AnimationUtils` загружает анимацию из XML-файла, представленного константой `R.anim.incorrect_shake`. Первый аргумент обозначает объект `Context`, содержащий ресурсы, к которым применяется анимация, — унаследованный метод `getActivity` возвращает объект `Activity`, управляющий этим объектом `Fragment`. Класс `Activity` является непрямым субклассом `Context`. В строке 77 метод `setRepeatCount` класса `Animation` задает количество повторений анимации.

Строки 80–94 получают ссылки на различные компоненты GUI, с которыми будут выполняться операции на программном уровне. Строки 97–102 последовательно получают каждую кнопку из четырех компонентов `guessLinearLayout` и регистрируют `guessButtonListener` (раздел 4.7.10) как слушателя `OnClickListener` — мы реализуем этот интерфейс для обработки события, инициируемого при касании любой из кнопок ответов.

Строки 105–106 заполняют `questionNumberTextView` текстом строки, возвращаемой при вызове перегруженной версии унаследованного метода класса `Fragment`. В первом аргументе передается строковый ресурс `R.string.question`, который представляет форматную строку `Question %1$d of %2$d`.

Форматная строка содержит заполнители для двух целочисленных значений (см. раздел 4.4.5). В остальных аргументах передаются значения, подставляемые в форматную строку. Строка 107 возвращает графический интерфейс `MainActivityFragment`.

4.7.4. Метод updateGuessRows

Метод `updateGuessRows` (листинг 4.16) вызывается из объекта `MainActivity` при запуске приложения, а также каждый раз, когда пользователь изменяет количество вариантов ответа, отображаемых для каждого флага. Строки 113–114 используют аргумент `SharedPreferences` для получения строки, соответствующей ключу `MainActivity.CHOICES`, — константы с именем, под которым `SettingsActivityFragment` хранит количество отображаемых вариантов ответа.

Строка 115 преобразует значение параметра к типу `int` и делит его на два, чтобы вычислить значение `guessRows`, определяющее количество отображаемых компонентов `guessLinearLayout` (по одному на каждые две кнопки). Затем в строках 118–119 скрываются все компоненты `guessLinearLayout`, чтобы строки 122–123 могли отобразить правильное количество `guessLinearLayout` на основании значения `guessRows`. Константа `View.GONE` (строка 119) означает, что система Android не должна учитывать размеры заданных компонентов при размещении остальных компонентов в макете. Также существует константа `View.INVISIBLE`, которая просто скрывает компонент, а все место, выделенное для этого компонента, остается пустым на экране.

Листинг 4.16. Метод updateGuessRows класса MainActivityFragment

```
110    // Обновление guessRows на основании значения SharedPreferences
111    public void updateGuessRows(SharedPreferences sharedPreferences) {
112       // Получение количества отображаемых вариантов ответа
113       String choices =
114          sharedPreferences.getString(MainActivity.CHOICES, null);
115       guessRows = Integer.parseInt(choices) / 2;
116
117       // Все компоненты LinearLayout скрываются
118       for (LinearLayout layout : guessLinearLayouts)
119             layout.setVisibility(View.GONE);
120
121       // Отображение нужных компонентов LinearLayout
122       for (int row = 0; row < guessRows; row++)
123          guessLinearLayouts[row].setVisibility(View.VISIBLE);
124    }
125
```

4.7.5. Метод updateRegions

Метод `updateRegions` (листинг 4.17) вызывается из объекта `MainActivity` при запуске приложения, а также каждый раз, когда пользователь изменяет набор регионов. В строках 128–129 аргумент `SharedPreferences` используется для получения имен всех включенных регионов в виде коллекции `Set<String>`. Константа `MainActivity.REGIONS` содержит имя, под которым в объекте `SettingsActivityFragment` хранится список включаемых регионов.

Листинг 4.17. Метод updateRegions класса MainActivityFragment

```
126    // Обновление выбранных регионов по данным из SharedPreferences
127    public void updateRegions(SharedPreferences sharedPreferences) {
128       regionsSet =
129          sharedPreferences.getStringSet(MainActivity.REGIONS, null);
130    }
131
```

4.7.6. Метод resetQuiz

Метод `resetQuiz` (листинг 4.18) настраивает и запускает викторину. Напомним, что изображения, используемые в игре, хранятся в папке `assets` приложения. Чтобы обратиться к содержимому этой папки, метод получает объект `AssetManager` приложения (строка 135) вызовом метода `getAssets` родительской активности. Затем строка 136 очищает `fileNameList`, чтобы подготовиться к загрузке имен файлов изображений только для географических регионов, включенных в викторину. Цикл в строках 140–147 перебирает все включенные регионы. Для каждого региона вызывается метод `list` класса `AssetManager` (строка 143) для получения массива имен файлов с изображениями флагов, который сохраняется в строковом массиве `paths`. В строках 145–146 из имени файла удаляется расширение `.png`, а оставшиеся имена помещаются в `fileNameList`. Метод `list` класса `AssetManager` выдает исключения `IOException`, которые являются *проверяемыми* (поэтому исключение необходимо перехватить или объявить). Если исключение происходит из-за того, что приложение не может получить доступ к папке `assets`, в строках 149–152 исключение перехватывается и регистрируется для последующей отладки встроенными средствами Android. Статический метод `e` класса `Log` используется для регистрации сообщений об ошибках в журнале. Полный список методов `Log` доступен по адресу

http://developer.android.com/reference/android/util/Log.html

Листинг 4.18. Метод resetQuiz класса MainActivityFragment

```
132    // Настройка и запуск следующей серии вопросов
133    public void resetQuiz() {
134       // Использование AssetManager для получения имен файлов изображений
135       AssetManager assets = getActivity().getAssets();
136       fileNameList.clear(); // Пустой список имен файлов изображений
137
138       try {
139          // Перебрать все регионы
140          for (String region : regionsSet) {
141             // get a list of all flag image files in this region
142             String[] paths = assets.list(region);
143
144             for (String path : paths)
145                fileNameList.add(path.replace(".png", ""));
146          }
147       }
148       catch (IOException exception) {
149          Log.e(TAG, "Error loading image file names", exception);
150       }
151
152       correctAnswers = 0; // Сброс количества правильных ответов
153       totalGuesses = 0; // Сброс общего количества попыток
```

```
154       quizCountriesList.clear(); // Очистка предыдущего списка стран
155
156       int flagCounter = 1;
157       int numberOfFlags = fileNameList.size();
158
159       // Добавление FLAGS_IN_QUIZ случайных файлов в quizCountriesList
160       while (flagCounter <= FLAGS_IN_QUIZ) {
161          int randomIndex = random.nextInt(numberOfFlags);
162
163          // Получение случайного имени файла
164          String filename = fileNameList.get(randomIndex);
165
166          // Если регион включен, но еще не был выбран
167          if (!quizCountriesList.contains(filename)) {
168             quizCountriesList.add(filename); // Добавить файл в список
169             ++flagCounter;
170          }
171       }
172
173       loadNextFlag(); // Запустить викторину загрузкой первого флага
174    }
175
```

Затем в строках 154–156 обнуляются счетчики правильных попыток, сделанных пользователем (`correctAnswers`), и общего количества попыток (`totalGuesses`), а также очищается список `quizCountriesList`.

В строках 160–171 в список `quizCountriesList` добавляются `FLAGS_IN_QUIZ` (10) случайно выбранных имен файлов. Мы получаем общее количество флагов, после чего генерируем случайный индекс в диапазоне от 0 до количества флагов, уменьшенного на 1. Сгенерированный индекс используется для выбора одного имени файла из `fileNamesList`. Если `quizCountriesList` еще не содержит это имя, оно добавляется в `quizCountriesList` с увеличением счетчика `flagCounter`. Процесс повторяется до тех пор, пока не будут выбраны `FLAGS_IN_QUIZ` уникальных имен файлов. Затем в строке 173 вызывается метод `loadNextFlag` (листинг 4.19) для загрузки первого флага.

4.7.7. Метод loadNextFlag

Метод `loadNextFlag` (листинг 4.19) загружает и отображает следующий флаг и соответствующий набор кнопок с вариантами ответа. В списке `quizCountriesList` хранятся имена файлов в формате

регион-страна

без расширения `.png`. Если *регион* или *страна* содержат несколько слов, то слова разделяются символом подчеркивания (_).

Листинг 4.19. Метод loadNextFlag класса MainActivityFragment

```
176    // Загрузка следующего флага после правильного ответа
177    private void loadNextFlag() {
178       // Получение имени файла следующего флага и удаление его из списка
179       String nextImage = quizCountriesList.remove(0);
180       correctAnswer = nextImage; // Обновление правильного ответа
181       answerTextView.setText(""); // Очистка answerTextView
182
183       // Отображение номера текущего вопроса
184       questionNumberTextView.setText(getString(
185          R.string.question, (correctAnswers + 1), FLAGS_IN_QUIZ));
186
187       // Извлечение региона из имени следующего изображения
188       String region = nextImage.substring(0, nextImage.indexOf('-'));
189
190       // Использование AssetManager для загрузки следующего изображения
191       AssetManager assets = getActivity().getAssets();
192
193       // Получение объекта InputStream для ресурса следующего флага
194       // и попытка использования InputStream
195       try (InputStream stream =
196          assets.open(region + "/" + nextImage + ".png")) {
197          // Загрузка графики в виде объекта Drawable и вывод на flagImageView
198          Drawable flag = Drawable.createFromStream(stream, nextImage);
199          flagImageView.setImageDrawable(flag);
200
201          animate(false); // Анимация появления флага на экране
202       }
203       catch (IOException exception) {
204          Log.e(TAG, "Error loading " + nextImage, exception);
205       }
206
207       Collections.shuffle(fileNameList); // Перестановка имен файлов
208
209       // Помещение правильного ответа в конец fileNameList
210       int correct = fileNameList.indexOf(correctAnswer);
211       fileNameList.add(fileNameList.remove(correct));
212
213       // Добавление 2, 4, 6 или 8 кнопок в зависимости от значения guessRows
214       for (int row = 0; row < guessRows; row++) {
215          // Размещение кнопок в currentTableRow
216          for (int column = 0;
217               column < guessLinearLayouts[row].getChildCount();
218               column++) {
219             // Получение ссылки на Button
220             Button newGuessButton =
221                (Button) guessLinearLayouts[row].getChildAt(column);
222             newGuessButton.setEnabled(true);
223
224             // Назначение названия страны текстом newGuessButton
225             String filename = fileNameList.get((row * 2) + column);
226             newGuessButton.setText(getCountryName(filename));
227          }
```

```
228        }
229
230        // Случайная замена одной кнопки правильным ответом
231        int row = random.nextInt(guessRows); // Выбор случайной строки
232        int column = random.nextInt(2); // Выбор случайного столбца
233        LinearLayout randomRow = guessLinearLayouts[row]; // Получение строки
234        String countryName = getCountryName(correctAnswer);
235        ((Button) randomRow.getChildAt(column)).setText(countryName);
236    }
237
```

Строка 179 удаляет первое имя из `quizCountriesList` и сохраняет его в `nextImage`. Имя также сохраняется в `correctAnswer` для последующей проверки правильности ответа. Затем приложение очищает `answerTextView` и выводит следующий номер вопроса в поле `questionNumberTextView` (строки 184–185) с использованием отформатированного строкового ресурса `R.string.question`.

Строка 188 извлекает из `nextImage` регион, который должен использоваться как имя вложенной папки из `assets`, из которой будет загружаться изображение. Затем мы получаем объект `AssetManager` и используем его в команде `try` с ресурсами для открытия потока `InputStream` (пакет `java.io`), чтобы прочитать байты из файла с изображением флага. Поток передается в аргументе статического метода `createFromStream` класса `Drawable`, создающем объект `Drawable` (пакет `android.graphics.drawable`). Объект `Drawable` назначается визуальным источником данных компонента `flagImageView` вызовом метода `setImageDrawable`. Если происходит исключение, оно регистрируется для отладки (строка 204). Затем вызывается метод `animate` с аргументом `false`, чтобы вывести следующий флаг и набор кнопок на экран при помощи анимации (строка 201).

Затем строка 207 переставляет элементы `fileNameList` в случайном порядке, а строки 210–211 находят правильный ответ `correctAnswer` и перемещают его в конец `fileNameList` — позднее этот ответ будет случайно помещен на одну из кнопок с вариантами.

Цикл в строках 214–228 перебирает кнопки в компонентах `guessLinearLayout` для текущего количества панелей с кнопками `guessRows`. Для каждой кнопки:

- строки 220–221 получают ссылку на следующую кнопку;
- строка 222 разблокирует кнопку;
- строка 225 получает имя файла с флагом из `fileNameList`;
- строка 226 заполняет текст кнопки названием страны, возвращаемым методом `getCountryName` (раздел 4.7.8).

Строки 231–235 выбирают случайную строку (на основании текущего значения `guessRows`) и столбец кнопок, после чего задают текст соответствующей кнопки.

4.7.8. Метод getCountryName

Метод `getCountryName` (листинг 4.20) выделяет название страны из имени файла с изображением. Сначала выделяется подстрока, которая начинается с дефиса, отделяющего регион от названия страны. Затем вызывается метод `replace` класса `String` для замены подчеркиваний (`_`) пробелами.

Листинг 4.20. Метод getCountryName класса MainActivityFragment

```
238    // Метод разбирает имя файла с флагом и возвращает название страны
239    private String getCountryName(String name) {
240       return name.substring(name.indexOf('-') + 1).replace('_', ' ');
241    }
242
```

4.7.9. Метод animate

Метод `animate` (листинг 4.21) применяет анимацию кругового раскрытия со всем макетом (`quizLinearLayout`) при переходе к следующему вопросу. Строки 246–247 немедленно переходят к первому вопросу, чтобы первый вопрос просто появлялся на экране без анимации. Строки 250–253 вычисляют экранные координаты центра пользовательского интерфейса. Строки 256–257 вычисляют максимальный радиус круга в анимации (минимальный радиус всегда равен 0). Метод `animate` получает один параметр `animateOut` и может использоваться двумя способами. Строка 262 на основании `animateOut` определяет, должна ли анимация отображать или скрывать изображение.

Листинг 4.21. Метод animate класса MainActivityFragment

```
243    // Весь макет quizLinearLayout появляется или исчезает с экрана
244    private void animate(boolean animateOut) {
245       // Предотвращение анимации интерфейса для первого флага
246       if (correctAnswers == 0)
247          return;
248
249       // Вычисление координат центра
250       int centerX = (quizLinearLayout.getLeft() +
251          quizLinearLayout.getRight()) / 2;
252       int centerY = (quizLinearLayout.getTop() +
253          quizLinearLayout.getBottom()) / 2;
254
255       // Вычисление радиуса анимации
256       int radius = Math.max(quizLinearLayout.getWidth(),
257          quizLinearLayout.getHeight());
258
259       Animator animator;
260
261       // Если изображение должно исчезать с экрана
262       if (animateOut) {
```

```
263          // Создание круговой анимации
264          animator = ViewAnimationUtils.createCircularReveal(
265             quizLinearLayout, centerX, centerY, radius, 0);
266          animator.addListener(
267             new AnimatorListenerAdapter() {
268                // Вызывается при завершении анимации
269                @Override
270                public void onAnimationEnd(Animator animation) {
271                   loadNextFlag();
272                }
273             }
274          );
275       }
276       else { // Если макет quizLinearLayout должен появиться
277          animator = ViewAnimationUtils.createCircularReveal(
278             quizLinearLayout, centerX, centerY, 0, radius);
279       }
280
281       animator.setDuration(500); // Анимация продолжительностью 500 мс
282       animator.start(); // Начало анимации
283    }
284
```

Если `animate` вызывается со значением `true`, то метод выполняет анимацию исчезновения `quizLinearLayout` с экрана (строки 264–274). В строках 264–265 создается объект `Animator` для анимации кругового появления, для чего вызывается метод `createCircularReveal` класса `ViewAnimationUtils`. Метод получает пять параметров:

- первый параметр задает представление, к которому должна применяться анимация (`quizLinearLayout`);
- второй и третий параметры определяют координаты центра круга анимации;
- последние два параметра определяют начальный и конечный радиус круга анимации.

Так как `quizLinearLayout` исчезает с экрана, начальный радиус совпадает с вычисленным, а конечный равен 0. Строки 266–274 создают и связывают объект `AnimatorListenerAdapter` с `Animator`. Метод `onAnimationEnd` (строки 269–272) объекта `AnimatorListenerAdapter` вызывается при завершении анимации и загрузке следующего флага (строка 271).

Если `animate` вызывается со значением `false`, то метод отображает анимацию появления `quizLinearLayout` на экране в начале следующего вопроса. Строки 277–278 создают объект `Animator` вызовом метода `createCircularReveal`, но на этот раз начальный радиус устанавливается равным 0, а конечный равен вычисленному. В результате `quizLinearLayout` исчезает с экрана, вместо того чтобы появляться на нем.

Строка 281 вызывает метод `setDuration` объекта `Animator` для назначения продолжительности анимации (500 миллисекунд). Наконец, в строке 282 запускается анимация.

4.7.10. Анонимный внутренний класс, реализующий интерфейс OnClickListener

В строках 97–102 (листинг 4.22) объект `guessButtonListener` (листинг 4.29) регистрируется как обработчик события для каждой кнопки с вариантом ответа. Переменная экземпляра `guessButtonListener` содержит ссылку на объект анонимного внутреннего класса, реализующий интерфейс `OnClickListener` для обработки событий кнопок. Метод получает нажатую кнопку в параметре `v`. Мы получаем текст кнопки (строка 290) и название страны (строка 291), после чего увеличиваем `totalGuesses`. Если ответ правилен (строка 294), мы увеличиваем счетчик `correctAnswers`. Затем компонент `answerTextView` заполняется текстом с названием страны, а в качестве цвета шрифта выбирается цвет, представленный константой `R.color.correct_answer` (зеленый). После этого вызывается вспомогательный метод `disableButtons` (раздел 4.7.11), блокирующий все кнопки ответов.

Листинг 4.22. Анонимный внутренний класс, реализующий OnClickListener

```
285    // Вызывается при нажатии кнопки ответа
286    private OnClickListener guessButtonListener = new OnClickListener() {
287       @Override
288       public void onClick(View v) {
289          Button guessButton = ((Button) v);
290          String guess = guessButton.getText().toString();
291          String answer = getCountryName(correctAnswer);
292          ++totalGuesses; // Увеличение количества попыток пользователя
293
294          if (guess.equals(answer)) { // Если ответ правилен
295             ++correctAnswers; // Увеличить количество правильных ответов
296
297             // Правильный ответ выводится зеленым цветом
298             answerTextView.setText(answer + "!");
299             answerTextView.setTextColor(
300                getResources().getColor(R.color.correct_answer,
301                   getContext().getTheme()));
302
303             disableButtons(); // Блокировка всех кнопок ответов
304
305             // Если пользователь правильно угадал FLAGS_IN_QUIZ флагов
306             if (correctAnswers == FLAGS_IN_QUIZ) {
307                // DialogFragment для вывода статистики и перезапуска
308                DialogFragment quizResults =
309                   new DialogFragment() {
310                   // Создание окна AlertDialog
```

```
311                 @Override
312                 public Dialog onCreateDialog(Bundle bundle) {
313                    AlertDialog.Builder builder =
314                       new AlertDialog.Builder(getActivity());
315                    builder.setMessage(
316                       getString(R.string.results,
317                          totalGuesses,
318                          (1000 / (double) totalGuesses)));
319
320                    // Кнопка сброса "Reset Quiz"
321                    builder.setPositiveButton(R.string.reset_quiz,
322                       new DialogInterface.OnClickListener() {
323                          public void onClick(DialogInterface dialog,
324                             int id) {
325                                resetQuiz();
326                          }
327                       }
328                    );
329
330                    return builder.create(); // Вернуть AlertDialog
331                 }
332              };
333
334           // Использование FragmentManager для вывода DialogFragment
335           quizResults.setCancelable(false);
336           quizResults.show(getFragmentManager(), "quiz results");
337           }
338           else { // Ответ правильный, но викторина не закончена
339              // Загрузка следующего флага после двухсекундной задержки
340              handler.postDelayed(
341                 new Runnable() {
342                    @Override
343                    public void run() {
344                      animate(true); // Анимация исчезновения флага
345                    }
346                 }, 2000); // 2000 миллисекунд для двухсекундной задержки
347           }
348        }
349        else { // Неправильный ответ
350           flagImageView.startAnimation(shakeAnimation); // Встряхивание
351
352           // Сообщение "Incorrect!" выводится красным шрифтом
353           answerTextView.setText(R.string.incorrect_answer);
354           answerTextView.setTextColor(getResources().getColor(
355               R.color.incorrect_answer, getContext().getTheme()));
356           guessButton.setEnabled(false); // Блокировка неправильного ответа
357        }
358     }
359  };
360
```

Если значение `correctAnswers` равно `FLAGS_IN_QUIZ` (строка 306), значит, викторина завершена. В строках 308–332 создается новый анонимный внутренний класс, расширяющий класс `DialogFragment` (пакет `android.support.a4.app`),

который будет использоваться для вывода результатов. Метод `onCreateDialog` класса `DialogFragment` использует класс `AlertDialog.Builder` (см. ниже) для настройки и создания `AlertDialog`, после чего возвращает его. Когда пользователь касается кнопки `Reset Quiz` в диалоговом окне, вызывается метод `resetQuiz`, который запускает новую игру (строка 325). Строка 335 означает, что диалоговое окно не отменяется, — пользователь должен взаимодействовать с диалоговым окном, потому что касание за пределами диалогового окна или касание кнопки `Back` не вернет пользователя к викторине. Чтобы отобразить фрагмент `DialogFragment`, строка 336 вызывает его метод `show`, передавая в аргументах объект `FragmentManager`, полученный при вызове `getFragmentManager` и `String`. Второй аргумент может использоваться с методом `getFragmentByTag` класса `FragmentManager` для получения ссылки на `DialogFragment` в будущем — в нашем приложении эта возможность не используется.

Если значение `correctAnswers` меньше `FLAGS_IN_QUIZ`, то в строках 340–346 вызывается метод `postDelayed` объекта `handler`. Первый аргумент определяет анонимный внутренний класс, реализующий интерфейс `Runnable`, — он представляет выполняемую задачу `animate(true)`, которая убирает флаги и кнопки ответа с экрана и начинает переход к следующему вопросу через заданное количество миллисекунд. Второй аргумент определяет задержку в миллисекундах (2000). Если ответ неправилен, то строка 350 вызывает метод `startAnimation` компонента `flagImageView` для воспроизведения анимации `shakeAnimation`, загруженной в методе `onCreateView`. Также компонент `answerTextView` настраивается для вывода текста `"Incorrect!"` красным шрифтом (строки 353–355), а кнопка, соответствующая неправильному ответу, блокируется.

СОВЕТ ПО ОФОРМЛЕНИЮ 4.4

Заголовок AlertDialog (который выводится над сообщением диалогового окна) можно назначить методом setTitle класса AlertDialog.Builder. Согласно рекомендациям по дизайну приложений Android (http://developer.android.com/design/building-blocks/dialogs.html) для большинства диалоговых окон заголовки не обязательны. Диалоговое окно должно отображать заголовок для «небезопасных операций, сопряженных с возможным риском потери данных, разрывом связи, дополнительными затратами и т. д.». Кроме того, диалоговые окна, в которых отображаются списки вариантов, используют заголовки для описания предназначения диалогового окна.

Создание и настройка AlertDialog

В строках 313–329 объект `AlertDialog.Builder` используется для создания и настройки `AlertDialog`.

Строки 313–314 создают объект `AlertDialog.Builder`, передавая активность фрагмента в аргументе `Context`, — диалоговое окно будет отображаться в контексте

активности, управляющей MainActivityFragment. Затем строки 315–318 задают сообщению диалогового окна отформатированную строку с результатами викторины — ресурс R.string.results содержит заполнители для общего количества попыток и процента правильных ответов.

В этом окне AlertDialog должна присутствовать только одна кнопка для подтверждения сообщения и сброса состояния викторины. Эта кнопка назначается *позитивной кнопкой* диалогового окна (строки 321–328) — касание этой кнопки означает, что пользователь подтверждает сообщение, выведенное в окне. Методу setPositiveButton передается надпись на кнопке (заданная строковым ресурсом R.string.reset_quiz) и ссылка на обработчик события кнопки. Нашему диалоговому окну обрабатывать это событие не нужно, поэтому вместо обработчика события передается null. В данном случае передается объект анонимного внутреннего класса, реализующего интерфейс DialogInterface.OnClickListener. Мы переопределяем метод onClick этого интерфейса, чтобы реагировать на событие при нажатии пользователем соответствующей кнопки диалогового окна.

4.7.11. Метод disableButtons

Метод disableButtons (листинг 4.23) перебирает кнопки с вариантами ответов и блокирует их. Этот метод вызывается в том случае, когда пользователь дает правильный ответ.

Листинг 4.23. Метод disableButtons класса MainActivityFragment

```
361    // Вспомогательный метод, блокирующий все кнопки
362    private void disableButtons() {
363       for (int row = 0; row < guessRows; row++) {
364          LinearLayout guessRow = guessLinearLayouts[row];
365          for (int i = 0; i < guessRow.getChildCount(); i++)
366             guessRow.getChildAt(i).setEnabled(false);
367       }
368    }
369 }
```

4.8. Класс SettingsActivity

Класс SettingsActivity (листинг 4.24) управляет SettingsActivityFragment при запуске приложения в портретной ориентации. Переопределенный метод onCreate (строки 11–18) вызывает метод setContentView класса Activity, чтобы заполнить графический интерфейс, определяемый в файле activity_settings.xml, представленный ресурсом R.layout.activity_ settings. Затем отображается объект Toolbar, определенный в макете SettingsActivity. Строка 17 отображает

на панели приложения кнопку Up, при прикосновении к которой пользователь возвращается к родительской активности MainActivity. IDE добавляет эту строку при включении SettingsActivity в проект и назначении его иерархического родителя (раздел 4.4.12). Мы удалили из этого класса остальной сгенерированный код, не используемый в приложении. Также можно удалить неиспользуемый ресурс меню menu_settings.xml.

Листинг 4.24. SettingsActivity отображает SettingsActivityFragment на телефонах и планшетах в портретной ориентации

```
1 // SettingsActivity.java
2 // Activity to display SettingsActivityFragment on a phone
3 package com.deitel.flagquiz;
4
5 import android.os.Bundle;
6 import android.support.v7.app.AppCompatActivity;
7 import android.support.v7.widget.Toolbar;
8
9 public class SettingsActivity extends AppCompatActivity {
10    // Заполняет GUI, отображает Toolbar и добавляет кнопку "up"
11    @Override
12    protected void onCreate(Bundle savedInstanceState) {
13       super.onCreate(savedInstanceState);
14       setContentView(R.layout.activity_settings);
15       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
16       setSupportActionBar(toolbar);
17       getSupportActionBar().setDisplayHomeAsUpEnabled(true);
18    }
19 }
```

4.9. Класс SettingsActivityFragment

Класс SettingsFragment (листинг 4.25) расширяет класс PreferenceFragment (пакет android.preference). При создании объекта SettingsActivityFragment переопределенный метод onCreate (строки 10–14) вызывает унаследованный от PreferenceFragment метод addPreferencesFromResource для построения графического интерфейса управления конфигурацией на базе preferences.xml (раздел 4.4.11). В процессе работы с графическим интерфейсом настройки автоматически сохраняются в файле SharedPreferences на устройстве. Если файл не существует, он создается; в противном случае он обновляется. Другой автоматически сгенерированный код удален из класса.

Листинг 4.25. Субкласс PreferenceFragment для управления настройками приложения

```
1 // SettingsActivityFragment.java
2 // Субкласс PreferenceFragment для управления настройками приложения
3 package com.deitel.flagquiz;
```

```
 4
 5 import android.os.Bundle;
 6 import android.preference.PreferenceFragment;
 7
 8 public class SettingsActivityFragment extends PreferenceFragment {
 9    // Создание GUI настроек по файлу preferences.xml из res/xml
10    @Override
11    public void onCreate(Bundle bundle) {
12       super.onCreate(bundle);
13       addPreferencesFromResource(R.xml.preferences); // Загрузить из XML
14    }
15 }
```

4.10. AndroidManifest.xml

В листинге 4.26 показан автоматически сгенерированный манифест приложения `Flag Quiz`. Каждая активность в приложении должна быть объявлена в файле `AndroidManifest.xml` приложения; в противном случае Android не будет знать о том, что активность существует, и не сможет запустить ее. При создании приложения среда разработки включила объявление класса `MainActivity` в `AndroidManifest.xml`. Запись

```
.MainActivity
```

в строке 12 означает, что класс находится в пакете, указанном в строке 2, и является сокращением для

```
com.deitel.flagquiz.MainActivity
```

Мы добавили строку 14, о которой речь пойдет ниже.

Листинг 4.26. AndroidManifest.xml с объявлением SettingsActivity

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <manifest package="com.deitel.flagquiz"
 3    xmlns:android="http://schemas.android.com/apk/res/android">
 4
 5    <application
 6       android:allowBackup="true"
 7       android:icon="@mipmap/ic_launcher"
 8       android:label="@string/app_name"
 9       android:supportsRtl="true"
10       android:theme="@style/AppTheme">
11       <activity
12          android:name=".MainActivity"
13          android:label="@string/app_name"
14          android:launchMode="singleTop"
15          android:theme="@style/AppTheme.NoActionBar">
16          <intent-filter>
```

```
17              <action android:name="android.intent.action.MAIN"/>
18
19              <category android:name="android.intent.category.LAUNCHER"/>
20           </intent-filter>
21        </activity>
22        <activity
23           android:name=".SettingsActivity"
24           android:label="@string/title_activity_settings"
25           android:parentActivityName=".MainActivity"
26           android:theme="@style/AppTheme.NoActionBar">
27           <meta-data
28              android:name="android.support.PARENT_ACTIVITY"
29              android:value="com.deitel.flagquiz.MainActivity">
30        </activity>
31     </application>
32
33 </manifest>
```

При добавлении `SettingsActivity` в проект (раздел 4.4.1) среда разработки автоматически включила `SettingsActivity` в файл манифеста (строки 22–30). Если бы новая активность создавалась без использования средств IDE, вам пришлось бы объявить новую активность, включив элемент `<activity>` наподобие приведенного в строках 22–30. За полной информацией о файле манифеста обращайтесь по адресу

http://developer.android.com/guide/topics/manifest/manifest-intro.html

Свойство launchMode

В строке 14 задается параметр `launchMode` класса `MainActivity`. По умолчанию все созданные вами активности используют «стандартный» режим запуска. В этом режиме при получении интента для запуска активности Android создает новый экземпляр активности.

Вспомните, как в разделе 4.4.12 мы назначили *иерархического родителя* `SettingsActivity`. Напомним, что это позволяет Android определить на панели приложения кнопку `Up`, при помощи которой пользователь может перейти к заданной родительской активности. Когда пользователь касается кнопки, а родительская активность использует «стандартный» режим запуска, Android использует интент для запуска родительской активности. В результате создается новый экземпляр `MainActivity`. Это также приводит к тому, что приложение `Flag Quiz` аварийно завершается в слушателе `OnSharedPreferenceChangeListener` активности `MainActivity` (раздел 4.6.7), когда тот пытается обновить несуществующий фрагмент `quizFragment` — потому что тот был определен в другом экземпляре `MainActivity`.

Строка 14 решает эту проблему, заменяя значение атрибута `launchMode` активности `MainActivity` значением `"singleTop"`. В этом режиме запуска при нажатии

кнопки Up Android выводит существующий экземпляр MainActivity на передний план — вместо того чтобы создавать новый объект MainActivity. За дополнительной информацией о значениях атрибута launchMode элемента <activity> обращайтесь по адресу

https://developer.android.com/guide/topics/manifest/activity-element.html#lmode

4.11. Резюме

В этой главе было создано приложение Flag Quiz для проверки знания пользователем флагов различных государств. Ключевой особенностью этой главы стало использование фрагментов для формирования частей графического интерфейса активности. Мы использовали две активности для отображения фрагментов QuizFragment и SettingsFragment при запуске приложения в портретной ориентации и одну активность для отображения обоих фрагментов при выполнении приложения на планшете в альбомной ориентации — таким образом обеспечивается более эффективное использование пространства экрана. Субкласс PreferenceFragment помогает автоматизировать процесс хранения и загрузки настроек приложения, а субкласс DialogFragment отображает диалоговое окно AlertDialog. Мы рассмотрели основные стадии жизненного цикла Fragment и показали, как при помощи FragmentManager получить ссылку на фрагмент для выполнения с ним операций в программе.

В портретной ориентации меню действий приложения использовалось для отображения активности SettingsActivity, содержащей SettingsActivityFragment. Для запуска SettingsActivity использовался явный интент. Вы видели, как получить настройки из файла SharedPreferences и как изменить этот файл при помощи объекта SharedPreferences.Editor.

Мы показали, как при помощи объекта Configuration определить, выполняется ли приложение на планшете в альбомной ориентации. Также в этой главе было показано, как управлять большим количеством графических ресурсов с использованием вложенных папок в папке assets приложения и как обращаться к этим ресурсам через AssetManager. Мы построили объект Drawable на основе байтов изображения, для этого мы прочитали их из InputStream, а затем отобразили Drawable в ImageView.

Также были рассмотрены другие папки из папки res приложения — menu для хранения файлов ресурсов меню, anim для хранения файлов ресурсов анимации и xml для хранения файлов с разметкой XML. Также вы узнали, как использовать квалификаторы при создании папки для хранения макета, который должен использоваться только на больших устройствах в альбомной ориентации. Мы также показали, как использовать ресурс списка цветов состояний, для того

чтобы гарантировать удобочитаемость текста на кнопках как для доступного, так и для заблокированного состояния.

Временные окна `Toast` использовались для вывода сообщений о второстепенных ошибках или информации, которая должна на непродолжительное время появляться на экране. Чтобы следующий флаг выводился после короткой задержки, мы использовали объект `Handler`, выполняющий задачу `Runnable` по истечении заданного промежутка в миллисекундах. Вы узнали, что задача `Runnable` выполняется в программном потоке, создавшем `Handler` (UI-поток в этом приложении).

Мы определяем анимацию в файле XML и применяем ее к компоненту `ImageView` приложения, если пользователь выбрал неправильный ответ. Класс `ViewAnimationUtils` использовался для создания круговой анимации перехода между вопросами. Вы научились сохранять информацию об исключениях для последующей диагностики с использованием встроенного механизма Android и класса `Log`. Мы также использовали вспомогательные классы и интерфейсы из пакета `java.util`, включая `List`, `ArrayList`, `Collections` и `Set`.

В главе 5 будет представлено приложение `Doodlz`, использующее графические возможности Android для превращения экрана устройства в *виртуальный холст*. Также вы узнаете о *режиме погружения* и возможностях печати.

5 Приложение Doodlz

Двумерная графика, объекты Canvas и Bitmap, акселерометр, объекты SensorManager, события многоточечных касаний, MediaStore, печать, разрешения Android 6.0, Gradle

В этой главе…

- Обнаружение событий, связанных с касанием экрана, перемещением пальца вдоль экрана и отведением пальца от экрана
- Обработка многоточечных касаний экрана и возможность рисования несколькими пальцами одновременно
- Использование объекта SensorManager и акселерометра для обнаружения событий перемещения
- Применение объектов Paint для определения цвета и ширины линии
- Использование объектов Path для хранения данных линий и объектов Canvas для рисования линий на Bitmap
- Создание меню и отображение команд на панели действий
- Использование инфраструктуры печати и класса PrintHelper из Android Support Library для вывода изображений на печать
- Использование новой модели разрешений Android 6.0 для сохранения изображений во внешнем хранилище
- Добавление библиотек в приложение с помощью системы сборки Gradle

5.1. Введение

Приложение `Doodlz` позволяет рисовать на экране одним или несколькими пальцами (рис. 5.1). В приложении предусмотрены инструменты для определения цвета и толщины линии, а также дополнительные команды для:

- очистки экрана;
- сохранения текущего рисунка на устройстве;
- вывода текущего рисунка на печать.

В зависимости от размера экрана некоторые команды приложения могут отображаться в виде значков на панели приложения, а те, которые на панели приложения не поместились, отображаются в виде текста в раскрывающемся меню (⋮) на панели приложения.

Рис. 5.1. Приложение Doodlz с завершенным рисунком

В этом приложении также применяется новый механизм разрешений Android 6.0. Например, Android запрашивает у пользователя разрешение для сохранения файлов (рисунка на экране) на устройстве. В Android 6.0 вместо того чтобы запрашивать у пользователя полный список необходимых разрешений во время установки, приложение запрашивает каждое разрешение по отдельности — в тот

момент, когда это разрешение необходимо для выполнения задачи в первый раз. В этом приложении Android запрашивает разрешение при первой попытке сохранения рисунка.

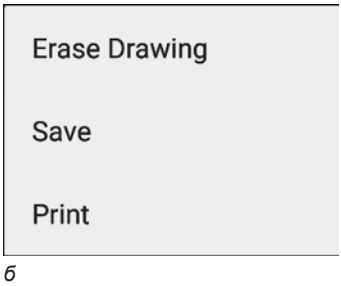
Начнем с тестового запуска приложения, а затем рассмотрим технологии, использованные при его построении. Далее приводится обзор графического интерфейса приложения. В завершение мы рассмотрим полный код приложения, при этом особое внимание будет уделяться новым возможностям приложения.

5.2. Тестирование приложения Doodlz на виртуальном устройстве AVD

Запустите Android Studio и откройте приложение `Doodlz` из папки `Doodlz` в примерах книги, после чего запустите приложение на AVD или на устройстве. Среда разработки строит и запускает проект.

На рис. 5.2, *а* и 5.2, *б* показана панель приложения и раскрывающееся меню на Nexus 6 AVD; на рис. 5.2, *в* показана панель приложения в AVD для Nexus 6.

а



б

в

Рис. 5.2. Панель приложения Doodlz и раскрывающееся меню: *а* — панель приложения в AVD Nexus 6; *б* — раскрывающееся меню в AVD Nexus 6; *в* — панель приложения в AVD Nexus 9, на панели хватает места для размещения всех команд меню в виде значков

В приложении используются следующие элементы меню.

- `Color` (🎨) — открывает диалоговое окно для изменения цвета линии.
- `Line Width` (🖌) — открывает диалоговое окно для изменения толщины линии, которую пользователь рисует пальцем на экране.

- `Erase Image` () — сначала запрашивает подтверждение для удаления всего изображения, а затем стирает рисунок, если операция не была отменена.
- `Save` () — сохраняет изображение на устройстве. Чтобы просмотреть изображение в приложении Google `Photos`, откройте меню приложения и коснитесь команды `Device Folders`, чтобы увидеть миниатюры сохраненных изображений[1].
- Print () — отображает интерфейс для выбора принтера, чтобы вы могли вывести изображение на печать или сохранить его в формате PDF (используется по умолчанию).

Вскоре мы рассмотрим каждую из этих команд.

СОВЕТ ПО ОФОРМЛЕНИЮ

Когда команда меню отображается на панели приложения и у этой команды меню имеется значок, отображается этот значок; в противном случае выводится текст команды меню. Любые команды меню, не помещающиеся на панели приложения, могут быть вызваны из раскрывающегося меню (), в котором команды меню представлены текстовыми метками.

Изменение цвета кисти на красный

Чтобы изменить цвет кисти, сначала коснитесь команды на панели приложения или выберите команду `Color` в раскрывающемся меню, если значок не отображается на панели. На экране появится диалоговое окно выбора цвета (рис. 5.3).

Цвета определяются с помощью цветовой схемы *ARGB*, в которой с помощью целочисленных цветовых значений (от 0 до 255) определены следующие цветовые компоненты: *альфа-канал* (прозрачность), красный, зеленый и синий. Для альфа-канала значение 0 соответствует полной прозрачности, а 255 — полной непрозрачности. Для красной, зеленой и синей составляющей 0 означает нулевую интенсивность, а 255 — максимальную интенсивность этого цвета. Интерфейс настройки цвета состоит из четырех полос `SeekBar`, с помощью которых можно выбирать степень прозрачности и интенсивности красного, зеленого и синего цветов. Цвета изменяются перетаскиванием ползунка `SeekBar`, при этом образец нового цвета отображается под полосами. Выберите красный цвет, перетащив ползунок `Red` в крайнее правое положение, как на рис. 5.3. Прикоснитесь к кнопке `SET COLOR`, чтобы выбрать цвет и закрыть диалоговое окно. Если вы передумали изменять цвет, достаточно коснуться за пределами окна, чтобы закрыть его.

[1] На некоторых устройствах для успешного сохранения файлов из приложения Doodlz необходимо сначала сделать снимок на камеру устройства.

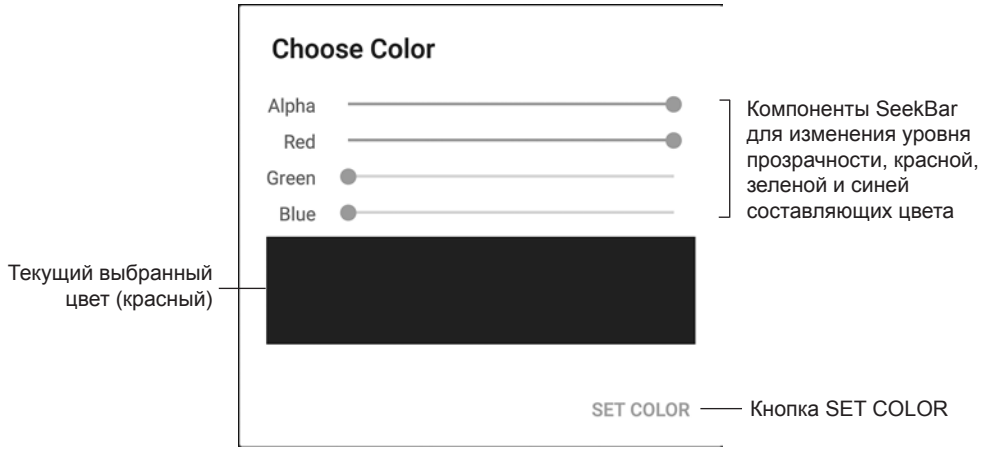


Рис. 5.3. Выбор красного цвета для рисования

Чтобы переключиться в «режим ластика», просто выберите белый цвет (переместите все четыре ползунка вправо).

Изменение толщины линии

Чтобы изменить толщину линии, коснитесь команды на панели действий или выберите команду `LINE WIDTH` в раскрывающемся меню, если значок не отображается на панели. На экране появляется диалоговое окно `Choose Line Width`. Перетащите полосу `SeekBar`, предназначенную для настройки толщины линии, вправо (рис. 5.4), чтобы увеличить толщину линии. Прикоснитесь к кнопке `Set Line Width`, чтобы вернуться в область рисования.

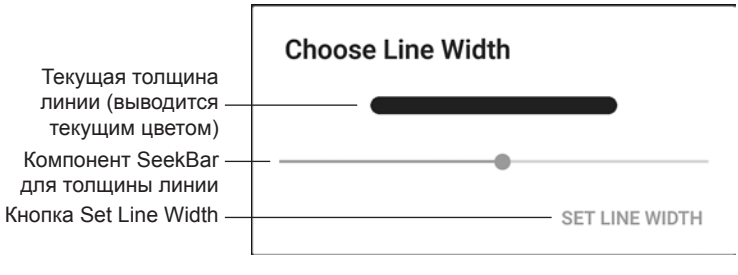


Рис. 5.4. Изменение толщины линии

Рисование лепестков

Проведите пальцем (мышью при использовании эмулятора) по области рисования, чтобы нарисовать лепестки цветка (рис. 5.5).

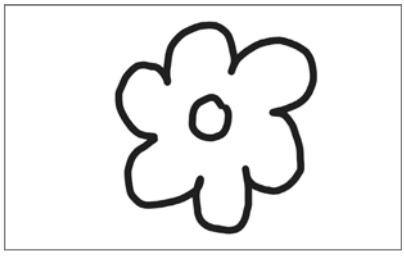


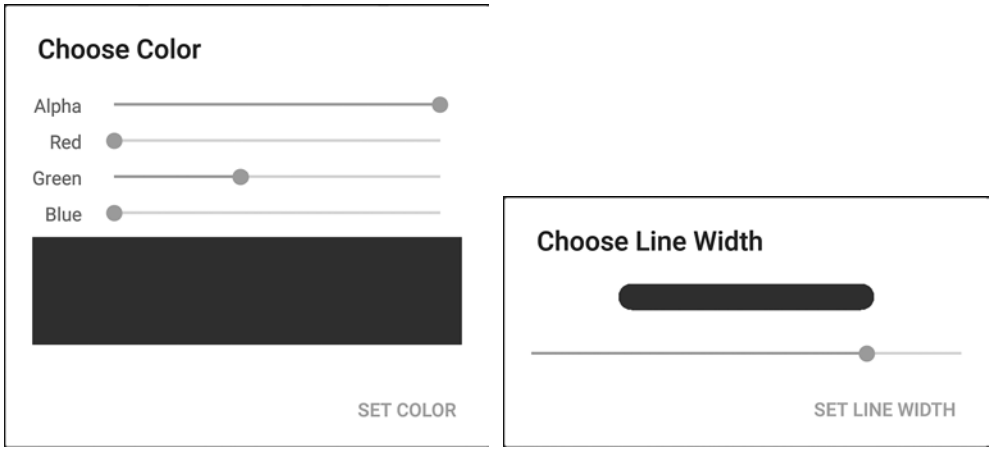
Рис. 5.5. Рисование лепестков

Изменение цвета кисти на темно-зеленый

Коснитесь команды (🎨) или выберите команду `Color` для вызова диалогового окна `Choose Color`. Выберите темно-зеленый цвет, перетащив ползунок полосы `Green` в крайнее правое положение, а ползунки полос `Red` и `Blue` — в крайнее левое положение (рис. 5.6, *а*).

Изменение толщины линии для рисования стебля и листьев

Коснитесь команды (🖌) или выберите команду `Line Width` для вызова диалогового окна `Choose Line Width`. Перетащите полосу `SeekBar`, предназначенную для настройки толщины линии, вправо (рис. 5.6, *б*). Нарисуйте стебель цветка и листья. Повторите последние два шага для выбора более светлого оттенка зеленого и меньшей толщины линии и нарисуйте траву (рис. 5.7).



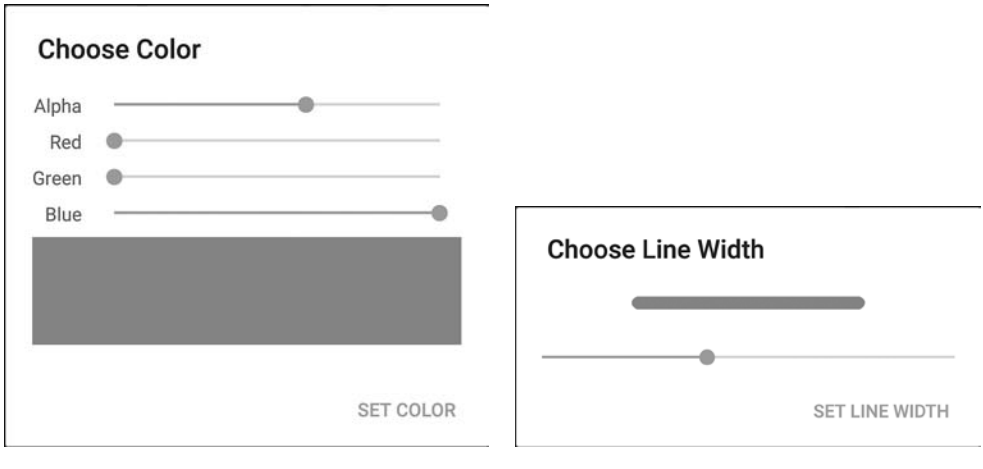
а *б*

Рис. 5.6. Переключение на темно-зеленый цвет и увеличение толщины линии: *а* — выбор темно-зеленого цвета линии; *б* — увеличение толщины линии

Рис. 5.7. Рисование стебля и травы

Завершение рисунка

Переключитесь на полупрозрачный синий цвет линии (рис. 5.8, *а*) и меньшую толщину (рис. 5.8, *б*). Затем нарисуйте капли дождя (рис. 5.9).



а *б*

Рис. 5.8. Выбор синего цвета и меньшей толщины линии: *а* — выбор синего цвета линии; *б* — уменьшение толщины линии

Рис. 5.9. Рисование дождевых капель с выбором другого цвета и толщины линии

Сохранение изображения

При желании можно сохранить изображение на устройстве и просмотреть его в приложении `Photos`. Для этого коснитесь на панели приложения или выберите команду `Save` в раскрывающемся меню, если значок не отображается на панели. Чтобы просмотреть это и другие изображения, хранящиеся на устройстве, откройте приложение `Photos`.

Печать изображения

Чтобы распечатать изображение, коснитесь на панели приложения или выберите команду `Print` в раскрывающемся меню, если значок не отображается на панели. На экране появляется диалоговое окно печати, которое по умолчанию предлагает сохранить изображение в виде документа PDF. Чтобы выбрать принтер, выберите режим `Save as PDF` и выберите один из доступных принтеров. Если в списке нет ни одного принтера, возможно, вам придется настроить сервис Google Cloud Print для вашего принтера. Дополнительная информация доступна на странице

http://www.google.com/cloudprint/learn/

5.3. Обзор применяемых технологий

В этом разделе рассматриваются новые технологии, задействованные при создании приложения `Doodlz`.

5.3.1. Методы жизненного цикла активности и фрагмента

Жизненный цикл фрагмента связывается с жизненным циклом его родительской активности. Существуют шесть методов жизненного цикла активности, у которых имеются соответствующие методы жизненного цикла фрагмента — `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` и `onDestroy`. Когда система вызывает эти методы для активности, это приводит к вызову соответствующих методов всех присоединенных фрагментов активности (а возможно, и других методов жизненного цикла фрагментов).

В данном приложении используются методы жизненного цикла фрагмента `onResume` и `onPause`. Метод `onResume` вызывается в тот момент, когда фрагмент находится на экране и готов к взаимодействию с пользователем. Когда активность, управляющая фрагментами, возобновляет работу, вызываются методы `onResume` всех ее фрагментов. В нашем приложении `MainActivityFragment` переопределяет `onResume` для включения прослушивания событий акселерометра, чтобы пользователь мог встряхнуть устройство для стирания текущего изображения (раздел 5.7.3).

Метод активности `onPause` вызывается при передаче фокуса другой активности, что приводит к приостановке активности, потерявшей фокус, и переводе ее в фоновый режим. Когда активность, управляющая фрагментами, приостанавливается, вызываются методы `onPause` всех ее фрагментов. В нашем приложении `MainActivityFragment` переопределяет `onPause` для приостановки прослушивания событий акселерометра (раздел 5.7.4).

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ 5.1

В момент приостановки приложение должно отключать слушателей для событий датчика, чтобы эти события не доставлялись приложению, пока оно не выводит ничего на экран. Это делается для экономии заряда батареи.

Другие методы жизненного цикла активностей и фрагментов будут рассмотрены по мере необходимости. За дополнительной информацией о полном жизненном цикле активности обращайтесь по адресу

http://developer.android.com/reference/android/app/Activity.html #ActivityLifecycle

Подробности о полном жизненном цикле фрагментов доступны по адресу

http://developer.android.com/guide/components/fragments.html#Lifecycle

5.3.2. Пользовательские представления

Вы можете создать собственное представление, расширив класс `View` или один из его субклассов, как мы это делаем с классом `DoodleView` (раздел 5.8), расширяющим `View`. Чтобы добавить пользовательский компонент в XML-файл макета, необходимо указать его *полное имя* (то есть имя, включающее имя пакета и класса); это означает, что класс пользовательского представления должен существовать до того, как он будет добавлен в макет. В разделе 5.5.2 будет показано, как создать класс `DoodleView` и добавить его в макет.

5.3.3. Использование SensorManager для прослушивания событий акселерометра

Для удаления текущего рисунка, созданного с помощью приложения, достаточно встряхнуть устройство. Многие устройства Android снабжены *акселерометрами*, с помощью которых приложения отслеживают перемещения устройств. Также в настоящее время Android поддерживает датчики гравитации, гироскопы, датчики освещения, линейного ускорения, магнитного поля, ориентации, давления, приближения, вектора поворота и температуры. Перечень констант `Sensor`, представляющих эти типы датчиков, приведен по адресу

http://developer.android.com/reference/android/hardware/Sensor.html

Обработка событий акселерометра и датчиков рассматривается в разделе 5.7. За информацией о других датчиках Android обращайтесь по адресу

http://developer.android.com/guide/topics/sensors/sensors_overview.html

5.3.4. Пользовательские реализации DialogFragment

В предыдущих приложениях для вывода или получения данных от пользователя (в виде нажатий кнопок) применялись окна `AlertDialog` во фрагментах `DialogFragment`. Объекты `AlertDialog`, использовавшиеся ранее, создавались с помощью анонимных внутренних классов, которые расширяли `DialogFragment` и позволяли выводить только текст и кнопки. Объекты `AlertDialog` также могут содержать пользовательские представления. В этом приложении будут определены следующие три субкласса `DialogFragment`.

- `ColorDialogFragment` (раздел 5.9) отображает окно `AlertDialog` с пользовательским представлением, содержащим компоненты GUI для предварительного просмотра и выбора нового цвета линии в формате ARGB.
- `LineWidthDialogFragment` (раздел 5.10) отображает окно `AlertDialog` с пользовательским представлением, содержащим компоненты GUI для предварительного просмотра и выбора толщины линии.
- `EraseImageDialogFragment` (раздел 5.11) отображает стандартное окно `AlertDialog` для подтверждения стирания всего рисунка.

Пользовательские представления фрагментов `ColorDialogFragment` и `EraseImageDialogFragment` будут заполняться из ресурсного файла макета. В каждом из трех субклассов `DialogFragment` также будут переопределяться следующие методы жизненного цикла `Fragment`:

- `onAttach` — *первый* метод жизненного цикла фрагмента, вызываемый при присоединении фрагмента к родительской активности;
- `onDetach` — *последний* метод жизненного цикла фрагмента, вызываемый перед отсоединением фрагмента от родительской активности.

Предотвращение одновременного появления нескольких диалоговых окон

Обработчик события встряхивания может попытаться вывести диалоговое окно подтверждения стирания в тот момент, когда на экране уже находится другое диалоговое окно. Чтобы этого не произошло, мы воспользуемся методами `onAttach` и `onDetach` для задания логической переменной, которая сообщает, когда диалоговое окно находится на экране. Если переменная истинна, то приложение запрещает обработчику события встряхивания отображать диалоговое окно.

5.3.5. Рисование с использованием Canvas, Paint и Bitmap

Методы класса `Canvas` могут использоваться для рисования текста, линий и кругов. Вывод осуществляется на объектах `Bitmap` (пакет `android.graphics`). Разработчик может связать с `Bitmap` объект `Canvas`, а затем использовать `Canvas` для рисования на `Bitmap`; изображение появится на экране (см. раздел. 5.8). Объект `Bitmap` также можно сохранить в файле — мы воспользуемся этой возможностью для сохранения рисунков в галерее изображений устройства командой `Save`. Каждый метод графического вывода в классе `Canvas` использует объект класса `Paint` (пакет `android.graphics`) для определения характеристик рисования: цвет,

толщина линии, размер шрифта и т. д. В приложении эти возможности представлены в методе `onDraw` класса `Doodleview` (раздел 5.8.6).

За дополнительной информацией о характеристиках, задаваемых при помощи объекта `Paint`, обращайтесь по адресу

http://developer.android.com/reference/android/graphics/Paint.html

5.3.6. Обработка событий многоточечных касаний и хранение данных линий в объектах Path

Чтобы создать рисунок, пользователь проводит одним или несколькими пальцами по экрану. Приложение сохраняет информацию от *каждого* пальца в объекте `Path` (пакет `android.graphics`), представляющем геометрический контур из набора отрезков и кривых. Обработка событий касания осуществляется переопределением метода `onTouchEvent` класса `View` (раздел 5.8.7). Этот метод получает объект `MotionEvent` (пакет `android.view`) с информацией о типе произошедшего события касания и идентификатором пальца, сгенерировавшего событие. По идентификатору можно различать пальцы и добавлять информацию в соответствующие объекты `Path`. По типу события касания мы определяем, какую операцию выполнил пользователь — коснулся экрана, провел по экрану или отнял палец от экрана.

Кроме стандартной обработки событий касания, в Android 6.0 предусмотрена расширенная обработка для приложений, использующих Bluetooth-стилусы. В ней учитывается сила нажатия и кнопка стилуса, нажатая пользователем. Например, в нашем приложении кнопка стилуса может включать режим стирания или сила нажатия может динамически изменять толщину линии во время рисования. За дополнительной информацией обращайтесь по адресу

https://developer.android.com/about/versions/marshmallow/android-6.0.html#bluetooth-stylus

5.3.7. Сохранение данных на устройстве

Команда меню `Save` сохраняет рисунок на устройстве. Изображение можно просмотреть в приложении `Photos`; выберите команду `Device Folders` из меню приложения, чтобы просмотреть миниатюры хранимых изображений — коснитесь миниатюры, чтобы просмотреть полноразмерное изображение. С помощью объекта `ContentResolver` (пакет `android.content`) приложение считывает данные с устройства, а также сохраняет их на этом же устройстве. В приложении объект

`ContentResolver` (раздел 5.8.11) и метод `insertImage` класса `MediaStore.Images.Media` используются для сохранения изображения в галерее приложения `Photos`. Объект `MediaStore` управляет графическими, звуковыми и видеофайлами, хранящимися на устройстве.

5.3.8. Поддержка печати и класс PrintHelper из Android Support Library

В нашем приложении класс `PrintHelper` (раздел 5.8.12) из инфраструктуры печати Android используется для вывода текущего изображения на печать. Класс `PrintHelper` предоставляет пользовательский интерфейс для выбора принтера, содержит метод для проверки поддержки печати заданным устройством, а также метод для вывода `Bitmap` на печать. Класс `PrintHelper` входит в Android Support Library — группу библиотек, часто используемых для реализации новых возможностей в более старых версиях Android. Библиотеки также включают дополнительные вспомогательные средства (такие как класс `PrintHelper`) для конкретных версий Android.

5.3.9. Новая модель разрешения Android 6.0 (Marshmallow)

Прежде чем записывать данные во внешнее хранилище, необходимо сначала получить разрешение `android.permission.WRITE_EXTERNAL_PERMISSION`. В приложении `Doodlz` это разрешение необходимо для сохранения рисунка, созданного пользователем.

В Android 6.0 (Marshmallow) появилась новая модель разрешений, более удобная для пользователя. До выхода Android 6.0 пользователь должен был во время установки заранее предоставить все разрешения, которые могли когда-либо понадобиться приложению, — это приводило к тому, что многие люди отказывались от установки некоторых приложений. С новой моделью приложение устанавливается, не требуя разрешений, а пользователь предоставляет разрешение только при первом использовании нужной функции.

Предоставленное разрешение сохраняется у приложения до тех пор, пока приложение не будет переустановлено или пользователь не изменит разрешения приложения в приложении Android `Settings`.

О том, как реализуется новая модель разрешений, рассказано в разделах 5.7.8–5.7.9.

5.3.10. Добавление зависимостей с использованием системы сборки Gradle

Android Studio использует систему сборки Gradle для компиляции кода приложения в файл APK (устанавливаемый пакет). Gradle также управляет зависимостями проекта — в частности, включением в процесс сборки библиотек, используемых приложением. В приложении `Doodlz` в проект будет добавлена зависимость для библиотеки поддержки, чтобы вы могли использовать класс `PrintHelper` для вывода изображений (раздел 5.4.2).

5.4. Создание проекта и ресурсов

В этом разделе мы создадим проект, импортируем значки материального дизайна для команд меню приложения, а также отредактируем различные ресурсы, используемые приложением `Doodlz`.

5.4.1. Создание проекта

Начните с создания нового проекта на базе шаблона `Blank Activity`. На шаге `New Project` диалогового окна `Create New Project` укажите следующие значения:

- `Application name`: `Doodlz`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

На остальных шагах диалогового окна `Create New Project` используйте те же настройки, что и в разделе 4.4.1. Android Studio создает активность `MainActivity` с встроенным фрагментом. Фрагмент управляет областью рисования и обеспечивает реакцию на касания пользователя. Также добавьте в проект значок приложения так, как описано в разделе 2.5.2.

Когда проект откроется в Android Studio, в макетном редакторе выберите Nexus 6 в раскрывающемся списке виртуальных устройств (см. рис. 2.8). Также удалите компонент `TextView` с текстом `Hello World!` из макета `fragment_main.xml` и кнопку `FloatingActionButton` из `activity_main.xml`.

5.4.2. Gradle: добавление библиотеки поддержки в проект

Для использования класса `PrintHelper` в приложении необходима библиотека поддержки Android Support Library. Чтобы добавить библиотеку поддержки в число зависимостей проекта, выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `app` и выберите команду `Open Module Settings`.
2. В открывшемся окне `Project Structure` перейдите на вкладку `Dependencies`.
3. Щелкните на кнопке `Add` (+) и выберите вариант `Library dependency`. На экране появляется диалоговое окно `Choose Library Dependency`.
4. Выберите в списке вариант `support-v4 (com.android.support:support-v4:23.1.0)` и нажмите кнопку `OK`. Зависимость появляется в списке на вкладке `Dependencies`.
5. Нажмите кнопку `OK`. IDE выводит сообщение `Gradle project sync in progress...`, пока проект настраивается для использования библиотеки Android Support Library.

За дополнительной информацией о том, когда использовать и как настраивать Android Support Library, обращайтесь по адресам

http://developer.android.com/tools/support-library

http://developer.android.com/tools/support-library/setup.html

5.4.3. Файл strings.xml

Строковые ресурсы уже создавались в предыдущих главах, поэтому сейчас мы приведем только таблицу с именами ресурсов и соответствующими значениями (табл. 5.1). Сделайте двойной щелчок на файле `strings.xml` из папки `res/values`, а затем щелкните на ссылке `Open editor`, чтобы запустить редактор `Translations Editor` для создания этих строковых ресурсов.

СОВЕТ ПО ОФОРМЛЕНИЮ 5.2

Для языков с поддержкой символов верхнего регистра спецификация материального дизайна Google рекомендует записывать текст на кнопке в верхнем регистре (например, CANCEL или SET COLOR).

Таблица 5.1. Строковые ресурсы, используемые в приложении Doodlz

Ключ	Значение по умолчанию
button_erase	Erase Image
button_set_color	Set Color
button_set_line_width	Set Line Width
line_imageview_description	This displays the line thickness
label_alpha	Alpha

Таблица 5.1 (*окончание*)

Ключ	Значение по умолчанию
label_red	Red
label_green	Green
label_blue	Blue
menuitem_color	Color
menuitem_delete	Erase Drawing
menuitem_line_width	Line Width
menuitem_save	Save
menuitem_print	Print
message_erase	Erase the drawing?
message_error_saving	There was an error saving the image
message_saved	Your saved painting can be viewed in the Photos app by selecting Device Folders from that app\'s menu [Примечание: \' — служебная последовательность для представления апострофа ('). Без префикса \ IDE выдает сообщение «Перед апострофом нет символа \».]
message_error_printing	Your device does not support printing
permission_explanation	To save an image, the app requires permission to write to external storage
title_color_dialog	Choose Color
title_line_width_dialog	Choose Line Width

5.4.4. Импортирование значков для команд меню

В меню приложения указаны значки для всех команд. Значок отображается в том случае, если команда помещается на панели приложения (впрочем, это зависит от устройства). Используйте приемы, описанные в разделе 4.4.9, для импортирования следующих векторных значков:

-  (`ic_palette_24dp`)
-  (`ic_brush_24dp`)

-  (ic_delete_24dp)
-  (ic_save_24dp)
-  (ic_print_24dp)

В круглых скобках указаны имена, которые будут отображаться в подсказке диалогового окна Vector Asset Studio при наведении указателя мыши на изображение. Для каждого изображения откройте файл XML и замените значение fillColor на

```
@android:color/white,
```

чтобы значки выводились белым цветом на синей панели приложения.

5.4.5. Меню MainActivityFragment

В главе 4 для отображения команды меню Settings в приложении Flag Quiz мы отредактировали стандартное меню, сгенерированное средой разработки. В этом приложении мы определим собственное меню для MainActivity, поэтому файл main.xml можно удалить из папки res/menu проекта. Также можно удалить методы из класса MainActivity, методы onCreateOptionsMenu и onOptionsItemSelected, так как они в приложении не используются.

Меню для разных версий Android

Не забудьте, что печать поддерживается только в Android 4.4 и выше. Если вы разрабатываете приложение с меню для разных версий Android, включите в него несколько ресурсов меню при помощи квалификаторов, упоминавшихся в описаниях предыдущих приложений. Например, можно создать два ресурса меню: один для версий Android до 4.4 и другой для версий 4.4 и выше. В ресурсе меню для старых версий отсутствуют команды меню, недоступные в этих версиях. За дополнительной информацией о создании ресурсов меню обращайтесь по адресу

http://developer.android.com/guide/topics/ui/menus.html

Создание меню

Чтобы создать ресурс меню, выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке res/menu и выберите команду New ▸ Menu Resource File.
2. Введите в поле File name имя doodle_fragment_menu.xml и нажмите кнопку OK. IDE открывает в редакторе файл с разметкой XML-файла. Чтобы добавить

новые команды в ресурс меню, вам придется напрямую редактировать разметку.

3. В этом меню свойство `showAsAction` каждой команды будет указывать, что команда должна отображаться на панели приложения, если там хватает свободного места. Если в приложении используется библиотека поддержки для реализации панели приложения с обратной совместимостью, необходимо использовать атрибут `showAsAction` из пространства имен XML `app` (вместо пространства имен `android`). Чтобы включить пространство имен `app`, отредактируйте открывающий тег элемента `<menu>`:

```
xmlns:app="http://schemas.android.com/apk/res-auto"
```

4. Добавьте в файл XML разметку первой команды меню из листинга 5.1. Свойство `id` команды должно быть равно `@+id/color`, свойство `title` — `@string/menuitem_color`, свойство `icon` — `@drawable/ic_palette_24dp` и свойство `showAsAction` — `ifRoom`. Значение `ifRoom` означает, что система Android должна разместить команду меню на панели приложения только при наличии свободного места; в противном случае команда меню будет отображаться в текстовом виде в раскрывающемся меню в правой части панели приложения. Другие возможные значения `showAsAction` доступны по адресу

   http://developer.android.com/guide/topics/resources/menu-resource.html

Листинг 5.1. Элемент <item>, представляющий команду меню

```
1 <item
2    android:id="@+id/color"
3    android:title="@string/menuitem_color"
4    android:icon="@drawable/ic_palette_24dp"
5    app:showAsAction="ifRoom">
6 </item>
```

5. Повторите шаг 4 для всех свойств `id` и `title` из табл. 5.2, чтобы создать команды меню `Line Width`, `Delete`, `Save` и `Print`. Сохраните и закройте файл меню. Полная разметка XML приведена в листинге 5.2.

Таблица 5.2. Дополнительные команды меню MainActivityFragment

Id	Title
@+id/line_width	@string/menuitem_line_width
@+id/delete_drawing	@string/menuitem_delete
@+id/save	@string/menuitem_save
@+id/print	@string/menuitem_print

Листинг 5.2. doodle_fragment_menu.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
   <item
      android:id="@+id/color"
      android:title="@string/menuitem_color"
      android:icon="@drawable/ic_palette_24dp"
      app:showAsAction="ifRoom">
   </item>

   <item
      android:id="@+id/line_width"
      android:title="@string/menuitem_line_width"
      android:icon="@drawable/ic_brush_24dp"
      app:showAsAction="ifRoom">
   </item>

   <item
      android:id="@+id/delete_drawing"
      android:title="@string/menuitem_delete"
      android:icon="@drawable/ic_delete_24dp"
      app:showAsAction="ifRoom">
   </item>

   <item
      android:id="@+id/save"
      android:title="@string/menuitem_save"
      android:icon="@drawable/ic_save_24dp"
      app:showAsAction="ifRoom">
   </item>

   <item
      android:id="@+id/print"
      android:title="@string/menuitem_print"
      android:icon="@drawable/ic_print_24dp"
      app:showAsAction="ifRoom">
   </item>
</menu>
```

5.4.6. Добавление разрешения в файл AndroidManifest.xml

Кроме новой модели разрешений Android 6.0, в которой приложение динамически запрашивает необходимые разрешения у пользователя, каждое приложение также должно указать все используемые разрешения в файле `AndroidManifest.xml`.

1. Откройте папку `manifests` проекта и откройте файл `AndroidManifest.xml`.
2. Добавьте в элемент `<manifest>` и перед элементом `<application>`

```
<uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

5.5. Построение графического интерфейса

В этом разделе мы построим графический интерфейс приложения и создадим классы для его диалоговых окон.

5.5.1. Макет content_main.xml для MainActivity

Макет `content_main.xml` активности `MainActivity` приложения содержит только фрагмент `MainActivityFragment`, который был создан автоматически при создании проекта. Чтобы упростить повторное использование кода, мы изменим свойство `id` фрагмента.

1. Откройте файл `content_main.xml` на вкладке `Design` макетного редактора.
2. Выберите фрагмент в окне `Component Tree`, в окне свойств задайте свойству `id` значение `doodleFragment` и сохраните макет.

5.5.2. Макет fragment_main.xml для MainActivityFragment

Макет `fragment_main.xml` класса `MainActivityFragment` должен отображать только представление `DoodleView`. При создании проекта в этот файл макета был автоматически включен компонент `RelativeLayout`. Чтобы заменить корневой элемент `RelativeLayout` на `DoodleView`, необходимо сначала создать класс `DoodleView` (субкласс `View`), чтобы вы могли выбрать его при размещении пользовательского представления в макете.

1. Откройте папку `java` в окне `Project`.
2. Щелкните правой кнопкой мыши на узле `com.deitel.doodlz`, выберите команду `New` ▸ `Java Class`.
3. В открывшемся диалоговом окне `Create New Class` введите в поле `Name` значение `DoodleView` и нажмите кнопку `OK`. Файл автоматически открывается в редакторе.
4. В файле `DoodleView.java` укажите, что класс `DoodleView` является субклассом `View`; для этого добавьте в определение класса текст `extends View`. Если IDE не добавит команду `import` для `android.view.View`, установите курсор сразу же за `extends View`. Щелкните на красной лампочке () над началом определения класса `DoodleView` и выберите команду `Import Class`.
5. IDE выводит сообщение об ошибке, в котором говорится, что вы не определили конструктор для нового класса. Чтобы решить проблему, поместите курсор сразу же за `extends View`. Щелкните на красной лампочке () над началом

определения класса DoodleView и выберите команду Create constructor matching super. В диалоговом окне Choose Super Class Constructors выберите конструктор с двумя аргументами и нажмите кнопку OK. IDE включает конструктор в класс. Код конструктора будет добавлен в разделе 5.8.3. Конструктор с двумя аргументами вызывается Android в процессе заполнения DoodleView на основе макета — второй аргумент задает свойства View, заданные в файле XML с макетом. О конструкторах класса View можно больше узнать по адресу

http://developer.android.com/reference/android/view/View.html#View(android.content.Context)

6. Вернитесь к файлу fragment_main.xml в макетном редакторе и перейдите на вкладку Text.
7. Замените RelativeLayout на com.deitel.doodlz.DoodleView.
8. Удалите свойства padding для верхней, левой, правой и нижней сторон — представление DoodleView должно занимать весь экран.
9. В режиме Design выберите в окне Component Tree компонент CustomView - com.deitel.doodlz.DoodleView, а затем задайте id значение doodleView.
10. Сохраните и закройте файл fragment_main.xml.

5.5.3. Макет fragment_color.xml для фрагмента ColorDialogFragment

Файл fragment_color.xml для фрагмента ColorDialogFragment содержит двухстолбцовый компонент GridLayout с графическим интерфейсом выбора и предварительного просмотра образца цвета. В этом разделе мы создадим макет ColorDialogFragment и класс ColorDialogFragment. Чтобы создать макет fragment_color.xml, выполните следующие действия.

1. Откройте узел res/layout в окне Project.
2. Щелкните правой кнопкой мыши на папке layout и выберите команду New ▸ Layout resource file. На экране появляется диалоговое окно New Resource File.
3. В поле File name введите текст fragment_color.xml.
4. В поле Root element введите GridLayout и нажмите кнопку OK.
5. В окне Component Tree выберите компонент GridLayout.
6. В окне свойств задайте свойству id значение colorDialogGridLayout, а свойству columnCount значение 2.

7. Из палитры макетного редактора перетащите компоненты `Plain TextView` и `SeekBar` на узел `colorDialogGridLayout` в окне `Component Tree`. Перетащите элементы в том порядке, в котором они перечислены на рис. 5.10, и задайте свойству `id` каждого компонента значение, приведенное в таблице.

Рис. 5.10. Окно Component Tree для fragment_color.xml

Добавление colorView в макет

Представлению `colorView` не нужен собственный класс — для изменения цвета, отображаемого в `colorView`, мы будем использовать методы класса `View` на программном уровне. Android Studio не предоставляет визуальных средств для включения объекта класса `View` в макет, поэтому для добавления `colorView` придется редактировать разметку XML напрямую.

1. Щелкните на вкладке `Text` в нижней части макетного редактора, чтобы перейти из режима представления `Design` в режим разметки.
2. Добавьте код в листинге 5.3 сразу же после тега `</GridLayout>`.

Листинг 5.3. fragment_color.xml

```
1    <View
2       android:layout_width="wrap_content"
3       android:layout_height="@dimen/color_view_height"
4       android:id="@+id/colorView"
5       android:layout_column="0"
6       android:layout_columnSpan="2"
7       android:layout_gravity="fill_horizontal"/>
```

3. Вернитесь на вкладку `Design` в макетном редакторе.
4. Задайте свойства компонентов GUI так, как показано в табл. 5.3. Напомним, что для метрики `color_view_height` в диалоговом окне `Resources` можно нажать кнопку `New Resource` и выбрать вариант `New Dimension Value...`, чтобы

открыть диалоговое окно `New Dimension Value Resource`. Задайте `color_view_height` значение `80dp`.

5. Сохраните и закройте файл `fragment_color.xml`.

Таблица 5.3. Значения свойств компонентов GUI в файле fragment_color.xml

Компонент	Свойство	Значение
colorDialogGridLayout	columnCount	2
	orientation	vertical
	useDefaultMargins	true
	padding top	@dimen/activity_vertical_margin
	padding bottom	@dimen/activity_vertical_margin
	padding left	@dimen/activity_horizontal_margin
	padding right	@dimen/activity_horizontal_margin
alphaTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	0
	Другие свойства	
	text	@string/label_alpha
alphaSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	0
	Другие свойства	
	max	255
redTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	1
	Другие свойства	
	text	@string/label_red
redSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	1
	Другие свойства	
	max	255

Таблица 5.3 (*окончание*)

Компонент	Свойство	Значение
greenTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	2
	Другие свойства	
	text	@string/label_red
greenSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	2
	Другие свойства	
	max	255
blueTextView	Параметры layout	
	layout:column	0
	layout:gravity	right, center_vertical
	layout:row	3
	Другие свойства	
	text	@string/label_red
blueSeekBar	Параметры layout	
	layout:column	1
	layout:gravity	fill_horizontal
	layout:row	3
	Другие свойства	
	max	255
colorView	Параметры layout	
	layout:height	@dimen/color_view_height
	layout:column	0
	layout:columnSpan	2
	layout:gravity	fill_horizontal

Добавление класса ColorDialogFragment в проект

Чтобы добавить класс `ColorDialogFragment` в проект, выполните следующие действия.

1. Щелкните правой кнопкой мыши на пакете `com.deitel.doodlz` в папке `java` и выберите команду `New` ▸ `Java Class`. На экране появляется диалоговое окно `Create New Class`.

2. Введите в поле Name значение ColorDialogFragment.
3. Нажмите кнопку OK, чтобы создать класс. Код этого класса будет создан в разделе 5.9.

5.5.4. Макет fragment_line_width.xml для фрагмента LineWidthDialogFragment

Файл fragment_line_width.xml для фрагмента LineWidthDialogFragment содержит компонент GridLayout с графическим интерфейсом выбора и предварительного просмотра толщины линии. В этом разделе мы создадим макет LineWidthDialogFragment и класс LineWidthDialogFragment. Чтобы создать макет fragment_line_width.xml, выполните следующие действия.

1. Откройте узел res/layout в окне Project.
2. Щелкните правой кнопкой мыши на папке layout и выберите команду New ▸ Layout resource file. На экране появится диалоговое окно New Resource File.
3. В поле File name введите имя файла fragment_line_width.xml.
4. В поле Root Element выберите компонент GridLayout и нажмите OK.
5. В окне Component Tree выделите компонент GridLayout и замените значение его свойства id на lineWidthDialogGridLayout.
6. Используя палитру макетного редактора, перетащите компоненты ImageView и SeekBar на узел lineWidthDialogGridLayout в окне Component Tree. Окно должно выглядеть так, как показано на рис. 5.11. Задайте идентификаторы элементов в соответствии с иллюстрацией.

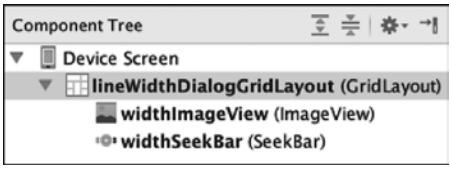


Рис. 5.11. Окно Component Tree для fragment_line_width.xml

7. Задайте свойства компонентов GUI в соответствии с табл. 5.4. Задайте line_imageview_height значение 50dp.
8. Сохраните и закройте файл fragment_line_width.xml.

Таблица 5.4. Значения свойств компонентов GUI в файле fragment_line_width.xml

Компонент	Свойство	Значение
lineWidthDialogGridLayout	columnCount	1
	orientation	vertical
	useDefaultMargins	true
	padding top	@dimen/activity_vertical_margin
	padding bottom	@dimen/activity_vertical_margin
	padding left	@dimen/activity_horizontal_margin
	padding right	@dimen/activity_horizontal_margin
widthImageView	Параметры layout	
	layout:height	@dimen/line_imageview_height
	layout:gravity	fill_horizontal
	Другие свойства	
	contentDescription	@string/line_imageview_description
widthSeekBar	Параметры layout	
	layout:gravity	fill_horizontal
	Другие свойства	
	max	50

Добавление класса LineWidthDialogFragment в проект

Чтобы добавить класс `LineWidthDialogFragment` в проект, выполните следующие действия.

1. Щелкните правой кнопкой мыши на пакете `com.deitel.doodlz` в папке `java` и выберите команду `New` ▸ `Java Class`. На экране появляется диалоговое окно `Create New Class`.
2. Введите в поле `Name` значение `LineWidthDialogFragment`.
3. Нажмите кнопку `OK`, чтобы создать класс.

5.5.5. Добавление класса EraseImageDialogFragment

Для класса `EraseImageDialogFragment` ресурс макета не нужен, поскольку он просто выводит окно `AlertDialog` с текстом. Чтобы добавить в проект класс `EraseImageDialogFragment`, выполните следующие действия.

1. Щелкните правой кнопкой мыши на пакете `com.deitel.doodlz` в папке `java` и выберите команду `New` ▸ `Class`. На экране появляется диалоговое окно `Create New Class`.

2. Введите в поле `Name` значение `EraseImageDialogFragment`.
3. Нажмите кнопку `OK`, чтобы создать класс.

5.6. Класс MainActivity

Приложение состоит из шести классов.

- `MainActivity` (см. ниже) — родительская активность для фрагментов этого приложения.
- `MainActivityFragment` (раздел 5.7) — управляет `DoodleView` и обработкой событий акселерометра.
- `DoodleView` (раздел 5.8) — предоставляет функции рисования, сохранения и печати.
- `ColorDialogFragment` (раздел 5.9) — субкласс `DialogFragment`, отображаемый командой меню для выбора цвета.
- `LineWidthDialogFragment` (раздел 5.10) — субкласс `DialogFragment`, отображаемый командой меню для выбора толщины линии.
- `EraseImageDialogFragment` (раздел 5.11) — субкласс `DialogFragment`, отображаемый командой меню для стирания или встряхиванием устройства для стирания текущего рисунка.

Метод `onCreate` класса `MainActivity` (листинг 5.4) заполняет графический интерфейс (строка 16), после чего использует средства, описанные в разделе 4.6.3, для определения размера устройства и назначения ориентации `MainActivity`. Если приложение выполняется на очень большом экране (строка 26), выбирается альбомная ориентация (строки 27–28); в противном случае назначается портретная ориентация (строки 30–31). Другие автоматически сгенерированные методы были удалены из класса `MainActivity`, поскольку в этом приложении они не используются.

Листинг 5.4. Класс MainActivity

```
1 // MainActivity.java
2 // Подготовка макета MainActivity
3 package com.deitel.doodlz;
4
5 import android.content.pm.ActivityInfo;
6 import android.content.res.Configuration;
7 import android.os.Bundle;
8 import android.support.v7.app.AppCompatActivity;
9 import android.support.v7.widget.Toolbar;
10
```

```
11 public class MainActivity extends AppCompatActivity {
12    // Настройка ориентации экрана для приложения
13    @Override
14    protected void onCreate(Bundle savedInstanceState) {
15       super.onCreate(savedInstanceState);
16       setContentView(R.layout.activity_main);
17       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
18       setSupportActionBar(toolbar);
19
20       // Определение размера экрана
21       int screenSize =
22          getResources().getConfiguration().screenLayout &
23             Configuration.SCREENLAYOUT_SIZE_MASK;
24
25       // Альбомная ориентация только для сверхбольших планшетов
26       if (screenSize ==  Configuration.SCREENLAYOUT_SIZE_XLARGE)
27          setRequestedOrientation(
28             ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
29       else
30          setRequestedOrientation(
31             ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
32    }
33 }
```

5.7. Класс MainActivityFragment

Класс `MainActivityFragment` (разделы 5.7.1–5.7.10) отображает представление `MainActivityView` (раздел 5.8), управляет командами на панели приложения и в меню, а также обработкой событий для функции стирания рисунка встряхиванием устройства.

5.7.1. Команды package, import и переменные экземпляров

В разделе 5.3 рассматриваются новые классы и интерфейсы, используемые классом `MainActivityFragment`. Эти классы и интерфейсы выделены в листинге 5.5. Переменная `doodleView` класса `DoodleView` (строка 24) представляет область рисования. Вещественные переменные, объявляемые в строках 25–27, используются для пересчета изменений в ускорении устройства для выявления события встряхивания (чтобы спросить, действительно ли пользователь хочет стереть рисунок). В строке 28 определяется логическая переменная, по умолчанию содержащая `false`; она будет использоваться для обозначения присутствия диалогового окна на экране. Таким образом предотвращается одновременное появление нескольких диалоговых окон — например, если пользователь случайно встряхнет устройство в то время, когда на экране находится диалоговое окно `Choose Color`, диалоговое окно стирания рисунка отображаться *не должно*. Константа в строке 31 используется для того, чтобы малые перемещения не

интерпретировались как встряхивание, — мы подобрали эту константу методом проб и ошибок на нескольких разных устройствах. Константа в строке 35 используется для идентификации запроса на получение разрешения, необходимого для сохранения рисунка.

Листинг 5.5. Класс MainActivityFragment: команды package, import и переменные экземпляров

```
 1 // MainActivityFragment.java
 2 // Фрагмент, в котором отображается DoodleView
 3 package com.deitel.doodlz;
 4
 5 import android.Manifest;
 6 import android.app.AlertDialog;
 7 import android.content.Context;
 8 import android.content.DialogInterface;
 9 import android.content.pm.PackageManager;
10 import android.hardware.Sensor;
11 import android.hardware.SensorEvent;
12 import android.hardware.SensorEventListener;
13 import android.hardware.SensorManager;
14 import android.os.Bundle;
15 import android.support.v4.app.Fragment;
16 import android.view.LayoutInflater;
17 import android.view.Menu;
18 import android.view.MenuInflater;
19 import android.view.MenuItem;
20 import android.view.View;
21 import android.view.ViewGroup;
22
23 public class MainActivityFragment extends Fragment {
24    private DoodleView doodleView; // Обработка событий касания и рисования
25    private float acceleration;
26    private float currentAcceleration;
27    private float lastAcceleration;
28    private boolean dialogOnScreen = false;
29
30    // Используется для обнаружения встряхивания устройства
31    private static final int ACCELERATION_THRESHOLD = 100000;
32
33    // Используется для идентификации запросов на использование
34    // внешнего хранилища; необходимо для работы функции сохранения
35    private static final int SAVE_IMAGE_PERMISSION_REQUEST_CODE = 1;
36
```

5.7.2. Переопределение метода onCreateView

Метод `onCreateView` (листинг 5.6) заполняет графический интерфейс `MainActivityFragment` и инициализирует переменные экземпляров. Фрагмент, как и активность, может размещать элементы на панели действий приложения и в меню. Для этого он должен вызвать свой метод `setHasOptionsMenu`

с аргументом `true`. Если родительская активность также размещает свои команды в меню, то элементы активности и фрагмента будут размещаться на панели действий и в меню (в зависимости от их настроек).

Листинг 5.6. Переопределение метода onCreateView фрагмента

```
37    // Вызывается при создании представления фрагмента
38    @Override
39    public View onCreateView(LayoutInflater inflater, ViewGroup container,
40       Bundle savedInstanceState) {
41       super.onCreateView(inflater, container, savedInstanceState);
42       View view =
43          inflater.inflate(R.layout.fragment_main, container, false);
44
45          setHasOptionsMenu(true); // у фрагмента имеются команды меню
46
47       // Получение ссылки на DoodleView
48       doodleView = (DoodleView) view.findViewById(R.id.doodleView);
49
50       // Инициализация параметров ускорения
51       acceleration = 0.00f;
52       currentAcceleration = SensorManager.GRAVITY_EARTH;
53       lastAcceleration = SensorManager.GRAVITY_EARTH;
54       return view;
55    }
56
```

Строка 48 получает ссылку на `DoodleView`, после чего в строках 51–53 инициализируются переменные экземпляров, которые используются при вычислении изменений ускорения и помогают определить, встряхнул ли пользователь устройство. Изначально переменным `currentAcceleration` и `lastAcceleration` присваивается константа `GRAVITY_EARTH` класса `SensorManager`, представляющая ускорение, обусловленное земным притяжением. `SensorManager` также предоставляет константы для других планет Солнечной системы и ряд других занимательных значений; с ними можно ознакомиться по адресу

http://developer.android.com/reference/android/hardware/SensorManager.html

5.7.3. Методы onResume и enableAccelerometerListening

Прослушивание показаний акселерометра должно быть включено только в то время, когда `MainActivityFragment` находится на экране. По этой причине мы переопределяем метод жизненного цикла фрагмента `onResume` (листинг 5.7, строки 58–62), который вызывает метод `enableAccelerometerListening` (строки 65–75) для включения прослушивания событий акселерометра. Объект `SensorManager` используется для регистрации слушателей событий акселерометра.

Метод `enableAccelerometerListening` сначала использует метод `getSystemService` активности для получения системного объекта `SensorManager`, через который приложение взаимодействует с датчиками устройства. Затем в строках 72–74 приложение регистрируется для получения событий акселерометра, для чего используется метод `registerListener` объекта `SensorManager`, получающий три аргумента.

- Объект `SensorEventListener`, реагирующий на события (см. раздел 5.7.5).
- Объект `Sensor`, представляющий тип данных, которые приложение желает получать от датчиков, — для его получения вызывается метод `getDefaultSensor` объекта `SensorManager` с передачей константы типа датчика (`Sensor.TYPE_ACCELEROMETER` в нашем приложении).
- Частота получения событий датчика приложением. Константа `SENSOR_DELAY_NORMAL` определяет частоту событий по умолчанию — большая частота позволит получать более точные данные, но она также требует больших затрат процессорного времени и заряда батареи.

Листинг 5.7. Методы onResume и enableAccelerometerListening

```
57    // Начало прослушивания событий датчика
58    @Override
59    public void onResume() {
60       super.onResume();
61       enableAccelerometerListening(); // Прослушивание события встряхивания
62    }
63
64    // Включение прослушивания событий акселерометра
65    private void enableAccelerometerListening() {
66       // Получение объекта SensorManager
67       SensorManager sensorManager =
68       (SensorManager) getActivity().getSystemService(
69          Context.SENSOR_SERVICE);
70
71       // Регистрация для прослушивания событий акселерометра
72       sensorManager.registerListener(sensorEventListener,
73          sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
74          SensorManager.SENSOR_DELAY_NORMAL);
75    }
76
```

5.7.4. Методы onPause и disableAccelerometerListening

Чтобы отключить прослушивание событий акселерометра на то время, когда `MainActivityFragment` не находится на экране, мы переопределяем метод `onPause` жизненного цикла фрагмента (листинг 5.8; строки 78–82), вызывая в нем метод `disableAccelerometerListening` (строки 85–94). Метод

disableAccelerometerListening вызывает метод unregisterListener класса SensorManager для прекращения прослушивания событий акселерометра.

Листинг 5.8. Методы onPause и disableAccelerometerListening

```
77    // Прекращение прослушивания событий акселерометра
78    @Override
79    public void onPause() {
80       super.onPause();
81       disableAccelerometerListening(); // Прекращение прослушивания
82    }
83
84    // Отказ от прослушивания событий акселерометра
85    private void disableAccelerometerListening() {
86       // Получение объекта SensorManager
87       SensorManager sensorManager =
88          (SensorManager) getActivity().getSystemService(
89             Context.SENSOR_SERVICE);
90
91       // Прекращение прослушивания событий акселерометра
92       sensorManager.unregisterListener(sensorEventListener,
93       sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER));
94    }
95
```

5.7.5. Анонимный внутренний класс для обработки событий акселерометра

В листинге 5.9 переопределяется метод onSensorChanged класса SensorEventListener (строки 100–123), обрабатывающий события акселерометра. При перемещении устройства пользователем этот метод определяет, следует ли рассматривать данное перемещение как встряхивание. Если проверка дает положительный результат, в строке 121 вызывается метод confirmErase (раздел 5.7.6), который отображает фрагмент EraseImageDialogFragment (раздел 5.11) для подтверждения удаления рисунка. Интерфейс SensorEventListener также включает метод onAccuracyChanged (строка 127) — так как в нашем приложении этот метод не используется, его тело остается пустым.

Листинг 5.9. Анонимный внутренний класс, реализующий интерфейс SensorEventListener для обработки событий акселерометра

```
 96    // Обработчик для событий акселерометра
 97    private final SensorEventListener sensorEventListener =
 98       new SensorEventListener() {
 99          // Проверка встряхивания по показаниям акселерометра
100          @Override
101            public void onSensorChanged(SensorEvent event) {
102             // На экране не должно быть других диалоговых окон
103             if (!dialogOnScreen) {
```

```
104                // Получить значения x, y и z для SensorEvent
105                float x = event.values[0] ;
106                float y = event.values[1];
107                float z = event.values[2];
108
109                // Сохранить предыдущие данные ускорения
110                lastAcceleration = currentAcceleration;
111
112                // Вычислить текущее ускорение
113                currentAcceleration = x * x + y * y + z * z;
114
115                // Вычислить изменение ускорения
116                acceleration = currentAcceleration *
117                   (currentAcceleration - lastAcceleration);
118
119                // Если изменение превышает заданный порог
120                if (acceleration > ACCELERATION_THRESHOLD)
121                   confirmErase();
122             }
123          }
124
125          // Обязательный метод интерфейса SensorEventListener
126          @Override
127            public void onAccuracyChanged(Sensor sensor, int accuracy){}
128       };
129
```

Пользователь может случайно встряхнуть устройство даже в тех случаях, если диалоговые окна отображаются на экране. По этой причине метод onSensorChanged сначала проверяет, отображается ли диалоговое окно (строка 103). Эта проверка гарантирует отсутствие на экране других диалоговых окон, что особенно важно в случае, если события датчика происходят в другом потоке выполнения. Без такой проверки окно подтверждения стирания изображения могло бы появиться одновременно с отображением на экране другого диалогового окна.

Параметр SensorEvent содержит информацию о происшедшем изменении состояния датчика. Предназначенный для хранения информации о событиях акселерометра массив значений этого параметра включает три элемента, которые представляют ускорение (в м/с^2) в направлениях *x* (влево/вправо), *y* (вверх/вниз) и *z* (вперед/назад). Описание и диаграмму системы координат, используемой API SensorEvent, можно найти на сайте:

http://developer.android.com/reference/android/hardware/SensorEvent.html

По ссылке вы также познакомитесь с интерпретацией значений *x*, *y* и *z* SensorEvent для различных датчиков.

Компоненты ускорения сохраняются в строках 105–107. Очень важно быстро обрабатывать события датчика или копировать данные событий (как сделано в нашем приложении), потому что массив значений датчика *заново* используется

для каждого события. Последнее значение `currentAcceleration` сохраняется в строке 110. В строке 113 суммируются квадраты компонентов ускорения *x*, *y* и *z*, и сумма сохраняется в переменной `currentAcceleration`. По значениям `currentAcceleration` и `lastAcceleration` вычисляется значение `acceleration`, которое сравнивается с константой `ACCELERATION_THRESHOLD`. Если значение превышает порог, это означает, что пользователь перемещает устройство достаточно быстро и перемещение может интерпретироваться как встряхивание. В этом случае вызывается метод `confirmErase`.

5.7.6. Метод confirmErase

Метод `confirmErase` (листинг 5.10) просто создает объект `EraseImageDialogFragment` (раздел 5.11) и использует метод `show` класса `DialogFragment` для его отображения.

Листинг 5.10. Метод confirmErase отображает фрагмент EraseImageDialogFragment

```
130    // Подтверждение стирания рисунка
131    private void confirmErase() {
132       EraseImageDialogFragment fragment = new EraseImageDialogFragment();
133       fragment.show(getFragmentManager(), "erase dialog");
134    }
135
```

5.7.7. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected

В листинге 5.11 переопределяется метод `onCreateOptionsMenu` класса `Fragment` (строки 137–141), добавляющий команды в аргумент `Menu` с использованием аргумента `MenuInflater`. Когда пользователь выбирает команду меню, метод `onOptionsItemSelected` фрагмента (строки 144–169) обрабатывает его действие.

Листинг 5.11. Переопределение методов onCreateOptionsMenu и onOptionsItemSelected фрагмента

```
136    // Отображение команд меню фрагмента
137    @Override
138    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
139       super.onCreateOptionsMenu(menu, inflater);
140       inflater.inflate(R.menu.doodle_fragment_menu, menu);
141    }
142
143    // Обработка выбора команд меню
144    @Override
145    public boolean onOptionsItemSelected(MenuItem item) {
```

```
146       // Выбор в зависимости от идентификатора MenuItem
147       switch (item.getItemId()) {
148          case R.id.color:
149             ColorDialogFragment colorDialog = new ColorDialogFragment();
150             colorDialog.show(getFragmentManager(), "color dialog");
151             return true; // Событие меню обработано
152          case R.id.line_width:
153             LineWidthDialogFragment widthDialog =
154                new LineWidthDialogFragment();
155             widthDialog.show(getFragmentManager(), "line width dialog");
156             return true; // Событие меню обработано
157          case R.id.delete_drawing:
158             confirmErase(); // Получить подтверждение перед стиранием
159             return true; // Событие меню обработано
160          case R.id.save:
161             saveImage(); // Проверить разрешение и сохранить рисунок
162             return true; // Событие меню обработано
163          case R.id.print:
164             doodleView.printImage(); // Напечатать текущий рисунок
165             return true; // Событие меню обработано
166       }
167
168       return super.onOptionsItemSelected(item);
169    }
170
```

Метод `getItemID` класса `MenuItem` (строка 147) используется для получения идентификатора ресурса выбранной команды меню, после чего приложение выполняет различные действия в зависимости от выбора пользователя. Эти действия перечислены ниже.

- `R.id.color` — в строках 157–158 создается и отображается фрагмент ColorDialogFragment (раздел 5.9), с помощью которого пользователь выбирает новый цвет линии рисунка.
- `R.id.lineWidth` — в строках 153–155 создается и отображается фрагмент `LineWidthDialogFragment` (раздел 5.10), с помощью которого пользователь выбирает новую толщину линии.
- `R.id.delete_drawing` — в строке 158 вызывается метод `confirmErase` (листинг 5.7.6), который отображает фрагмент `EraseImageDialogFragment` (раздел 5.11) и предлагает пользователю подтвердить стирание изображения.
- `R.id.save` — в строке 161 вызывается метод `saveImage`, сохраняющий текущий рисунок в галерее `Photos` устройства после проверки, а при необходимости и в запросе разрешения записи во внешнее хранилище.
- `R.id.print` — в строке 164 для представления `doodleView` вызывается метод `printImage`, позволяющий сохранить изображение в формате PDF или вывести его на печать.

5.7.8. Метод saveImage

Метод `saveImage` (листинг 5.12) вызывается методом `onOptionsItemSelected`, когда пользователь выбирает команду `Save` в меню команд. Метод `saveImage` реализует один из аспектов новой модели разрешений Android 6.0, который перед выполнением операции проверяет, имеет ли приложение необходимое разрешение. Если разрешение отсутствует, приложение запрашивает его у пользователя, прежде чем пытаться выполнять операцию.

Строки 176–178 проверяют, имеет ли приложение разрешение для записи во внешнее хранилище перед сохранением изображения. Если приложение не обладает разрешением `android.permission.WRITE_EXTERNAL_STORAGE`, строки 181–182 используют встроенный метод `shouldShowRequestPermissionRationale` для определения того, нужно ли вывести объяснение необходимости вывода разрешения. Метод возвращает `true`, если такое пояснение может быть полезным для пользователя — например, если пользователь ранее отклонил запрос на получение разрешения. В таком случае строки 183–203 создают и отображают диалоговое окно с объяснением.

Когда пользователь нажимает кнопку `OK` в диалоговом окне, в строках 195–197 запрашивается разрешение `android.permission.WRITE_EXTERNAL_STORAGE` перед использованием унаследованного метода `requestPermissions`. Если объяснение не требуется — например, если разрешение понадобилось приложению впервые, — то команды в строках 207–209 немедленно запрашивают разрешение.

Листинг 5.12. Метод saveImage

```
171    // При необходимости метод запрашивает разрешение
172    // или сохраняет изображение, если разрешение уже имеется
173    private void saveImage() {
174       // Проверить, есть ли у приложения разрешение,
175       // необходимое для сохранения
176       if (getContext().checkSelfPermission(
177          Manifest.permission.WRITE_EXTERNAL_STORAGE) !=
178          PackageManager.PERMISSION_GRANTED) {
179
180          // Объяснить, почему понадобилось разрешение
181          if (shouldShowRequestPermissionRationale(
182              Manifest.permission.WRITE_EXTERNAL_STORAGE)){
183             AlertDialog.Builder builder =
184                new AlertDialog.Builder(getActivity());
185
186             // Назначить сообщение AlertDialog
187             builder.setMessage(R.string.permission_explanation);
188
189             // Добавить кнопку OK в диалоговое окно
190             builder.setPositiveButton( android.R.string.ok,
```

```
191                new DialogInterface.OnClickListener() {
192                   @Override
193                   public void onClick(DialogInterface dialog, int which) {
194                      // Запросить разрешение
195                      requestPermissions(new String[]{
196                         Manifest.permission.WRITE_EXTERNAL_STORAGE},
197                         SAVE_IMAGE_PERMISSION_REQUEST_CODE);
198                   }
199                }
200             );
201
202             // Отображение диалогового окна
203             builder.create().show();
204          }
205          else {
206             // Запросить разрешение
207             requestPermissions(
208                new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
209                SAVE_IMAGE_PERMISSION_REQUEST_CODE);
210          }
211       }
212       else { // Если разрешение уже имеет разрешение для записи
213          doodleView.saveImage(); // Сохранить изображение
214       }
215    }
216
```

Метод `requestPermissions` получает строковый массив разрешений, запрашиваемых приложением, и целое число (`SAVE_IMAGE_PERMISSION_REQUEST_CODE`), которое используется для идентификации запроса на получение разрешения. При вызове `requestPermissions` Android выводит диалоговое окно (рис. 5.12), в котором пользователь может запретить (`DENY`) или разрешить (`ALLOW`) выполнение операции. Система вызывает метод `onRequestPermissionsResult` (раздел 5.7.9) для обработки ответа пользователя. Если приложение уже обладает необходимыми разрешениями, то строка 213 вызывает метод `saveImage` класса `DoodleView` для сохранения изображения.

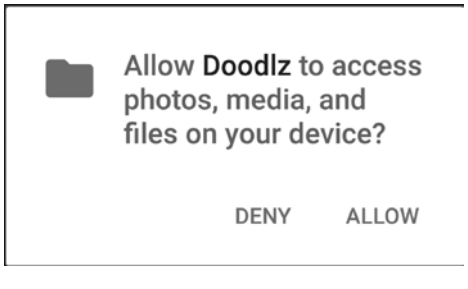


Рис. 5.12. Диалоговое окно для отказа или предоставления разрешения записи во внешнее хранилище данных

5.7.9. Переопределенный метод onRequestPermissionsResult

Метод `onRequestPermissionsResult` (листинг 5.13) получает код `requestCode` для выданного запроса и передает его конструкции `switch` в строках 224–229, выполняющей соответствующий код для запроса. В нашем приложении используется только один запрос разрешения, поэтому конструкция `switch` содержит всего одну секцию `case` с константой `SAVE_IMAGE_PERMISSION_REQUEST_CODE`. Для приложений, требующих нескольких разрешений, следует задать уникальное значение для каждого разрешения при вызове метода `requestPermissions`. Строка 226 проверяет, предоставил ли пользователь разрешение для записи во внешнее хранилище. В таком случае строка 227 вызывает метод `saveImage` класса `DoodleView` для сохранения изображения.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 5.1

Если при попытке сохранить изображение пользователь не дает необходимое разрешение, то при следующей попытке сохранения в диалоговое окно будет включен флажок «Никогда не спрашивать снова». Если пользователь установит этот флажок и отклонит разрешение, то при будущих попытках сохранения метод onRequestPermissionResult будет вызываться с аргументом PackageManager.PERMISSION_DENIED. Приложение должно обработать эту ситуацию и сообщить пользователю, как изменить разрешения приложения в приложении Settings.

Листинг 5.13. Переопределенный метод onRequestPermissionsResult

```
217    // Вызывается системой, когда пользователь предоставляет
218    // или отклоняет разрешение для сохранения изображения
219    @Override
220    public void onRequestPermissionsResult(int requestCode,
221       String[] permissions, int[] grantResults) {
222       // switch выбирает действие в зависимости от того,
223       // какое разрешение было запрошено
224       switch (requestCode) {
225          case SAVE_IMAGE_PERMISSION_REQUEST_CODE:
226             if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
227                doodleView.saveImage(); // Сохранить изображение
228             return;
229       }
230    }
231
```

5.7.10. Методы getDoodleView и setDialogOnScreen

Методы `getDoodleView` и `setDialogOnScreen` (листинг 5.14) вызываются методами субклассов `DialogFragment`, используемыми в нашем приложении. Метод

getDoodleView возвращает ссылку на объект DoodleView текущего фрагмента, чтобы реализация DialogFragment могла назначить цвет и толщину линии или стереть изображение. Метод setDialogOnScreen вызывается методами жизненного цикла фрагмента субклассов DialogFragment приложения; он устанавливает флаг присутствия диалогового окна на экране.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 5.2

Фрагменты в этом приложении взаимодействуют друг с другом напрямую. Мы выбрали эту схему с сильным связыванием для простоты. Обычно взаимодействия между фрагментами в приложении проходят под управлением родительской активности. Чтобы передать данные фрагменту, активность предоставляет объект Bundle с аргументами. Каждый класс фрагмента обычно предоставляет интерфейс из методов обратного вызова, реализуемых активностью. Когда фрагменту потребуется оповестить родительскую активность об изменении состояния, фрагмент вызывает подходящий метод обратного вызова. Этот механизм расширяет возможности повторного использования фрагментов в разных активностях. Его применение будет продемонстрировано в приложении Address Book главы 9.

Листинг 5.14. Методы getDoodleView и setDialogOnScreen

```
232    // Метод возвращает объект DoodleView
233    public DoodleView getDoodleView() {
234       return doodleView;
235    }
236
237    // Проверяет, отображается ли диалоговое окно
238    public void setDialogOnScreen(boolean visible) {
239       dialogOnScreen = visible;
240    }
241 }
```

5.8. Класс DoodleView

Класс DoodleView (разделы 5.8.1–5.8.12) обрабатывает касания пользователя и рисует соответствующие линии.

5.8.1. Команды package и import

В листинге 5.15 приведены команды package и import, а также поля класса DoodleView. Новые классы и интерфейсы выделены жирным шрифтом. Многие из них описаны в разделе 5.3, а остальные будут рассматриваться по мере их использования в классе DoodleView.

Листинг 5.15. Команды package и import класса DoodleView

```
 1 // DoodleView.java
 2 // Главное представление приложения Doodlz.
 3 package com.deitel.doodlz;
 4
 5 import android.content.Context;
 6 import android.graphics.Bitmap;
 7 import android.graphics.Canvas;
 8 import android.graphics.Color;
 9 import android.graphics.Paint;
10 import android.graphics.Path;
11 import android.graphics.Point;
12 import android.provider.MediaStore;
13 import android.support.v4.print.PrintHelper;
14 import android.util.AttributeSet;
15 import android.view.Gravity;
16 import android.view.MotionEvent;
17 import android.view.View;
18 import android.widget.Toast;
19
20 import java.util.HashMap;
21 import java.util.Map;
22
```

5.8.2. Статические переменные и переменные экземпляров DoodleView

Переменные класса `DoodleView` (листинг 5.16) используются для управления данными набора линий, нарисованных пользователем, и для прорисовки этих линий. В строке 34 создается объект `pathMap`, связывающий идентификатор каждого пальца (называемый *указателем*) с объектом `Path` для рисуемых линий. В строке 35 создается объект `previousPointMap`, в котором хранится последняя точка каждого пальца, — с перемещением пальца проводится линия от текущей точки к предыдущей. Другие поля будут рассматриваться по мере их использования в классе `DoodleView`.

Листинг 5.16. Статические переменные и переменные экземпляров класса DoodleView

```
23 // пользовательское представление, на котором рисует пользователь
24 public class DoodleView extends View {
25    // Смещение, необходимое для продолжения рисования
26    private static final float TOUCH_TOLERANCE = 10;
27
28    private Bitmap bitmap; // Область рисования для вывода или сохранения
29    private Canvas bitmapCanvas; // Используется для рисования на Bitmap
30    private final Paint paintScreen; // Используется для вывода Bitmap на экран
31    private final Paint paintLine; // Используется для рисования линий на Bitmap
32
```

```
33    // Данные нарисованных контуров Path и содержащихся в них точек
34    private final Map<Integer, Path> pathMap = new HashMap<>();
35    private final Map<Integer, Point> previousPointMap =  new HashMap<>();
36
```

5.8.3. Конструктор doodleview

Конструктор (листинг 5.17) инициализирует некоторые переменные экземпляров класса — две коллекции `Map` инициализируются в их объявлениях в листинге 5.17. Строка 40 создает объект `paintScreen` класса `Paint`, который будет использоваться для рисования на экране, а в строке 43 создается объект `paintLine` класса `Paint`, определяющий настройки линии, которую в настоящий момент рисует пользователь. Параметры `paintLine` задаются в строках 44–48. Метод `setAntiAlias` класса `Paint` вызывается со значением `true` для включения режима сглаживания границ линий. Наконец, вызов метода `setStyle` класса `Paint` назначает объекту `Paint` стиль `Paint.Style.STROKE`. Выбирая стиль `STROKE`, `FILL` или `FILL_AND_STROKE`, можно выбрать режим рисования линий, заполненных фигур без контура и заполненных фигур с контуром соответственно. По умолчанию используется значение `Paint.Style.FILL`. Толщина линии задается методом `setStrokeWidth` объекта `Paint`. При этом приложению назначается принятая по умолчанию толщина линии в 5 пикселов. Мы также используем метод `setStrokeCap` класса `Paint` для вывода линий с закругленными концами (`Paint.Cap.ROUND`).

Листинг 5.17. Конструктор DoodleView

```
37    // Конструктор DoodleView инициализирует объект DoodleView
38    public DoodleView(Context context, AttributeSet attrs) {
39       super(context, attrs); // Конструктору View передается контекст
40       paintScreen = new Paint(); // Используется для вывода на экран
41
42       // Исходные параметры рисуемых линий
43       paintLine = new Paint();
44       paintLine.setAntiAlias(true); // Сглаживание краев
45       paintLine.setColor(Color.BLACK); // По умолчанию черный цвет
46       paintLine.setStyle(Paint.Style.STROKE); // Сплошная линия
47       paintLine.setStrokeWidth(5); // Толщина линии по умолчанию
48       paintLine.setStrokeCap(Paint.Cap.ROUND); // Закругленные концы
49    }
50
```

5.8.4. Переопределенный метод onSizeChanged

Размер `DoodleView` определяется только после того, как представление будет заполнено и добавлено в иерархию представлений `MainActivity`; следовательно, определить размер объекта `Bitmap`, используемого при рисовании, в методе

onCreate не удастся. По этой причине мы переопределяем метод onSizeChanged класса View (листинг 5.18), который вызывается при изменении размера DoodleView, например при добавлении в иерархию представлений активности или при повороте устройства. В нашем приложении метод onSizeChanged вызывается только при добавлении в иерархию представлений активности Doodlz, потому что приложение всегда выполняется в портретном режиме на телефонах и малых планшетах и в альбомном режиме на больших планшетах.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 5.3

В приложениях, которые поддерживают как портретную, так и альбомную ориентацию, метод onSizeChanged вызывается при каждом повороте устройства. В данном приложении это приведет к тому, что каждый раз будет создаваться новый объект Bitmap. При замене Bitmap следует вызвать метод recycle для предыдущего экземпляра Bitmap, чтобы освободить занимаемые ресурсы.

Листинг 5.18. Переопределенный метод onSizeChanged

```
51    // Создание объектов Bitmap и Canvas на основании размеров View
52    @Override
53    public void onSizeChanged(int w, int h, int oldW, int oldH) {
54       bitmap = Bitmap.createBitmap(getWidth(), getHeight(),
55          Bitmap.Config.ARGB_8888);
56       bitmapCanvas = new Canvas(bitmap);
57       bitmap.eraseColor(Color.WHITE); // Bitmap стирается белым цветом
58    }
59
```

Статический метод createBitmap класса Bitmap создает объект Bitmap с заданной шириной и высотой — в качестве размеров Bitmap используются ширина и высота DoodleView. В последнем аргументе createBitmap передается *кодировка* Bitmap, определяющая, в каком формате хранится каждый пиксел Bitmap. Константа Bitmap.Config.ARGB_8888 означает, что цвет каждого пиксела хранится в четырех байтах (по одному байту для альфа-канала, красной, зеленой и синей составляющих). Затем мы создаем новый объект Canvas, который используется для рисования прямо на Bitmap. Наконец, метод eraseColor класса Bitmap заполняет Bitmap белыми пикселами (по умолчанию Bitmap использует черный цвет фона).

5.8.5. Методы clear, setDrawingColor, getDrawingColor, setLineWidth и getLineWidth

В листинге 5.19 определяются методы clear (строки 61–66), setDrawingColor (строки 69–71), getDrawingColor (строки 74–76), setLineWidth (строки 79–81) и getLineWidth (строки 84–86) класса DoodleView, вызываемые из MainActivityFragment. Метод clear, используемый в EraseImageDialogFragment,

очищает коллекции pathMap и previousPointMap, стирает Bitmap, заполняя все пикселы белым цветом, а затем вызывает унаследованный от View метод invalidate, чтобы сообщить о необходимости перерисовки View. Затем система автоматически определяет, когда должен быть вызван метод onDraw класса View. Метод setDrawingColor изменяет текущий цвет линий, назначая цвет объекта paintLine. Метод setColor класса Paint получает значение int, представляющее новый цвет в формате ARGB. Метод getDrawingColor возвращает текущий цвет, который используется в ColorDialogFragment. Метод setLineWidth задает толщину линии paintLine в пикселах. Метод getLineWidth возвращает текущую толщину линии, которая используется в LineWidthDialogFragment.

Листинг 5.19. Методы clear, setDrawingColor, getDrawingColor, setLineWidth и getLineWidth класса DoodleView

```
60    // Стирание рисунка
61    public void clear() {
62       pathMap.clear(); // Удалить все контуры
63       previousPointMap.clear(); // Удалить все предыдущие точки
64       bitmap.eraseColor(Color.WHITE); // Очистка изображения
65       invalidate(); // Перерисовать изображение
66    }
67
68    // Назначение цвета рисуемой линии
69    public void setDrawingColor(int color) {
70       paintLine.setColor(color);
71    }
72
73    // Получение цвета рисуемой линии
74    public int getDrawingColor() {
75       return paintLine.getColor();
76    }
77
78    // Назначение толщины рисуемой линии
79    public void setLineWidth(int width) {
80       paintLine.setStrokeWidth(width);
81    }
82
83    // Получение толщины рисуемой линии
84    public int getLineWidth() {
85       return (int) paintLine.getStrokeWidth();
86    }
87
```

5.8.6. Переопределенный метод onDraw

Когда представлению потребуется перерисовка, вызывается его метод onDraw. В листинге 5.20 метод onDraw переопределяется для отображения объекта Bitmap (содержащего изображение) на DoodleView, для чего вызывается метод drawBitmap аргумента Canvas. В первом аргументе передается объект Bitmap,

в двух других — координаты *x* и *y* левого верхнего угла `Bitmap` в представлении и в последнем — объект `Paint` с характеристиками графического вывода. Далее цикл в строках 95–96 перебирает и отображает объекты `Path` текущего рисунка. Для каждого целочисленного ключа в `pathMap` соответствующий объект `Path` передается методу `drawPath` объекта `Canvas` для прорисовки с использованием объекта `paintLine`, определяющего толщину и цвет линии.

Листинг 5.20. Переопределенный метод onDraw

```
88    // Перерисовка при обновлении DoodleView на экране
89    @Override
90       protected void onDraw(Canvas canvas) {
91       // Перерисовка фона
92       canvas.drawBitmap(bitmap, 0, 0, paintScreen);
93
94       // Для каждой выводимой линии
95       for (Integer key : pathMap.keySet())
96          canvas.drawPath(pathMap.get(key), paintLine); // Рисование линии
97    }
98
```

5.8.7. Переопределенный метод onTouchEvent

Метод `onTouchEvent` (листинг 5.21) вызывается при получении представлением события касания. Android поддерживает *многоточечные* касания, то есть прикосновение к экрану сразу несколькими пальцами. В любой момент пользователь может приложить к экрану другие пальцы или отвести их от экрана. По этой причине каждому пальцу присваивается уникальный *идентификатор*, который идентифицирует его в разных событиях касания. Мы используем его для идентификации объектов `Path`, представляющих каждую линию, рисуемую в настоящий момент. Объекты `Path` хранятся в коллекции `pathMap`.

Листинг 5.21. Переопределенный метод onTouchEvent

```
 99    // Обработка события касания
100    @Override
101      public boolean onTouchEvent(MotionEvent event) {
102       int action = event.getActionMasked(); // Тип события
103       int actionIndex = event.getActionIndex(); // Указатель (палец)
104
105       // Что происходит: начало касания, конец, перемещение?
106       if (action == MotionEvent.ACTION_DOWN ||
107            action == MotionEvent.ACTION_POINTER_DOWN) {
108          touchStarted( event.getX(actionIndex), event.getY(actionIndex),
109             event.getPointerId(actionIndex));
110       }
111       else if (action == MotionEvent.ACTION_UP ||
112                action == MotionEvent.ACTION_POINTER_UP) {
113          touchEnded(event.getPointerId(actionIndex));
```

```
114       }
115       else {
116          touchMoved(event);
117       }
118
119       invalidate(); // Перерисовка
120       return true;
121    }
122
```

Метод `getActionMasked` (строка 102) класса `MotionEvent` возвращает значение `int` — признак типа события `MotionEvent`, который может использоваться в сочетании с константами класса `MotionEvent` для определения того, как должно обрабатываться каждое событие. Метод `getActionIndex` (строка 103) класса `MotionEvent` возвращает целочисленный индекс пальца, сгенерировавшего событие. Этот индекс не является уникальным идентификатором пальца — он всего лишь определяет, в какой позиции объекта `MotionEvent` хранится информация от этого пальца. Чтобы получить уникальный идентификатор пальца, сохраняющийся между событиями `MotionEvent` до того, как пользователь отведет палец от экрана, мы воспользуемся методом `getPointerID` (строки 109 и 113) класса `MotionEvent`; этот метод получает индекс пальца в аргументе.

Если обнаружено действие `MotionEvent.ACTION_DOWN` или `MotionEvent.ACTION_POINTER_ DOWN` (строки 106–107), значит, пользователь коснулся экрана новым пальцем. Первый палец, коснувшийся экрана, генерирует событие `MotionEvent.ACTION_DOWN`, а все остальные пальцы генерируют события `MotionEvent.ACTION_POINTER_DOWN`. В таких случаях вызывается метод `touchStarted` (листинг 5.22) для хранения исходных координат касания. Если обнаружено действие `MotionEvent.ACTION_UP` или `MotionEvent.ACTION_POINTER_UP`, значит, пользователь отвел палец от экрана, поэтому вызывается метод `touchEnded` (листинг 5.24) для рисования завершенного контура `Path` на растровом изображении. Для остальных событий касания вызывается метод `touchMoved` (листинг 5.23) для рисования линий. После обработки события строка 119 вызывает унаследованный от `View` метод `invalidate` для перерисовки экрана, а строка 120 возвращает *true* — признак того, что событие было обработано.

5.8.8. Метод touchStarted

Метод `touchStarted` (листинг 5.22) вызывается при первом соприкосновении пальца с экраном. В аргументах передаются координаты касания и идентификатор пальца. Если объект `Path` для заданного идентификатора уже существует (строка 129), мы вызываем метод `reset` объекта `Path` для стирания всех существующих точек, чтобы заново использовать `Path` для нового контура. В противном случае создается новый объект `Path`, он добавляется в `pathMap`, а новый объект Point добавляется в `previousPointMap`. В строках 142–144 вызывается

метод moveTo объекта Path для задания начальных координат Path и задания значений x и y нового объекта Point.

Листинг 5.22. Метод touchStarted класса DoodleView

```
123    // Вызывается при касании экрана
124    private void touchStarted(float x, float y, int lineID) {
125       Path path; // Для хранения контура с заданным идентификатором
126       Point point; // Для хранения последней точки в контуре
127
128       // Если для lineID уже существует объект Path
129       if (pathMap.containsKey(lineID)) {
130          path = pathMap.get(lineID); // Получение Path
131          path.reset(); // Очистка Path с началом нового касания
132          point = previousPointMap.get(lineID); // Последняя точка Path
133       }
134       else {
135          path = new Path();
136          pathMap.put(lineID, path); // Добавление Path в Map
137          point = new Point(); // Создание нового объекта Point
138          previousPointMap.put(lineID, point); // Добавление Point в Map
139       }
140
141       // Переход к координатам касания
142       path.moveTo(x, y);
143       point.x = (int) x;
144       point.y = (int) y;
145    }
146
```

5.8.9. Метод touchMoved

Метод touchMoved (листинг 5.23) вызывается при проведении одним или несколькими пальцами по экрану. Системный объект MotionEvent, передаваемый из onTouchEvent, содержит информацию о нескольких перемещениях на экране, если они происходят одновременно. Метод getPointerCount класса MotionEvent (строка 150) возвращает количество касаний, описываемых объектом MotionEvent. Для каждого касания идентификатор пальца сохраняется в pointerID (строка 152), а соответствующий индекс из MotionEvent (строка 153) сохраняется в pointerIndex. Затем проверяется наличие соответствующего объекта Path в коллекции pathMap (строка 156). Если объект присутствует в коллекции, методы getX и getY класса MotionEvent используются для получения последних координат события перетаскивания для заданного значения pointerIndex. Мы получаем объект Path и последний объект Point для pointerID из соответствующих коллекций HashMap, после чего вычисляем различия между последней и текущей точкой — объект Path должен обновиться только в том случае, если величина перемещения превысила константу TOUCH_TOLERANCE. Мы делаем это, потому что многие устройства обладают высокой чувствительностью и могут генерировать события MotionEvent для малых перемещений, даже когда

пользователь пытается держать палец неподвижно. Если пользователь переместил палец на расстояние, превышающее TOUCH_TOLERANCE, мы используем метод quadTo класса Path (строки 248–249) для добавления геометрической кривой (а конкретно квадратичной кривой Безье) от предыдущей точки к новой, после чего обновляем данные последней точки для этого пальца.

Листинг 5.23. Метод touchMoved класса DoodleView

```
147    // Вызывается при перемещении пальца по экрану
148    private void touchMoved(MotionEvent event) {
149       // Для каждого указателя (пальца) в объекте MotionEvent
150       for (int i = 0; i < event.getPointerCount() ; i++) {
151          // Получить идентификатор и индекс указателя
152          int pointerID = event.getPointerId(i);
153          int pointerIndex = event.findPointerIndex(pointerID);
154
155          // Если существует объект Path, связанный с указателем
156          if (pathMap.containsKey(pointerID)) {
157             // Получить новые координаты для указателя
158             float newX = event.getX(pointerIndex);
159             float newY = event.getY(pointerIndex);
160
161             // Получить объект Path и предыдущий объект Point,
162             // связанный с указателем
163             Path path = pathMap.get(pointerID);
164             Point point = previousPointMap.get(pointerID);
165
166             // Вычислить величину смещения от последнего обновления
167             float deltaX = Math.abs(newX - point.x);
168             float deltaY = Math.abs(newY - point.y);
169
170             // Если расстояние достаточно велико
171             if (deltaX >= TOUCH_TOLERANCE || deltaY >= TOUCH_TOLERANCE) {
172                // Расширение контура до новой точки
173                path.quadTo(point.x, point.y, (newX + point.x) / 2,
174                   (newY + point.y) / 2);
175
176                // Сохранение новых координат
177                point.x = (int) newX;
178                point.y = (int) newY;
179             }
180          }
181       }
182    }
183
```

5.8.10. Метод touchEnded

Метод touchEnded (листинг 5.24) вызывается, когда пользователь отводит палец от экрана. В аргументе метода передается идентификатор пальца (lineID), касание которого только что закончилось. Строка 186 получает соответствующий

объект Path. В строке 187 вызывается drawPath объекта bitmapCanvas для рисования Path на объекте Bitmap с именем bitmap перед очисткой Path. Сброс Path не приводит к стиранию соответствующей нарисованной линии с экрана, потому что эти линии уже были нарисованы на объекте bitmap, отображаемом на экране. Линии, рисуемые пользователем в настоящее время, рисуются поверх bitmap.

Листинг 5.24. Метод touchEnded класса DoodleView

```
184    // Вызывается при завершении касания
185    private void touchEnded(int lineID) {
186       Path path = pathMap.get(lineID); // Получение объекта Path
187       bitmapCanvas.drawPath(path, paintLine); // Рисование на bitmapCanvas
188       path.reset(); // Сброс объекта Path
189    }
190
```

5.8.11. Метод saveImage

Метод saveImage (листинг 5.25) сохраняет текущее изображение, создавая файл в галерее устройства. Строка 194 создает имя файла для изображения, после чего в строках 197–199 изображение сохраняется в приложении Photos, для чего вызывается метод insertImage класса MediaStore.Images.Media. Метод получает четыре аргумента:

- объект ContentResolver, используемый методом для определения места хранения изображения на устройстве;
- объект Bitmap с сохраняемым рисунком;
- имя изображения;
- описание изображения.

Метод insertImage возвращает строку, описывающую местонахождение изображения на устройстве, или null, если сохранить изображение не удалось. Строки 201–217 проверяют, было ли изображение сохранено, и отображают соответствующее временное окно Toast.

Листинг 5.25. Метод saveImage класса DoodleView

```
191    // Сохранение текущего изображения в галерее
192    public void saveImage() {
193       // Имя состоит из префикса "Doodlz" и текущего времени
194       final String name = "Doodlz" + System.currentTimeMillis() + ".jpg";
195
196       // Сохранение изображения в галерее устройства
197       String location = MediaStore.Images.Media.insertImage(
198          getContext().getContentResolver(), bitmap, name,
199          "Doodlz Drawing");
200
```

```
201       if (location != null) {
202          // Вывод сообщения об успешном сохранении
203          Toast message = Toast.makeText(getContext(),
204             R.string.message_saved,
205             Toast.LENGTH_SHORT);
206          message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
207             message.getYOffset() / 2);
208          message.show();
209       }
210       else {
211          // Вывод сообщения об ошибке сохранения
212          Toast message = Toast.makeText(getContext(),
213             R.string.message_error_saving, Toast.LENGTH_SHORT);
214          message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
215             message.getYOffset() / 2);
216          message.show();
217       }
218    }
219
```

5.8.12. Метод printImage

Метод `printImage` (листинг 5.26) использует класс `PrintHelper` из Android Support Library для печати текущего рисунка — эта возможность поддерживается только на устройствах с Android 4.4 и выше. Строка 222 сначала убеждается в том, что поддержка печати доступна на устройстве. Если проверка дает положительный результат, строка 224 создает объект `PrintHelper`. Затем строка 307 задает *режим масштабирования* устройства — значение `PrintHelper.SCALE_MODE_FIT` сообщает, что размеры изображения должны быть подогнаны под размеры печатаемой области листа. Также существует режим `PrintHelper.SCALE_ MODE_FILL`, при котором изображение заполняет лист (возможно, с отсечением части изображения). Наконец, строка 228 вызывает метод `printBitmap` класса `PrintHelper`, передавая в аргументах имя задания печати (используемое принтером для идентификации) и объект `Bitmap` с выводимым изображением. На экране появляется диалоговое окно печати Android, в котором пользователь может выбрать между сохранением изображения в формате PDF на устройстве и печатью на принтере.

Листинг 5.26. Метод printImage класса DoodleView

```
220    // Печать текущего изображения
221    public void printImage() {
222       if (PrintHelper.systemSupportsPrint()) {
223          // Использование класса PrintHelper для печати
224          PrintHelper printHelper = new PrintHelper(getContext());
225
226          // Изображение масштабируется и выводится на печать
227          printHelper.setScaleMode(PrintHelper.SCALE_MODE_FIT);
```

```
228          printHelper.printBitmap("Doodlz Image", bitmap);
229       }
230       else {
231          // Вывод сообщения о том, что система не поддерживает печать
232          Toast message = Toast.makeText(getContext(),
233             R.string.message_error_printing, Toast.LENGTH_SHORT);
234          message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
235             message.getYOffset() / 2);
236          message.show();
237       }
238    }
239 }
```

5.9. Класс ColorDialogFragment

Класс `ColorDialogFragment` (листинги 5.27–5.31) расширяет `DialogFragment` для создания окна `AlertDialog`, в котором определяется цвет линии. Переменные экземпляра класса (строки 18–23) используются для обращения к элементам графического интерфейса, предназначенным для выбора нового цвета, отображения его образца и сохранения цвета в виде 32-разрядного значения `int`, представляющего составляющие цвета в формате ARGB.

Листинг 5.27. Команды package, import и переменные экземпляров класса ColorDialogFragment

```
 1 // ColorDialogFragment.java
 2 // Используется для выбора цвета линии в DoodleView
 3 package com.deitel.doodlz;
 4
 5 import android.app.Activity;
 6 import android.app.AlertDialog;
 7 import android.app.Dialog;
 8 import android.content.DialogInterface;
 9 import android.graphics.Color;
10 import android.os.Bundle;
11 import android.support.v4.app.DialogFragment;
12 import android.view.View;
13 import android.widget.SeekBar;
14 import android.widget.SeekBar.OnSeekBarChangeListener;
15
16 // Класс диалогового окна выбора цвета
17 public class ColorDialogFragment extends DialogFragment {
18    private SeekBar alphaSeekBar;
19    private SeekBar redSeekBar;
20    private SeekBar greenSeekBar;
21    private SeekBar blueSeekBar;
22    private View colorView;
23    private int color;
24
```

5.9.1. Переопределенный метод onCreateDialog класса DialogFragment

Метод `onCreateDialog` (листинг 5.28) заполняет пользовательское представление (строки 31–32), определенное в файле `fragment_color.xml` с графическим интерфейсом выбора цвета, после чего связывает представление с окном `AlertDialog`, вызывая метод `setView` объекта `AlertDialog.Builder` (строка 33). Строки 39–47 получают ссылки на компоненты `SeekBar` и `colorView` диалогового окна. Затем строки 50–53 регистрируют объект `colorChangedListener` (листинг 5.31) слушателем событий компонентов `SeekBar`.

Листинг 5.28. Переопределенный метод onCreateDialog класса DialogFragment

```
25    // Создание и возвращение объекта AlertDialog
26    @Override
27    public Dialog onCreateDialog(Bundle bundle) {
28       // Создание диалогового окна
29       AlertDialog.Builder builder =
30          new AlertDialog.Builder(getActivity());
31       View colorDialogView = getActivity().getLayoutInflater().inflate(
32          R.layout.fragment_color, null);
33       builder.setView(colorDialogView); // Добавление GUI в диалоговое окно
34
35       // Назначение сообщения AlertDialog
36       builder.setTitle(R.string.title_color_dialog);
37
38       // Получение значений SeekBar и назначение слушателей onChange
39       alphaSeekBar = (SeekBar) colorDialogView.findViewById(
40          R.id.alphaSeekBar);
41       redSeekBar = (SeekBar) colorDialogView.findViewById(
42          R.id.redSeekBar);
43       greenSeekBar = (SeekBar) colorDialogView.findViewById(
44          R.id.greenSeekBar);
45       blueSeekBar = (SeekBar) colorDialogView.findViewById(
46          R.id.blueSeekBar);
47       colorView = colorDialogView.findViewById(R.id.colorView);
48
49       // Регистрация слушателей событий SeekBar
50       alphaSeekBar.setOnSeekBarChangeListener(colorChangedListener);
51       redSeekBar.setOnSeekBarChangeListener(colorChangedListener);
52       greenSeekBar.setOnSeekBarChangeListener(colorChangedListener);
53       blueSeekBar.setOnSeekBarChangeListener(colorChangedListener);
54
55       // Использование текущего цвета линии для инициализации
56       final DoodleView doodleView = getDoodleFragment().getDoodleView();
57       color = doodleView.getDrawingColor();
58       alphaSeekBar.setProgress(Color.alpha(color));
59       redSeekBar.setProgress(Color.red(color));
60       greenSeekBar.setProgress(Color.green(color));
61       blueSeekBar.setProgress(Color.blue(color));
62
63       // Добавление кнопки назначения цвета
```

```
64       builder.setPositiveButton(R.string.button_set_color,
65          new DialogInterface.OnClickListener() {
66             public void onClick(DialogInterface dialog, int id) {
67                doodleView.setDrawingColor(color);
68             }
69          }
70       );
71
72       return builder.create(); // Возвращение диалогового окна
73    }
74
```

В строке 56 вызывается метод `getDoodleFragment` (листинг 5.29) для получения ссылки на фрагмент `MainActivityFragment`, после чего вызывается метод `getDoodleView` класса `MainActivityFragment` для получения объекта `DoodleView`. Строки 57–61 получают текущий цвет рисования `DoodleView`, который используется для инициализации компонентов `SeekBar`. Статические методы `alpha`, `red`, `green` и `blue` класса `Color` извлекают значения составляющих ARGB, а метод `setProgress` класса `SeekBar` устанавливает ползунки в нужных позициях. В строках 64–70 позитивная кнопка `AlertDialog` применяет новый цвет рисования `DoodleView`. Строка 72 возвращает объект `AlertDialog` для отображения на экране.

5.9.2. Метод getDoodleFragment

Метод `getDoodleFragment` (листинг 5.29) использует объект `FragmentManager` для получения ссылки на `MainActivityFragment`.

Листинг 5.29. Метод getDoodleFragment

```
75    // Получение ссылки на MainActivityFragment
76    private MainActivityFragment getDoodleFragment() {
77       return (MainActivityFragment) getFragmentManager().findFragmentById(
78          R.id.doodleFragment);
79    }
80
```

5.9.3. Переопределенные методы onAttach и onDetach жизненного цикла фрагмента

При добавлении `ColorDialogFragment` в родительскую активность вызывается метод `onAttach` (листинг 5.30, строки 82–99). Строка 85 получает ссылку на `MainActivityFragment`. Если эта ссылка отлична от `null`, то строка 88 вызывает метод `setDialogOnScreen` класса `MainActivityFragment` для установки флага отображения диалогового окна `Choose Color`. При удалении `ColorDialogFragment` из родительской активности вызывается метод `onDetach` (строки 92–99). В строке 98

вызывается метод `setDialogOnScreen` класса `MainActivityFragment`, указывающий, что диалоговое окно `Choose Color` перестало отображаться на экране.

Листинг 5.30. Переопределенные методы onAttach и onDetach жизненного цикла фрагмента

```
 81    // Сообщает DoodleFragment, что диалоговое окно находится на экране
 82    @Override
 83    public void onAttach(Activity activity) {
 84       super.onAttach(activity);
 85       MainActivityFragment fragment = getDoodleFragment();
 86
 87       if (fragment != null)
 88          fragment.setDialogOnScreen(true);
 89    }
 90
 91    // Сообщает MainActivityFragment, что диалоговое окно не отображается
 92    @Override
 93    public void onDetach() {
 94       super.onDetach();
 95       MainActivityFragment fragment = getDoodleFragment();
 96
 97       if (fragment != null)
 98          fragment.setDialogOnScreen(false);
 99    }
100
```

5.9.4. Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener для обработки событий компонентов SeekBar

В листинге 5.31 определяется анонимный внутренний класс, реализующий интерфейс `OnSeekBarChangeListener` для обработки событий, возникающих при изменении компонентов `SeekBar` в диалоговом окне `Choose Color`. Он регистрируется как обработчик события `SeekBar` в листинге 5.28 (строки 50–53). Метод `onProgressChanged` (строки 105–114) вызывается при изменении позиции ползунка `SeekBar`. Если пользователь переместил ползунок `SeekBar` (строка 109), новый цвет сохраняется в строках 110–112. Статический метод `argb` класса `Color` объединяет значения `SeekBar` в `Color` и возвращает соответствующий цвет в формате `int`. Затем метод `setBackgroundColor` класса `View` используется для обновления `colorView` цветом, соответствующим текущему состоянию `SeekBar`.

Листинг 5.31. Анонимный внутренний класс, реализующий интерфейс OnSeekBarChangeListener для обработки событий компонентов SeekBar

```
101    // OnSeekBarChangeListener для компонентов SeekBar в диалоговом окне
102    private final OnSeekBarChangeListener colorChangedListener =
103       new OnSeekBarChangeListener() {
```

```
104          // Отображение обновленного цвета
105          @Override
106          public void onProgressChanged(SeekBar seekBar, int progress,
107             boolean fromUser) {
108
109                if (fromUser) // Изменено пользователем (не программой)
110                   color = Color.argb(alphaSeekBar.getProgress(),
111                      redSeekBar.getProgress(), greenSeekBar.getProgress(),
112                      blueSeekBar.getProgress());
113                colorView.setBackgroundColor(color);
114          }
115
116          @Override
117          public void onStartTrackingTouch(SeekBar seekBar) {}
                    // Обязательный метод
118
119          @Override
120          public void onStopTrackingTouch(SeekBar seekBar) {}
                    // Обязательный метод
121       };
122 }
```

5.10. Класс LineWidthDialogFragment

Класс `LineWidthDialogFragment` (листинг 5.32) расширяет `DialogFragment` для создания окна `AlertDialog`, в котором определяется толщина линии. Этот класс аналогичен классу `ColorDialogFragment`, поэтому здесь рассматриваются только важнейшие различия. Единственная переменная класса — компонент `ImageView` (строка 21), в которой рисуется образец линии с текущей толщиной.

Листинг 5.32. Класс LineWidthDialogFragment

```
 1 // LineWidthDialogFragment.java
 2 // Используется для выбора толщины линии в DoodleView
 3 package com.deitel.doodlz;
 4
 5 import android.app.Activity;
 6 import android.app.AlertDialog;
 7 import android.app.Dialog;
 8 import android.content.DialogInterface;
 9 import android.graphics.Bitmap;
10 import android.graphics.Canvas;
11 import android.graphics.Paint;
12 import android.os.Bundle;
13 import android.support.v4.app.DialogFragment;
14 import android.view.View;
15 import android.widget.ImageView;
16 import android.widget.SeekBar;
17 import android.widget.SeekBar.OnSeekBarChangeListener;
18
19 // Класс диалогового окна выбора цвета
```

```
20 public class LineWidthDialogFragment extends DialogFragment {
21    private ImageView widthImageView;
22
23    // Создает и возвращает AlertDialog
24    @Override
25    public Dialog onCreateDialog(Bundle bundle) {
26       // Создание диалогового окна
27       AlertDialog.Builder builder =
28          new AlertDialog.Builder(getActivity());
29       View lineWidthDialogView =
30          getActivity().getLayoutInflater().inflate(
31             R.layout.fragment_line_width, null);
32       builder.setView(lineWidthDialogView); // Добавление GUI
33
34       // Назначение сообщения AlertDialog
35       builder.setTitle(R.string.title_line_width_dialog);
36
37       // Получение ImageView
38       widthImageView = (ImageView) lineWidthDialogView.findViewById(
39          R.id.widthImageView);
40
41       // Настройка widthSeekBar
42       final DoodleView doodleView = getDoodleFragment().getDoodleView();
43       final SeekBar widthSeekBar = (SeekBar)
44          lineWidthDialogView.findViewById(R.id.widthSeekBar);
45       widthSeekBar.setOnSeekBarChangeListener(lineWidthChanged);
46       widthSeekBar.setProgress(doodleView.getLineWidth());
47
48       // Добавление кнопки Set Line Width
49       builder.setPositiveButton(R.string.button_set_line_width,
50          new DialogInterface.OnClickListener() {
51             public void onClick(DialogInterface dialog, int id) {
52                doodleView.setLineWidth(widthSeekBar.getProgress());
53             }
54          }
55       );
56
57       return builder.create(); // Возвращение диалогового окна
58    }
59
60    // Возвращает ссылку на MainActivityFragment
61    private MainActivityFragment getDoodleFragment() {
62       return (MainActivityFragment) getFragmentManager().findFragmentById(
63          R.id.doodleFragment);
64    }
65
66    // Сообщает MainActivityFragment, что диалоговое окно находится на экране
67    @Override
68    public void onAttach(Activity activity) {
69       super.onAttach(activity);
70       MainActivityFragment fragment = getDoodleFragment();
71
72       if (fragment != null)
73          fragment.setDialogOnScreen(true);
74    }
```

```
 75
 76    // Сообщает MainActivityFragment, что окно не отображается
 77    @Override
 78    public void onDetach() {
 79       super.onDetach();
 80       MainActivityFragment fragment = getDoodleFragment();
 81
 82       if (fragment != null)
 83          fragment.setDialogOnScreen(false);
 84    }
 85
 86    // OnSeekBarChangeListener для SeekBar в диалоговом окне толщины линии
 87    private final OnSeekBarChangeListener lineWidthChanged =
 88       new OnSeekBarChangeListener() {
 89          final Bitmap bitmap = Bitmap.createBitmap(
 90             400, 100, Bitmap.Config.ARGB_8888);
 91          final Canvas canvas = new Canvas(bitmap); // Рисует на Bitmap
 92
 93          @Override
 94          public void onProgressChanged(SeekBar seekBar, int progress,
 95             boolean fromUser) {
 96             // Настройка объекта Paint для текущего значения SeekBar
 97             Paint p = new Paint();
 98             p.setColor(
 99                getDoodleFragment().getDoodleView().getDrawingColor());
100             p.setStrokeCap(Paint.Cap.ROUND);
101             p.setStrokeWidth(progress);
102
103             // Стирание объекта Bitmap и перерисовка линии
104             bitmap.eraseColor(
105                getResources().getColor(android.R.color.transparent,
106                   getContext().getTheme()));
107             canvas.drawLine(30, 50, 370, 50, p);
108             widthImageView.setImageBitmap(bitmap);
109          }
110
111          @Override
112          public void onStartTrackingTouch(SeekBar seekBar) {} //
                    Обязательный метод
113
114          @Override
115          public void onStopTrackingTouch(SeekBar seekBar) {} //
                    Обязательный метод
116       };
117 }
```

5.10.1. Метод onCreateDialog

Метод `onCreateDialog` (строки 24–58) заполняет пользовательское представление (строки 29–31), определенное в файле `fragment_line_width.xml` с графическим интерфейсом выбора толщины линии, после чего связывает представление с окном `AlertDialog`, вызывая метод `setView` объекта `AlertDialog.Builder`

(строка 32). Строки 38–39 получают ссылку на компонент `ImageView`, в котором будет выводиться образец линии. Затем строки 42–46 получают ссылку на компонент `widthSeekBar`, регистрируют `lineWidthChanged` (строки 87–116) как слушателя `SeekBar` и устанавливают текущую толщину линии как текущее значение `SeekBar`. Строки 49–55 определяют позитивную кнопку диалогового окна таким образом, чтобы при нажатии кнопки `Set Line Width` вызывался метод `setLineWidth` класса `DoodleView`. Строка 57 возвращает объект `AlertDialog` для отображения на экране.

5.10.2. Анонимный внутренний класс для обработки событий widthSeekBar

В строках 87–116 определяется слушатель `OnSeekBarChangeListener` с именем `lineWidthChanged`, реагирующий на события при изменении компонента `SeekBar` в диалоговом окне `Choose Line Width`. В строках 89–90 создается объект `Bitmap`, на котором будет выводиться образец, представляющий выбранную толщину линии. В строке 91 создается объект `Canvas` для рисования на `Bitmap`. Метод `onProgressChanged` (строки 93–109) рисует образец линии с текущим цветом и толщиной, определяемой значением `SeekBar`. Сначала в строках 97–101 настраивается объект `Paint` для рисования образца линии. Метод `setStrokeCap` класса `Paint` (строка 100) определяет внешний вид концов линии — в нашем случае используются закругленные концы (`Paint.Cap.ROUND`). В строках 104–106 фон объекта `bitmap` заполняется предопределенным в Android цветом `android.R.color.transparent` методом `eraseColor` класса `Bitmap`. Объект `canvas` используется для рисования образца линии. Наконец, строка 115 выводит объект `bitmap` на `widthImageView`, передавая его методу `setImageBitmap` компонента `ImageView`.

5.11. Класс EraseImageDialogFragment

Класс `EraseImageDialogFragment` (листинг 5.33) расширяет `DialogFragment` для создания окна `AlertDialog`, в котором пользователь подтверждает стирание всего изображения. Этот класс аналогичен классам `ColorDialogFragment` и `LineWidthDialogFragment`, поэтому здесь рассматривается только метод `onCreateDialog` (строки 15–35). Метод создает окно `AlertDialog` с кнопками стирания изображения и отмены. В строках 24–30 кнопка стирания `Erase Image` настраивается как *позитивная* — когда пользователь касается ее, строка 27 в слушателе кнопки вызывает метод `clear` класса `DoodleView` для стирания изображения. Строка 33 настраивает кнопку отмены `Cancel` как

негативную — если пользователь коснется этой кнопки, диалоговое окно будет закрыто. В данном случае используется предопределенный строковый ресурс Android `android.R.string.cancel`. За информацией о других предопределенных строковых ресурсах обращайтесь по адресу

http://developer.android.com/reference/android/R.string.html

Строка 34 возвращает объект `AlertDialog`.

Листинг 5.33. Класс EraseImageDialogFragment

```
 1 // EraseImageDialogFragment.java
 2 // Фрагмент для стирания изображения
 3 package com.deitel.doodlz;
 4
 5 import android.app.Activity;
 6 import android.app.AlertDialog;
 7 import android.app.Dialog;
 8 import android.support.v4.app.DialogFragment;
 9 import android.content.DialogInterface;
10 import android.os.Bundle;
11
12 // Класс диалогового окна
13 public class EraseImageDialogFragment extends DialogFragment {
14    // Создание и возвращение объекта AlertDialog
15    @Override
16    public Dialog onCreateDialog(Bundle bundle) {
17       AlertDialog.Builder builder =
18          new AlertDialog.Builder(getActivity());
19
20       // Назначение сообщения AlertDialog
21       builder.setMessage(R.string.message_erase);
22
23       // Добавление кнопки стирания
24       builder.setPositiveButton(R.string.button_erase,
25          new DialogInterface.OnClickListener() {
26             public void onClick(DialogInterface dialog, int id) {
27                getDoodleFragment().getDoodleView().clear(); // Очистка
28             }
29          }
30       );
31
32       // Добавление кнопки стирания
33       builder.setNegativeButton( android.R.string.cancel, null);
34       return builder.create(); // Возвращает диалоговое окно
35    }
36
37    // Получение ссылки на MainActivityFragment
38    private MainActivityFragment getDoodleFragment() {
39       return (MainActivityFragment) getFragmentManager().findFragmentById(
40          R.id.doodleFragment);
41    }
42
```

```
43    // Сообщает MainActivityFragment, что окно находится на экране
44    @Override
45    public void onAttach(Activity activity) {
46       super.onAttach(activity);
47       MainActivityFragment fragment = getDoodleFragment();
48
49       if (fragment != null)
50          fragment.setDialogOnScreen(true);
51    }
52
53    // Сообщает MainActivityFragment, что окно не отображается
54    @Override
55    public void onDetach() {
56       super.onDetach();
57       MainActivityFragment fragment = getDoodleFragment();
58
59       if (fragment != null)
60          fragment.setDialogOnScreen(false);
61    }
62 }
```

5.12. Резюме

В этой главе мы создали приложение `Doodlz`, которое позволяет пользователю рисовать на экране одним или несколькими пальцами. В приложении реализована функция стирания посредством встряхивания устройства: класс Android `SensorManager` был использован для регистрации слушателя `SensorEventListener`, реагирующего на события акселерометра (также вы узнали о том, что Android поддерживает много других видов датчиков).

Мы создали субклассы `DialogFragment`, отображающие пользовательские представления в окнах `AlertDialog`, и переопределили методы `onAttach` и `onDetach` жизненного цикла фрагмента, которые вызываются при присоединении или отсоединении фрагментов от родительской активности.

Вы узнали, как связать объект `Canvas` с объектом `Bitmap`, чтобы затем использовать `Canvas` для рисования на `Bitmap`. Был рассмотрен принцип обработки многоточечных касаний, чтобы пользователь мог рисовать несколькими пальцами одновременно. Информация по каждому пальцу хранится в объекте `Path`. Обработка событий касания основана на переопределении метода `onTouchEvent` класса `View`, получающего объект `MotionEvent` с типом события и идентификатором пальца, сгенерировавшего событие. По идентификаторам мы различали пальцы, чтобы добавить информацию в соответствующие объекты `Path`.

Объект `ContentResolver` и метод `MediaStore.Images.Media.insertImage` использовались для сохранения изображений на устройстве. Для включения этой возможности использовалась новая модель разрешений Android 6.0, в которой

у пользователя запрашивается разрешение на сохранение во внешнем хранилище.

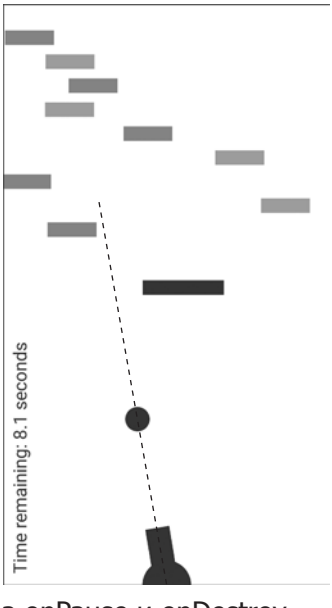
Также вы видели, как при помощи новой инфраструктуры печати Android пользователь может напечатать свой рисунок. Для вывода объектов `Bitmap` использовался класс `PrintHelper` из Android Support Library. Класс `PrintHelper` отображает интерфейс для выбора принтера или сохранения изображения в документе PDF. Для включения функциональности Android Support Library в приложение мы использовали систему Gradle для создания зависимости от этой библиотеки.

В главе 6 мы построим приложение `Cannon Game`. При его создании будет использована многопоточность и покадровая анимация, будет выполняться обработка жестов. Вы узнаете, как создать цикл с максимально быстрым обновлением экрана, чтобы анимации воспроизводились плавно, а с точки зрения пользователя приложение работало с постоянной скоростью независимо от скорости процессора конкретного устройства.

6 Приложение Cannon Game

Покадровая анимация, графика, звук, программные потоки, SurfaceView и SurfaceHolder, режим погружения и полноэкранный режим

В этой главе…



- Создание игры, несложной в реализации и интересной
- Создание субкласса SurfaceView и его использование для отображения игровой графики в отдельном программном потоке
- Создание графики с использованием Paint и Canvas
- Переопределение метода onTouchEvent класса View для обработки событий касания
- Простое обнаружение столкновений
- Добавление звука в приложение с использованием SoundPool и AudioManager
- Переопределение методов жизненного цикла onPause и onDestroy класса Fragment
- Использование режима погружения, при котором игра занимает весь экран, но у пользователя остается возможность получить доступ к системным панелям

6.1. Введение

В игре `Cannon Game`[1] требуется разрушить состоящую из семи частей мишень в течение 10 секунд, отведенных на игру (рис. 6.1). Игра включает четыре визуальных компонента — *пушку*, управляемую пользователем, *пушечное ядро*, *мишень* и *блок*, защищающий мишень. Чтобы навести пушку на цель, коснитесь пальцем экрана. Пушка нацелится в точку, в которой вы коснулись экрана пальцем, и выстрелит по прямой в указанном направлении.

В начале игры каждому пользователю выделяются 10 секунд игрового времени. При каждом попадании в секцию мишени к лимиту времени *добавляются* три секунды, а при каждом попадании в блок *вычитаются* две секунды. Игра считается выигранной, если были разрушены все секции мишени до истечения лимита времени. Если значение таймера уменьшается до нуля до разрушения мишени, вы проиграете. В конце игры приложение отображает диалоговое окно `AlertDialog`, в котором отображается результат игры, а также количество сделанных выстрелов и время игры (рис. 6.2).

После выстрела из пушки приложение воспроизводит *звук выстрела*. После попадания пушечного ядра в секцию мишени раздается звук разбивающегося стекла, а сама секция исчезает с экрана. Если ядро попадает в блок, раздается звук удара, а ядро отскакивает назад. При этом блок не разрушается. Мишень и блок перемещаются *по вертикали* с разной скоростью, а направление перемещения меняется после соприкосновения с верхней или нижней частью экрана.

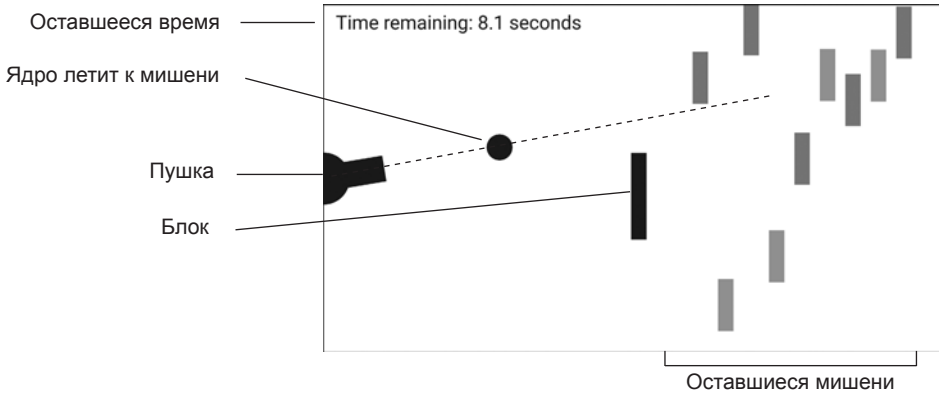
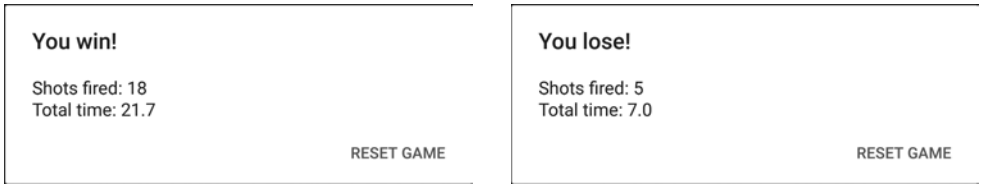


Рис. 6.1. Приложение Cannon Game

[1] Мы хотим поблагодарить профессора Юга Берсини (Hugues Bersini) — автора книги по объектно-ориентированному программированию на французском языке, написанной для Éditions Eyrolles, Secteur Informatique, — за его предложения по переработке исходной версии игры Cannon Game. Мы использовали их в последней версии приложения для этой книги, а также в книге «iOS® 8 for Programmers: An App-Driven Approach».



а *б*

Рис. 6.2. Диалоговое окно AlertDialog приложения Cannon Game с сообщениями о выигрыше и проигрыше: *а* — окно AlertDialog после разрушения всех семи секций мишени; *б* — окно AlertDialog выводится в том случае, если время истекло до разрушения всех секций мишени

(Примечание: из-за проблем быстродействия эмулятора Android это приложение следует тестировать на физическом устройстве.)

6.2. Тестирование приложения Cannon Game

Открытие и выполнение приложения

Запустите Android Studio и откройте проект Cannon Game из папки `CannonGame` в примерах книги. Запустите приложение в эмуляторе AVD или на устройстве. Среда разработки строит проект и запускает приложение.

Игра

Коснитесь экрана, чтобы навести пушку и выстрелить. Вы сможете выстрелить только в том случае, если на экране не находится другое ядро. Если приложение выполняется на экране AVD, в качестве «пальца» будет использоваться мышь. Попытайтесь уничтожить мишень как можно быстрее. Игра кончается, если таймер дойдет до нуля или будут уничтожены все девять мишеней.

6.3. Обзор применяемых технологий

В этом разделе вашему вниманию представлены несколько новых технологий, использованных при создании приложения `Cannon Game`. Эти технологии будут рассмотрены в порядке их описания в главе.

6.3.1. Использование папки ресурсов res/raw

Медиафайлы (например, звуки), используемые в приложении `Cannon Game`, находятся в папке ресурсов приложения `res/raw`. О том, как создать эту папку, рассказано в разделе 6.4.5; остается скопировать в нее звуковые файлы.

6.3.2. Методы жизненного цикла активности и фрагмента

Методы жизненного цикла активности и фрагмента были представлены в разделе 5.3.1. В этом приложении используется метод жизненного цикла `onDestroy` фрагмента. При завершении активности вызывается ее метод `onDestroy`, который в свою очередь вызывает методы `onDestroy` всех фрагментов, находящихся под управлением активности. В классе `CannonFragment` этот метод используется для освобождения звуковых ресурсов `CannonView`.

ЗАЩИТА ОТ ОШИБОК 6.1

Вызов метода onDestroy не гарантирован, поэтому его следует использовать только для освобождения ресурсов, но не для сохранения данных. Документация Android рекомендует сохранять данные в методах onPause и onSaveInstance.

6.3.3. Переопределение метода onTouchEvent класса View

Пользователь взаимодействует с приложением, касаясь экрана устройства. *Касание* нацеливает пушку на определенную точку экрана и производит выстрел. Чтобы организовать обработку простых событий касания `CannonView`, переопределите метод `onTouchEvent` класса `View` (раздел 6.13.14), а затем воспользуйтесь константами класса `MotionEvent` (пакет `android.view`) для определения типа происходящих событий и их последующей обработки.

6.3.4. Добавление звука с помощью SoundPool и AudioManager

Для управления звуковыми эффектами приложения используется класс `SoundPool` (пакет `android.media`), предоставляющий средства для загрузки, воспроизведения и выгрузки звуков. Для воспроизведения используется один из нескольких аудиопотоков Android, включающих потоки для *сигналов*, *музыки*, *уведомлений*, *телефонных звонков*, *системных звуков* и т. д. Для создания и настройки объекта `SoundPool` мы будем использовать объект `SoundPool.Builder`. Кроме того, объект `AudioAttributes.Builder` будет использоваться для создания объекта `AudioAttributes`, связанного с `SoundPool`. В документации Android рекомендуется использовать для воспроизведения звука в играх *музыкальный аудиопоток*. С помощью метода `setVolumeControlStream` класса `Activity` определяется возможность управления громкостью звука в игре кнопками регулировки

громкости устройства. Метод получает константу из класса `AudioManager` (пакет `android.media`), которая предоставляет доступ к управлению громкостью динамика и телефонных звонков.

6.3.5. Покадровая анимация с помощью потоков, SurfaceView и SurfaceHolder

В этом приложении анимации реализуются вручную — путем обновления элементов игры из отдельного потока выполнения. Для этого используется субкласс `Thread` с методом `run`, который приказывает пользовательскому классу `CannonView` обновить позиции всех элементов игры, а затем обновляет сами элементы. Метод `run` управляет *покадровой анимацией*.

Как правило, любые обновления пользовательского интерфейса должны выполняться в потоке выполнения GUI, потому что компоненты GUI не являются безопасными по отношению к потокам — обновления, выполняемые за пределами потока GUI, могут привести к его нарушению. Но зачастую в играх задействована сложная логика, которая должна работать в отдельных потоках выполнения, иногда эти потоки должны работать с экраном. Для подобных случаев Android поддерживает класс `SurfaceView` — субкласс `View`, на котором другие потоки могут «рисовать» без нарушения потоковой безопасности.

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ 6.1

Важно свести к минимуму объем работы, выполняемой в потоке GUI, чтобы графический интерфейс своевременно реагировал на действия пользователя и не выдавал сообщения ANR (Application Not Responding).

Классом `SurfaceView` можно манипулировать посредством объекта класса `SurfaceHolder`, позволяющего получить объект `Canvas`, на котором выводится графика. Класс `SurfaceHolder` также поддерживает методы, обеспечивающие потоку *монопольный* доступ к объекту `Canvas` для рисования, поскольку рисовать на `SurfaceView` может лишь один поток одновременно. Каждый субкласс `SurfaceView` должен реализовать интерфейс `SurfaceHolder.Callback`, включающий методы, которые вызываются при *создании*, *изменении* (размеров или ориентации) либо уничтожении компонента `SurfaceView`.

6.3.6. Простое обнаружение столкновений

С помощью класса `CannonView` выполняется простое *обнаружение столкновений* пушечного ядра с каким-либо краем компонента `CannonView`, с блоком или с секцией мишени. Способ обнаружения описан в разделе 6.13.11.

Многие фреймворки, предназначенные для разработки игр, предоставляют более сложные механизмы обнаружения столкновений. Сейчас разработчикам доступны многочисленные библиотеки разработки игр самого разного уровня, от простейших 2D-игр до сложных игр в стиле игровых приставок (таких, как Sony PlayStation® и Microsoft's Xbox®). В табл. 6.1 перечислены некоторые фреймворки для разработки игр — существуют десятки других. Многие поддерживают несколько платформ, включая Android и iOS; некоторые требуют программирования на C++ или других языках.

Таблица 6.1. Фреймворки для разработки игр

Название	URL
AndEngine	http://www.andengine.org
Cocos2D	http://code.google.com/p/cocos2d-android
GameMaker	http://www.yoyogames.com/studio
libgdx	https://libgdx.badlogicgames.com
Unity	http://www.unity3d.com
Unreal Engine	http://www.unrealengine.com

6.3.7. Режим погружения

Чтобы пользователь меньше отвлекался во время игры, разработчики игр часто используют полноэкранные темы — например, тема `Theme.Material.Light.NoActionBar.Fullscreen` отображает только нижнюю системную панель. В альбомной ориентации на телефонах системная панель располагается у правого края экрана.

В Android 4.4 (KitKat) была добавлена поддержка полноэкранного режима погружения (раздел 6.13.16), в котором приложение может использовать всю площадь экрана. Когда приложение работает в режиме погружения, пользователь может смахнуть от верхнего края экрана, чтобы временно вызвать системные панели. Если пользователь не взаимодействует с системными панелями, они исчезнут через несколько секунд.

6.4. Создание графического интерфейса приложения и файлов ресурсов

В этом разделе будут созданы файлы ресурсов приложения, файлы макетов и классы приложения.

6.4.1. Создание проекта

В этом приложении фрагмент и его макет будут добавлены вручную — большая часть автоматически сгенерированного кода в шаблоне `Blank Activity` здесь не нужна. Создайте новый проект на базе шаблона `Empty Activity`. На шаге `New Project` диалогового окна `Create New Project` укажите следующие значения:

- `Application name`: `Cannon Game`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

В макетном редакторе выберите Nexus 6 в раскрывающемся списке виртуальных устройств (см. рис. 2.8). Это устройство в очередной раз станет основой для разработки приложения. Также удалите компонент `TextView` с текстом `Hello World!` из макета `activity_main.xml`. Как и прежде, добавьте в проект значок приложения.

Настройка приложения для портретной ориентации

Игра спроектирована для выполнения только в портретной ориентации. Выполните действия, описанные в разделе 3.7, чтобы назначить приложению портретную ориентацию экрана, но на этот раз задайте атрибуту `android:screenOrientation` значение `landscape` вместо `portrait`.

6.4.2. Настройка темы для удаления заголовка и панели приложения

Как упоминалось в разделе 6.3.7, разработчики игр часто используют полноэкранные темы (например, тема `Theme.Material.Light.NoActionBar.Fullscreen` с отображением только нижней системной панели, которая в альбомной ориентации находится у правого края). Темы `AppCompat` не включат полноэкранную тему по умолчанию, но изменение темы приложения позволит добиться нужного эффекта.

1. Откройте `styles.xml`.
2. Добавьте следующие строки в элемент `<style>`:

```
<item name="windowNoTitle">true</item>
<item name="windowActionBar">false</item>
<item name="android:windowFullscreen">true</item>
```

Первая строка означает, что заголовок (обычно имя приложения) отображаться не должен. Вторая строка указывает, что не должна отображаться панель приложения. Наконец, последняя строка сообщает, что приложение будет использовать весь экран.

6.4.3. Файл strings.xml

Строковые ресурсы уже создавались в предыдущих главах, поэтому сейчас мы приведем только таблицу с именами ресурсов и соответствующими значениями (табл. 6.2). Сделайте двойной щелчок на файле `strings.xml` из папки `res/values`, чтобы запустить редактор `Translations Editor` для создания этих строковых ресурсов.

Таблица 6.2. Строковые ресурсы, используемые в приложении Cannon Game

Имя ресурса	Значение
results_format	Shots fired: %1$d\nTotal time: %2$.1f
reset_game	Reset Game
win	You win!
lose	You lose!
time_remaining_format	Time remaining: %.1f seconds

6.4.4. Цвета

Приложение рисует на объекте `Canvas` мишени разных цветов. Для нашего приложения в файл `colors.xml` добавлены ресурсы для темно-синего и желтого цвета:

```
<color name="dark">#1976D2</color>
<color name="light">#FFE100</color>
```

6.4.5. Добавление звуков в приложение

Как упоминалось ранее, звуковые файлы хранятся в папке `res/raw` приложения. В нашем приложении используются три звуковых файла — `blocker_hit.wav`, `target_hit.wav` и `cannon_fire.wav`, находящиеся во вложенной папке `sounds` из папки примеров. Чтобы добавить эти файлы в проект, выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `res` проекта и выберите команду `New` ▸ `Android resource directory`, чтобы открыть диалоговое окно `New Resource Directory`.
2. В раскрывающемся списке `Resource type` выберите вариант `raw`. Выбранное значение автоматически появляется в поле `Directory name`.
3. Нажмите кнопку `OK`, чтобы создать папку.

4. Скопируйте звуковые файлы в папку res/raw. В открывшемся диалоговом окне Copy нажмите кнопку OK.

6.4.6. Добавление класса MainActivityFragment

Затем в проект добавляется класс MainActivityFragment.

1. В окне Project щелкните на узле com.deitel.cannongame и выберите команду New ▸ Fragment ▸ Fragment (Blank).
2. В поле Fragment Name введите имя MainActivityFragment, а в поле Fragment Layout — имя fragment_main.
3. Снимите флажки Include fragment factory methods? и Include interface callbacks?.

По умолчанию fragment_main.xml содержит компонент FrameLayout, в котором находится компонент TextView. Удалите его — в этом приложении в компоненте FrameLayout будет отображаться представление CannonView.

6.4.7. Редактирование файла activity_main.xml

В приложении Cannon Game в макете MainActivity отображается только MainActivityFragment. Внесите в макет следующие изменения.

1. Откройте файл activity_main.xml в макетном редакторе и перейдите на вкладку Text.
2. Замените RelativeLayout на fragment и удалите свойства padding, чтобы элемент fragment заполнял весь экран.
3. Переключитесь в режим Design, выберите fragment в окне Component Tree и задайте id значение fragment.
4. Задайте свойству name значение com.deitel.cannongame.MainActivityFragment — вместо того чтобы вводить этот текст, щелкните на кнопке с многоточием справа от поля значения name и выберите класс в открывшемся диалоговом окне Fragments.

Напомним, что в режиме представления Design макетного редактора может выводиться предварительное изображение фрагмента, отображаемого в макете. Если вы не укажете нужный фрагмент в макете MainActivity, в макетном редакторе будет отображаться сообщение «Rendering Problems». Чтобы выбрать фрагмент, щелкните правой кнопкой мыши на фрагменте (на вкладке Design или в окне Component Tree) и выберите команду Choose Preview Layout.... Затем в диалоговом окне Resources выберите имя макета фрагмента.

6.4.8. Добавление CannonView в макет fragment_main.xml

Следующий шаг — добавление `CannonView` в `fragment_main.xml`. Впрочем, перед этим необходимо создать файл `CannonView.java`, чтобы вы могли выбрать класс `CannonView` при размещении `CustomView` в макете. Выполните следующие действия, чтобы создать файл `CannonView.java` и добавить `CannonView` в макет.

1. Откройте папку `java` в окне `Project`.
2. Щелкните правой кнопкой мыши на узле `com.deitel.cannongame`, выберите команду `New` ▸ `Java Class`.
3. В открывшемся диалоговом окне `Create New Class` введите в поле `Name` значение `CannonView` и нажмите кнопку `OK`. Файл автоматически открывается в редакторе.
4. В файле `CannonView.java` укажите, что класс `CannonView` является субклассом `SurfaceView`. Если IDE не добавит команду `import` для `android.view.SurfaceView`, установите курсор сразу же после имени класса `SurfaceView`. Щелкните на красной лампочке () над началом определения класса `CannonView` и выберите команду `Import Class`.
5. Поместите курсор сразу же за `SurfaceView`, если это не было сделано ранее. Щелкните на красной лампочке над началом определения класса `CannonView` и выберите команду `Create constructor matching super`. В диалоговом окне `Choose Super Class Constructors` выберите конструктор с двумя аргументами и нажмите кнопку `OK`. IDE автоматически включает конструктор в класс.
6. Вернитесь к режиму представления `Design` файла `fragment_main.xml` в макетном редакторе.
7. Щелкните на компоненте `CustomView` в разделе `Custom` палитры.
8. В открывшемся диалоговом окне `Views` выберите вариант `CannonView (com.deitel.cannongame)` и нажмите кнопку `OK`.
9. Наведите указатель мыши и щелкните на компоненте `FrameLayout` в окне `Component Tree`. Представление `(CustomView)` — которым в действительности является `CannonView` — должно появиться в `FrameLayout`.
10. Убедитесь в том, что в окне `Component Tree` выбрано представление `(CustomView)`. В окне свойств задайте свойствам `layout:width` и `layout:height` значение `match_parent`.
11. В окне свойств измените значение `id` с `view` на `CannonView`.
12. Сохраните и закройте файл `fragment_main.xml`.

6.5. Классы приложения

Приложение состоит из восьми классов.

- `MainActivity` (субкласс `Activity`; раздел 6.6) — управляет `MainActivityFragment`.
- `MainActivityFragment` (раздел 6.7) — отображает `CannonView`.
- `GameElement` (раздел 6.8) — суперкласс для элементов, перемещающихся вверх и вниз (`Blocker` и `Target`) или по экрану (`Cannonball`).
- `Blocker` (раздел 6.9) — представляет блок, усложняющий попадание в мишени.
- `Target` (раздел 6.10) — представляет мишень, которая может быть уничтожена ядром.
- `Cannon` (раздел 6.11) — представляет пушку, стреляющую при прикосновении к экрану.
- `Cannonball` (раздел 6.12) — представляет ядро, вылетающее из пушки при прикосновении к экрану.
- `CannonView` (раздел 6.13) — содержит логику игры и координирует поведение объектов `Blocker`, `Target`, `Cannonball` и `Cannon`.

Прежде всего следует создать классы `GameElement`, `Blocker`, `Target`, `Cannonball` и `Cannon`. Для каждого класса щелкните правой кнопкой мыши на папке `com.deitel.cannongame` в папке проекта `app/java` и выберите команду `New` ▸ `Java Class`. В диалоговом окне `Create New Class` введите имя класса в поле `Name` и нажмите кнопку `OK`.

6.6. Класс MainActivity

Класс `MainActivity` (листинг 6.1) управляет фрагментом `MainActivityFragment` приложения `Cannon Game`. В этом приложении переопределяется только метод `onCreate` класса `Activity`, заполняющий графический интерфейс. Мы удалили автоматически сгенерированные методы `MainActivity` для управления меню, потому что меню в данном приложении не используется.

Листинг 6.1. Класс MainActivity отображает MainActivityFragment

```
1 // MainActivity.java
2 // MainActivity отображает MainActivityFragment
3 package com.deitel.cannongame;
4
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7
```

```
 8 public class MainActivity extends AppCompatActivity {
 9    // Вызывается при первом запуске приложения
10    @Override
11    protected void onCreate(Bundle savedInstanceState) {
12       super.onCreate(savedInstanceState);
13       setContentView(R.layout.activity_main);
14    }
15 }
```

6.7. Класс MainActivityFragment

Класс фрагмента `MainActivityFragment` (листинг 6.6) переопределяет четыре метода `Fragment`.

- `onCreateView` (строки 17–28) — как вы узнали в разделе 4.3.3, этот метод вызывается после метода `onCreate` класса `Fragment` для построения и возвращения компонента `View`, содержащего графический интерфейс фрагмента. В строках 22–23 происходит заполнение графического интерфейса. Строка 26 получает ссылку на компонент `CannonView` класса `MainActivityFragment`, чтобы вызывать его методы в программе.
- `onActivityCreated` (строки 31–37) — метод вызывается после создания управляющей активности фрагмента. В строке 36 вызывается метод `setVolumeControlStream` класса `Activity`, чтобы громкостью звука в игре можно было управлять кнопками устройства. Существуют семь звуковых потоков, для идентификации которых используются константы `AudioManager`, но для звука в играх рекомендуется использовать музыкальный поток (`AudioManager.STREAM_MUSIC`).
- `onPause` (строки 40–44) — при переводе `MainActivity` в фоновый режим (а следовательно, приостановке) выполняется метод `onPause` фрагмента `MainActivityFragment`. В строке 43 вызывается метод `stopGame` компонента `CannonView` (раздел 6.13.12) для остановки покадровой анимации.
- `onDestroy` (строки 47–51) — при уничтожении активности `MainActivity` ее метод `onDestroy` вызывает метод `onDestroy` фрагмента `MainActivityFragment`. В строке 50 вызывается метод `releaseResources` компонента `CannonView` (раздел 6.13.12) для освобождения звуковых ресурсов.

Листинг 6.2. Класс MainActivityFragment создает и управляет представлением CannonView

```
1 // MainActivityFragment.java
2 // Класс MainActivityFragment создает и управляет CannonView
3 package com.deitel.cannongame;
4
5 import android.media.AudioManager;
```

```
6 import android.os.Bundle;
7 import android.support.v4.app.Fragment;
8 import android.view.LayoutInflater;
9 import android.view.View;
10 import android.view.ViewGroup;
11
12 public class MainActivityFragment extends Fragment {
13    private CannonView cannonView; // Пользовательское представление для игры
14
15    // Вызывается при создании представления фрагмента
16    @Override
17    public View onCreateView(LayoutInflater inflater, ViewGroup container,
18       Bundle savedInstanceState) {
19       super.onCreateView(inflater, container, savedInstanceState);
20
21       // Заполнение макета fragment_main.xml
22       View view =
23          inflater.inflate(R.layout.fragment_main, container, false);
24
25       // Получение ссылки на CannonView
26       cannonView = (CannonView) view.findViewById(R.id.cannonView);
27       return view;
28    }
29
30    // Настройка управления громкостью при создании активности
31    @Override
32    public void onActivityCreated(Bundle savedInstanceState) {
33       super.onActivityCreated(savedInstanceState);
34
35       // Разрешить использование кнопок управления громкостью
36       getActivity().setVolumeControlStream(AudioManager.STREAM_MUSIC);
37    }
38
39    // При приостановке MainActivity игра завершается
40    @Override
41    public void onPause() {
42       super.onPause();
43       cannonView.stopGame(); // Завершение игры
44    }
45
46    // При приостановке MainActivity освобождаются ресурсы
47    @Override
48    public void onDestroy() {
49       super.onDestroy();
50       cannonView.releaseResources();
51    }
52 }
```

6.8. Класс GameElement

Класс `GameElement` (рис. 6.3) — суперкласс `Blocker`, `Target` и `Cannonball` — содержит общие данные и функциональность перемещающегося объекта в приложении `Cannon Game`.

Листинг 6.3. Класс GameElement представляет игровой элемент прямоугольной формы

```
// GameElement.java
// Представляет прямоугольный игровой элемент
package com.deitel.cannongame;

import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;

public class GameElement {
   protected CannonView view; // Представление, содержащее GameElement
   protected Paint paint = new Paint(); // Объект Paint для рисования
   protected Rect shape; // Ограничивающий прямоугольник GameElement
   private float velocityY; // Вертикальная скорость GameElement
   private int soundId; // Звук, связанный с GameElement

   // Открытый конструктор
   public GameElement(CannonView view, int color, int soundId, int x,
      int y, int width, int length, float velocityY) {
      this.view = view;
      paint.setColor(color);
      shape = new Rect(x, y, x + width, y + length); // Определение границ
      this.soundId = soundId;
      this.velocityY = velocityY;
   }

   // Обновление позиции GameElement и проверка столкновений со стенами
   public void update(double interval) {
      // Обновление вертикальной позиции
      shape.offset(0, (int) (velocityY * interval));

      // Если GameElement сталкивается со стеной, изменить направление
      if ( shape.top < 0 && velocityY < 0 ||
          shape.bottom > view.getScreenHeight() && velocityY > 0)
         velocityY *= -1; // Изменить скорость на противоположную
   }

   // Прорисовка GameElement на объекте Canvas
   public void draw(Canvas canvas) {
      canvas.drawRect(shape, paint);
   }

   // Воспроизведение звука, соответствующего типу GameElement
   public void playSound() {
      view.playSound(soundId);
   }
}
```

6.8.1. Поля и конструктор

Конструктор `GameElement` получает ссылку на объект `CannonView` (раздел 6.13), который реализует логику игры и рисует игровые элементы. Конструктор

получает значение int, представляющее 32-разрядный цвет GameElement, и значение int, представляющее идентификатор звука, связанного с GameElement. Все звуки игры вместе с идентификаторами хранятся в CannonView. Конструктору также передаются:

- координаты *x* и *y* левого верхнего элемента GameElement (значения int);
- ширина и высота GameElement (значения int);
- начальная вертикальная скорость velocityY объекта GameElement.

В строке 20 задается цвет объекта paint с использованием представления цвета, переданного конструктору, в формате int. Строка 21 вычисляет границы GameElement и сохраняет их в объекте Rect, представляющем прямоугольник.

6.8.2. Методы update, draw и playSound

Класс GameElement содержит следующие методы.

- update (строки 27–35) — метод вызывается в каждой итерации игрового цикла для обновления позиции GameElement. В строке 29 вертикальная позиция фигуры обновляется с учетом вертикальной скорости (velocityY) и промежутка времени, прошедшего между вызовами update, который передается методу в параметре interval. Строки 32–34 проверяют, столкнулся ли объект GameElement с верхним или нижним краем экрана. При обнаружении столкновения вертикальная скорость объекта меняется на противоположную.
- draw (строки 38–40) — метод вызывается при перерисовке GameElement на экране. Метод получает объект Canvas и рисует GameElement в виде прямоугольника на экране — в классе Cannonball он будет переопределен для рисования круга. Переменная paint класса GameElement задает цвет прямоугольника, а переменная shape класса GameElement задает границы прямоугольника на экране.
- playSound (строки 43–45) — с каждым игровым элементом связывается звук, который воспроизводится вызовом метода playSound. Этот метод передает значение переменной экземпляра soundId методу playSound класса CannonView. Класс CannonView загружает игровые звуки и хранит ссылки на них.

6.9. Класс Blocker

Класс Blocker (листинг 6.4) — субкласс GameElement — представляет блок, который усложняет уничтожение мишени игроком. При столкновении Cannonball с Blocker значение переменной missPenalty класса Blocker вычитается из

оставшегося игрового времени. Для получения `missPenalty` используется метод `getMissPenalty` (строки 17–19) — он вызывается из метода `testForCollisions` класса `CannonView` при уменьшении текущего времени (раздел 6.13.11). Конструктор `Blocker` (строки 9–14) передает свои аргументы и идентификатор звука попадания в блок (`CannonView.BLOCKER_SOUND_ID`) конструктору суперкласса (строка 11), а затем инициализирует `missPenalty`.

Листинг 6.4. Класс Blocker

```
 1 // Blocker.java
 2 // Класс для представления блока (субкласс GameElement)
 3 package com.deitel.cannongame;
 4
 5 public class Blocker extends GameElement {
 6    private int missPenalty; // Потеря времени при попадании в блок
 7
 8    // Конструктор
 9    public Blocker(CannonView view, int color, int missPenalty, int x,
10       int y, int width, int length, float velocityY) {
11       super(view, color, CannonView.BLOCKER_SOUND_ID, x, y, width, length,
12          velocityY);
13       this.missPenalty = missPenalty;
14    }
15
16    // Возвращает штраф при попадании в блок
17    public int getMissPenalty() {
18       return missPenalty;
19    }
20 }
```

6.10. Класс Target

Класс `Target` (листинг 6.5) — субкласс `GameElement` — представляет мишень, которая может быть уничтожена игроком. При столкновении `Cannonball` с `Target` значение переменной `hitReward` класса `Target` прибавляется к оставшемуся игровому времени. Для получения `hitReward` используется метод `getHitReward` (строки 17–19) — он вызывается из метода `testForCollisions` класса `CannonView` при увеличении текущего времени (раздел 6.13.11). Конструктор `Target` (строки 9–14) передает свои аргументы и идентификатор звука попадания в мишень (`CannonView.TARGET_SOUND_ID`) конструктору суперкласса (строка 11), а затем инициализирует `hitReward`.

Листинг 6.5. Класс Target

```
 1 // Target.java
 2 // Класс для представления мишени (субкласс GameElement)
 3 package com.deitel.cannongame;
 4
```

```
 5 public class Target extends GameElement {
 6    private int hitReward; // Прирост времени при попадании в мишень
 7
 8    // Конструктор
 9    public Target(CannonView view, int color, int hitReward, int x, int y,
10       int width, int length, float velocityY) {
11       super(view, color, CannonView.TARGET_SOUND_ID, x, y, width, length,
12          velocityY);
13       this.hitReward = hitReward;
14    }
15
16    // Возвращает прирост при попадании в мишень
17    public int getHitReward() {
18       return hitReward;
19    }
20 }
```

6.11. Класс Cannon

Класс `Cannon` (листинги 6.6–6.10) представляет пушку в приложении `Cannon Game`. Пушка состоит из основания и ствола и стреляет ядрами.

6.11.1. Поля и конструктор

Конструктор `Cannon` (листинг 6.6) получает четыре параметра:

- представление `CannonView`, в котором находится объект `Cannon` (`view`);
- радиус основания пушки (`baseRadius`);
- длина ствола (`barrelLength`);
- толщина ствола (`barrelWidth`).

Строка 25 задает толщину линии объекта `Paint`, чтобы ствол рисовался с заданной толщиной линии `barrelWidth`. В строке 27 ствол пушки выравнивается так, чтобы в исходном состоянии он был параллелен верхнему и нижнему краям экрана. Класс `Cannon` содержит переменную `barrelEnd` типа `Point`, используемую при прорисовке ствола, переменную `barrelAngle` для хранения текущего угла наклона ствола и переменную `cannonball` для хранения объекта `Cannonball`, если тот еще находится на экране.

Листинг 6.6. Поля Cannon и конструктор

```
1 // Cannon.java
2 // Класс представляет пушку, стреляющую ядрами
3 package com.deitel.cannongame;
4
5 import android.graphics.Canvas;
```

```
6 import android.graphics.Color;
7 import android.graphics.Paint;
8 import android.graphics.Point;
9
10 public class Cannon {
11    private int baseRadius; // Радиус основания
12    private int barrelLength; // Длина ствола
13    private Point barrelEnd = new Point(); // Конечная точка ствола
14    private double barrelAngle; // Угол наклона ствола
15    private Cannonball cannonball; // Объект Cannonball
16    private Paint paint = new Paint(); // Объект Paint для рисования пушки
17    private CannonView view; // Представление, в котором находится пушка
18
19    // Конструктор
20    public Cannon(CannonView view, int baseRadius, int barrelLength,
21       int barrelWidth) {
22       this.view = view;
23       this.baseRadius = baseRadius;
24       this.barrelLength = barrelLength;
25       paint.setStrokeWidth(barrelWidth); // Назначение толщины ствола
26       paint.setColor(Color.BLACK); // Пушка окрашена в черный цвет
27       align(Math.PI / 2); // Ствол пушки обращен вправо
28    }
29
```

6.11.2. Метод align

Метод `align` (листинг 6.7) задает направление ствола пушки. Метод получает аргумент с углом наклона, заданным в радианах. Переменные `cannonLength` и `barrelAngle` определяют координаты x и y конечной точки ствола пушки, переменная `barrelEnd` используется для рисования линии от центра основания пушки у левого края экрана до конечной точки ствола. В строке 32 аргумент `barrelAngle` сохраняется для последующего выстрела.

Листинг 6.7. Метод align класса Cannon

```
30    // Нацеливание пушки с заданным углом
31    public void align(double barrelAngle) {
32       this.barrelAngle = barrelAngle;
33       barrelEnd.x = (int) (barrelLength * Math.sin(barrelAngle));
34       barrelEnd.y = (int) (-barrelLength * Math.cos(barrelAngle)) +
35          view.getScreenHeight() / 2;
36    }
37
```

6.11.3. Метод fireCannonball

Метод `fireCannonball` (листинг 6.8) «стреляет» ядром в текущем направлении ствола (`barrelAngle`). В строках 41–46 вычисляются горизонтальная

и вертикальная составляющие скорости Cannonball. Строки 49–50 вычисляют радиус ядра, который составляет CannonView.CANNONBALL_RADIUS_PERCENT процентов от высоты экрана. Строки 53–56 «заряжают» пушку (то есть создают новый объект Cannonball и помещают его в ствол пушки). Метод завершается воспроизведением звука выстрела (строка 58).

Листинг 6.8. Метод fireCannonball

```
38    // Метод создает ядро и стреляет в направлении ствола
39    public void fireCannonball() {
40       // Вычисление горизонтальной составляющей скорости ядра
41       int velocityX = (int) (CannonView.CANNONBALL_SPEED_PERCENT *
42          view.getScreenWidth() * Math.sin(barrelAngle));
43
44       // Вычисление вертикальной составляющей скорости ядра
45       int velocityY = (int) (CannonView.CANNONBALL_SPEED_PERCENT *
46          view.getScreenWidth() * -Math.cos(barrelAngle));
47
48       // Вычисление радиуса ядра
49       int radius = (int) (view.getScreenHeight() *
50          CannonView.CANNONBALL_RADIUS_PERCENT);
51
52       // Построение ядра и размещение его в стволе
53       cannonball = new Cannonball(view, Color.BLACK,
54          CannonView.CANNON_SOUND_ID, -radius,
55          view.getScreenHeight() / 2 - radius, radius, velocityX,
56          velocityY);
57
58       cannonball.playSound(); // Воспроизведение звука выстрела
59    }
60
```

6.11.4. Метод draw

Метод draw (листинг 6.9) рисует пушку на экране. Пушка состоит из двух частей. Сначала рисуется ствол пушки, а затем ее основание.

Листинг 6.9. Метод draw

```
61    // Рисование пушки на объекте Canvas
62    public void draw(Canvas canvas) {
63       // Рисование ствола пушки
64       canvas.drawLine(0, view.getScreenHeight() / 2, barrelEnd.x,
65          barrelEnd.y, paint);
66
67       // Рисование основания пушки
68       canvas.drawCircle(0, (int) view.getScreenHeight() / 2,
69          (int) baseRadius, paint);
70    }
71
```

Рисование ствола пушки методом drawLine

Мы воспользуемся методом `drawLine` класса `Canvas` для рисования ствола пушки (строки 64–65). Метод получает пять параметров: первые четыре представляют координаты *x* и *y* начала и конца линии, а в последнем передается объект `Paint`, определяющий характеристики линии (например, ее толщину). Вспомните, что объект `paint` был настроен для рисования ствола с толщиной, заданной в конструкторе (листинг 6.6, строка 25).

Рисование основания пушки методом drawCircle

В строках 68–69 метод `drawCircle` используется для рисования полукруглого основания пушки. При этом рисуется окружность, центр которой находится на левом краю экрана. Таким образом половина окружности скрывается за левой границей `SurfaceView`.

6.11.5. Методы getCannonball и removeCannonball

В листинге 6.10 приведены методы `getCannonball` и `removeCannonball`. Метод `getCannonball` (строки 73–75) возвращает текущий экземпляр `Cannonball`, хранящийся в объекте `Cannon`.

Если значение `cannonball` равно `null`, это означает, что в данный момент в игре объект `Cannonball` не существует. Представление `CannonView` использует этот метод для того, чтобы запретить появление на экране другого ядра, если одно ядро уже летит к цели (раздел 6.13.8, листинг 6.22). Метод `removeCannonball` (строки 78–80 в листинге 6.10) удаляет объект `CannonBall` из игры, присваивая `cannonball` значение `null`.

Класс `CannonView` использует этот метод для удаления `Cannonball` из игры, когда ядро уничтожает мишень или после выхода с экрана (раздел 6.13.11, рис. 6.25).

Листинг 6.10. Методы getCannonball и removeCannonball

```
72    // Возвращает объект Cannonball, представляющий выпущенное ядро
73    public Cannonball getCannonball() {
74       return cannonball;
75    }
76
77    // Удаляет объект Cannonball из игры
78    public void removeCannonball() {
79       cannonball = null;
80    }
81 }
```

6.12. Класс Cannonball

Класс `Cannonball`, являющийся субклассом `GameElement` (разделы 6.12.1–6.12.4), представляет ядро, летящее в мишень.

6.12.1. Поля и конструктор

Конструктор `Cannonball` (листинг 6.11) получает радиус ядра (вместо ширины и высоты, как конструктор `GameElement`). В строках 15–16 вызывается конструктор суперкласса со значениями ширины и высоты, вычисленными на основании радиуса. Конструктор также получает горизонтальную скорость ядра `velocityX` и вертикальную скорость `velocityY`. Строка 18 инициализирует `onScreen` значением `true`, потому что в исходном состоянии ядро находится на экране.

Листинг 6.11. Поля Cannonball и конструктор

```
 1 // Cannonball.java
 2 // Класс представляет выпущенное ядро
 3 package com.deitel.cannongame;
 4
 5 import android.graphics.Canvas;
 6 import android.graphics.Rect;
 7
 8 public class Cannonball extends GameElement {
 9    private float velocityX;
10    private boolean onScreen;
11
12    // Конструктор
13    public Cannonball(CannonView view, int color, int soundId, int x,
14       int y, int radius, float velocityX, float velocityY) {
15       super(view, color, soundId, x, y,
16          2 * radius, 2 * radius, velocityY);
17       this.velocityX = velocityX;
18       onScreen = true;
19    }
20
```

6.12.2. Методы getRadius, collidesWith, isOnScreen и reverseVelocityX

Метод `getRadius` (листинг 6.12, строки 22–24) вычисляет радиус ядра как половину расстояния между границами `shape.right` и `shape.left` объекта `Cannonball`. Метод `isOnScreen` (строки 32–34) возвращает `true`, если ядро находится на экране.

Листинг 6.12. Методы getRadius, collidesWith, isOnScreen и reverseVelocityX

```
21    // Метод вычисляет радиус ядра
22    private int getRadius() {
23       return (shape.right - shape.left) / 2;
24    }
25
26    // Метод проверяет, столкнулось ли ядро с объектом GameElement
27    public boolean collidesWith(GameElement element) {
28       return (Rect.intersects(shape, element.shape) && velocityX > 0);
29    }
30
31    // Метод возвращает true, если ядро находится на экране
32    public boolean isOnScreen() {
33       return onScreen;
34    }
35
36    // Метод инвертирует горизонтальную скорость ядра
37    public void reverseVelocityX() {
38       velocityX *= -1;
39    }
40
```

Проверка столкновений методом collidesWith

Метод `collidesWith` (строки 27–29) проверяет, столкнулось ли ядро с заданным объектом `GameElement`. Мы выполняем простейшую проверку столкновений для прямоугольных границ `Cannonball`. Для столкновения ядра с `GameElement` должны выполняться два условия.

- Границы `Cannonball`, хранящиеся в объекте `Rect`, должны пересекать границы заданного объекта `GameElement`. Для проверки пересечения используется метод `intersects` класса `Rect`.
- Ядро должно двигаться горизонтально по направлению к заданному объекту `GameElement`. Ядро перемещается слева направо (пока не попадет в блок). Если горизонтальная скорость `velocityX` положительна, значит, ядро двигается слева направо к объекту `GameElement`.

Инвертирование горизонтальной скорости ядра методом reverseVelocityX

Метод `reverseVelocityX` меняет горизонтальную скорость ядра на противоположную, для чего `velocityX` умножается на –1. Если метод `collidesWith` возвращает `true`, метод `testForCollisions` класса `CannonView` вызывает `reverseVelocityX` для смены горизонтальной скорости ядра на противоположную, чтобы ядро отразилось в направлении пушки (раздел 6.13.11).

6.12.3. Метод update

Метод `update` (листинг 6.13) сначала вызывает метод `update` суперкласса (строка 44) для обновления вертикальной скорости ядра и проверки вертикальных столкновений. В строке 47 метод `offset` класса `Rect` используется для горизонтального смещения объекта `Cannonball`. Величина смещения определяется умножением горизонтальной скорости (`velocityX`) на промежуток времени (`interval`). В строках 50–53 `onScreen` задается значение `false`, если `Cannonball` соприкасается с одной из сторон экрана.

Листинг 6.13. Переопределенный метод update класса GameElement

```
41    // Обновление позиции ядра
42    @Override
43    public void update(double interval) {
44       super.update(interval); // Обновление вертикальной позиции ядра
45
46       // Обновление горизонтальной позиции
47       shape.offset((int) (velocityX * interval), 0);
48
49       // Если ядро уходит за пределы экрана
50       if (shape.top < 0 || shape.left < 0 ||
51          shape.bottom > view.getScreenHeight() ||
52          shape.right > view.getScreenWidth())
53          onScreen = false; // Убрать с экрана
54    }
55
```

6.12.4. Метод draw

Метод `draw` (листинг 6.14) переопределяет метод `draw` класса `GameElement` и использует метод `drawCircle` класса `Canvas` для рисования ядра в текущей позиции. Первые два аргумента представляют координаты центра круга. Третий аргумент содержит радиус круга. В последнем аргументе передается объект `Paint` с графическими атрибутами рисования круга.

Листинг 6.14. Переопределенный метод draw класса GameElement

```
56    // Рисование ядра на объекте Canvas
57    @Override
58    public void draw(Canvas canvas) {
59       canvas.drawCircle(shape.left + getRadius(),
60          shape.top + getRadius(), getRadius(), paint);
61    }
62 }
```

6.13. Класс CannonView Subclass

Класс `CannonView` (листинги 6.15–6.29) является субклассом `View`. Он реализует логику приложения `Cannon Game` и рисует игровые объекты на экране.

6.13.1. Команды package и import

В листинге 6.15 приведены команды `package` и `import` класса `CannonView`.

В разделе 6.3 описаны новые классы и интерфейсы, используемые классом `CannonView`. В листинге 6.15 они выделены жирным шрифтом.

Листинг 6.15. Команды package и import класса CannonView

```
 1 // CannonView.java
 2 // Класс отображает игровые объекты и управляет приложением Cannon Game
 3 package com.deitel.cannongame;
 4
 5 import android.app.Activity;
 6 import android.app.AlertDialog;
 7 import android.app.Dialog;
 8 import android.app.DialogFragment;
 9 import android.content.Context;
10 import android.content.DialogInterface;
11 import android.graphics.Canvas;
12 import android.graphics.Color;
13 import android.graphics.Paint;
14 import android.graphics.Point;
15 import android.media.AudioAttributes;
16 import android.media.SoundPool;
17 import android.os.Build;
18 import android.os.Bundle;
19 import android.util.AttributeSet;
20 import android.util.Log;
21 import android.util.SparseIntArray;
22 import android.view.MotionEvent;
23 import android.view.SurfaceHolder;
24 import android.view.SurfaceView;
25 import android.view.View;
26
27 import java.util.ArrayList;
28 import java.util.Random;
29
30 public class CannonView extends SurfaceView
31   implements SurfaceHolder.Callback {
32
```

6.13.2. Поля и константы

В листинге 6.16 перечислены константы и переменные экземпляров класса `CannonView`. Многие из них не требуют дополнительных объяснений, но будут

рассмотрены по мере их применения в коде. Многие константы используются в вычислениях, изменяющих размеры игровых элементов в зависимости от размеров экрана.

Листинг 6.16. Статические переменные и переменные экземпляров CannonView

```
33    private static final String TAG = "CannonView"; // Для регистрации ошибок
34
35    // Игровые константы
36    public static final int MISS_PENALTY = 2; // Штраф при промахе
37    public static final int HIT_REWARD = 3; // Прибавка при попадании
38
39    // Константы для рисования пушки
40    public static final double CANNON_BASE_RADIUS_PERCENT = 3.0 / 40;
41    public static final double CANNON_BARREL_WIDTH_PERCENT = 3.0 / 40;
42    public static final double CANNON_BARREL_LENGTH_PERCENT = 1.0 / 10;
43
44    // Константы для рисования ядра
45    public static final double CANNONBALL_RADIUS_PERCENT = 3.0 / 80;
46    public static final double CANNONBALL_SPEED_PERCENT = 3.0 / 2;
47
48    // Константы для рисования мишеней
49    public static final double TARGET_WIDTH_PERCENT = 1.0 / 40;
50    public static final double TARGET_LENGTH_PERCENT = 3.0 / 20;
51    public static final double TARGET_FIRST_X_PERCENT = 3.0 / 5;
52    public static final double TARGET_SPACING_PERCENT = 1.0 / 60;
53    public static final double TARGET_PIECES = 9;
54    public static final double TARGET_MIN_SPEED_PERCENT = 3.0 / 4;
55    public static final double TARGET_MAX_SPEED_PERCENT = 6.0 / 4;
56
57    // Константы для рисования блока
58    public static final double BLOCKER_WIDTH_PERCENT = 1.0 / 40;
59    public static final double BLOCKER_LENGTH_PERCENT = 1.0 / 4;
60    public static final double BLOCKER_X_PERCENT = 1.0 / 2;
61    public static final double BLOCKER_SPEED_PERCENT = 1.0;
62
63    // Размер текста составляет 1/18 ширины экрана
64    public static final double TEXT_SIZE_PERCENT = 1.0 / 18;
65
66    private CannonThread cannonThread; // Управляет циклом игры
67    private Activity activity; // Для отображения окна в потоке GUI
68    private boolean dialogIsDisplayed = false;
69
70    // Игровые объекты
71    private Cannon cannon;
72    private Blocker blocker;
73    private ArrayList<Target> targets;
74
75    // Переменные размеров
76    private int screenWidth;
77    private int screenHeight;
78
79    // Переменные для игрового цикла и отслеживания состояния игры
80    private boolean gameOver; // Игра закончена?
```

```
81    private double timeLeft; // Оставшееся время в секундах
82    private int shotsFired; // Количество сделанных выстрелов
83    private double totalElapsedTime; // Затраты времени в секундах
84
85    // Константы и переменные для управления звуком
86    public static final int TARGET_SOUND_ID = 0;
87    public static final int CANNON_SOUND_ID = 1;
88    public static final int BLOCKER_SOUND_ID = 2;
89    private SoundPool soundPool; // Воспроизведение звуков
90    private SparseIntArray soundMap; // Связь идентификаторов с SoundPool
91
92    // Переменные Paint для рисования элементов на экране
93    private Paint textPaint; // Для вывода текста
94    private Paint backgroundPaint; // Для стирания области рисования
95
```

6.13.3. Конструктор

В листинге 6.17 приведен конструктор класса `CannonView`. При заполнении объекта `View` вызывается его конструктор, в аргументах которого передаются объекты `Context` и `AttributeSet`. Первый представляет активность, отображающую фрагмент `MainActivityFragment` с `CannonView`, а второй (пакет `android.util`) содержит значения атрибутов `CannonView`, заданные в XML-документе макета. Эти аргументы передаются конструктору суперкласса (строка 96), чтобы обеспечить правильную настройку пользовательского представления значениями стандартных атрибутов `View`, указанных в XML-документе макета. В строке 99 сохраняется ссылка на `MainActivity`, которая используется в конце игры для отображения окна `AlertDialog` из GUI-потока активности. Хотя мы решили сохранить ссылку на `Activity`, ее можно в любой момент получить вызовом метода `getContext`, унаследованного от `View`.

Листинг 6.17. Конструктор CannonView

```
 96    // Конструктор
 97    public CannonView(Context context, AttributeSet attrs) {
 98       super(context, attrs); // Вызов конструктора суперкласса
 99       activity = (Activity) context; // Ссылка на MainActivity
100
101       // Регистрация слушателя SurfaceHolder.Callback
102       getHolder().addCallback(this);
103
104       // Настройка атрибутов для воспроизведения звука
105       AudioAttributes.Builder attrBuilder = new AudioAttributes.Builder();
106       attrBuilder.setUsage(AudioAttributes.USAGE_GAME);
107
108       // Инициализация SoundPool для воспроизведения звука
109       SoundPool.Builder builder = new SoundPool.Builder();
110       builder.setMaxStreams(1);
111       builder.setAudioAttributes(attrBuilder.build());
112       soundPool = builder.build();
```

```
113
114       // Создание Map и предварительная загрузка звуков
115       soundMap = new SparseIntArray(3); // Создание SparseIntArray
116       soundMap.put(TARGET_SOUND_ID,
117          soundPool.load(context, R.raw.target_hit, 1));
118       soundMap.put(CANNON_SOUND_ID,
119          soundPool.load(context, R.raw.cannon_fire, 1));
120       soundMap.put(BLOCKER_SOUND_ID,
121          soundPool.load(context, R.raw.blocker_hit, 1));
122
123       textPaint = new Paint();
124       backgroundPaint = new Paint();
125       backgroundPaint.setColor(Color.WHITE);
126    }
127
```

Регистрация слушателя SurfaceHolder.Callback

В строке 102 `this` (то есть объект `CannonView`) регистрируется как объект, реализующий `SurfaceHolder.Callback` для получения вызовов методов, сообщающих о создании, обновлении и уничтожении `SurfaceView`. Унаследованный от `SurfaceView` метод `getHolder` возвращает объект `SurfaceHolder` для управления `SurfaceView`, а метод `addCallback` класса `SurfaceHolder` сохраняет объект, реализующий интерфейс `SurfaceHolder.Callback`.

Настройка SoundPool и загрузка звуков

В строках 105–121 настраивается конфигурация звуков, используемых в приложении. Сначала мы создаем объект `AudioAttributes.Builder` (строка 105) и вызываем метод `setUsage` (строка 106) с передачей константы, которая описывает, для чего будет использоваться звук. В этом приложении задействована константа `AudioAttribute.USAGE_GAME`, означающая, что звук будет использоваться в игре. Затем создается объект `SoundPool.Builder` (строка 109) для создания объекта `SoundPool`, применяемого для загрузки и воспроизведения звуковых эффектов в приложении. При вызове метода `setMaxStreams` класса `SoundPool.Builder` (строка 110) передается аргумент, представляющий максимальное количество одновременно воспроизводимых звуковых потоков. В игре одновременное воспроизведение звуков не используется, поэтому этот аргумент равен 1. Затем вызывается метод `setAudioAttributes` класса `AudioAttributes.Builder` (строка 111), связывающий атрибуты с объектом `SoundPool` после его создания.

В строке 115 создается объект `SparseIntArray` (переменная `soundMap`), связывающий целочисленные ключи с целочисленными значениями. Класс `SparseIntArray` похож на `HashMap<Integer,Integer>`, но более эффективен для небольшого количества пар «ключ—значение». В нашем примере ключи звуков (см. листинг 6.16, строки 86–88) связываются с идентификаторами загруженных звуков, представленных возвращаемыми значениями метода `load` класса `SoundPool`

(вызываемого в листинге 6.17, строки 117, 119 и 121). Идентификаторы могут использоваться для воспроизведения звука (и последующего возврата ресурсов системе). Метод `load` класса `SoundPool` получает три аргумента — контекст (`Context`) приложения, идентификатор загружаемого звукового файла, а также *приоритет* звука. В соответствии с документацией, описывающей этот метод, последний аргумент в настоящее время не используется, и в нем следует передавать значение 1.

Создание объектов Paint, используемых для рисования игровых элементов

В строках 123–124 создаются объекты `Paint`, используемые для рисования фона и текста `Time Remaining`. По умолчанию используется черный цвет текста, а в строке 125 фону назначается белый цвет.

6.13.4. Переопределение метода onSizeChanged класса View

В коде из листинга 6.18 выполняется переопределение метода `onSizeChanged` класса `View`, который вызывается в случае изменения размеров класса `View`, например при первом добавлении класса `View` в иерархию классов `View` в процессе заполнения разметки. Это приложение всегда отображается в *альбомном режиме*, поэтому метод `onSizeChanged` вызывается только в том случае, если метод `onCreate` активности заполняет графический интерфейс. Этот метод получает новые значения ширины и высоты объекта `View` (при первом вызове метода старые значения ширины и высоты равны 0). В строках 138–139 настраиваются объекты `textPaint`, используемые для вывода текста `Time Remaining`. В строке 138 размер текста устанавливается равным `TEXT_SIZE_PERCENT` процентам от высоты экрана (`screenHeight`). Значение `TEXT_SIZE_PERCENT` и другие масштабные коэффициенты подбирались «методом проб и ошибок», чтобы игровые элементы хорошо смотрелись на экране.

Листинг 6.18. Переопределенный метод onSizeChanged

```
128    // Вызывается при изменении размера SurfaceView,
129    // например при первом добавлении в иерархию View
130    @Override
131    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
132       super.onSizeChanged(w, h, oldw, oldh);
133
134       screenWidth = w; // Сохранение ширины CannonView
135       screenHeight = h; // Сохранение высоты CannonView
136
137       // Настройка свойств текста
138       textPaint.setTextSize((int) (TEXT_SIZE_PERCENT * screenHeight));
```

```
139        textPaint.setAntiAlias(true); // Сглаживание текста
140     }
141
```

6.13.5. Методы getScreenWidth, getScreenHeight и playSound

В листинге 6.19 методы `getScreenWidth` и `getScreenHeight` возвращают ширину и высоту экрана, обновляемые в методе `onSizeChanged` (листинг 6.18). Метод `playSound` (строки 153–155) использует метод `play` объекта `soundPool` для воспроизведения звука с заданным идентификатором `soundId` в `soundMap` (идентификатор был связан со звуком при создании `soundMap` (листинг 6.17, строки 113–119). Объект класса `GameElement` может вызвать метод `playSound` для воспроизведения своего звука.

Листинг 6.19. Методы getScreenWidth, getScreenHeight и playSound класса CannonView

```
142     // Получение ширины экрана
143     public int getScreenWidth() {
144        return screenWidth;
145     }
146
147     // Получение высоты экрана
148     public int getScreenHeight() {
149        return screenHeight;
150     }
151
152     // Воспроизведение звука с заданным идентификатором soundId в soundMap
153     public void playSound(int soundId) {
154        soundPool.play(soundMap.get(soundId), 1, 1, 1, 0, 1f);
155     }
156
```

6.13.6. Метод newGame

Метод `newGame` (листинг 6.20) сбрасывает состояние переменных, используемых для управления игрой. В строках 160–163 создается новый объект `Cannon` со следующими характеристиками:

- радиус основания составляет `CANNON_BASE_RADIUS_PERCENT` процентов от высоты экрана,
- длина ствола составляет `CANNON_BARREL_LENGTH_PERCENT` процентов от ширины экрана и
- толщина ствола составляет `CANNON_BARREL_WIDTH_PERCENT` от высоты экрана.

Листинг 6.20. Метод newGame

```
157    // Сброс всех экранных элементов и запуск новой игры
158    public void newGame() {
159       // Создание новой пушки
160       cannon = new Cannon(this,
161          (int) (CANNON_BASE_RADIUS_PERCENT * screenHeight),
162          (int) (CANNON_BARREL_LENGTH_PERCENT * screenWidth),
163          (int) (CANNON_BARREL_WIDTH_PERCENT * screenHeight));
164
165       Random random = new Random(); // Для случайных скоростей
166       targets = new ArrayList<>(); // Построение нового списка мишеней
167
168       // Инициализация targetX для первой мишени слева
169       int targetX = (int) (TARGET_FIRST_X_PERCENT * screenWidth);
170
171       // Вычисление координаты Y
172       int targetY = (int) ((0.5 - TARGET_LENGTH_PERCENT / 2) *
173          screenHeight);
174
175       // Добавление TARGET_PIECES мишеней в список
176       for (int n = 0; n < TARGET_PIECES; n++) {
177
178          // Получение случайной скорости в диапазоне от min до max
179          // для мишени n
180          double velocity = screenHeight * (random.nextDouble() *
181             (TARGET_MAX_SPEED_PERCENT - TARGET_MIN_SPEED_PERCENT) +
182             TARGET_MIN_SPEED_PERCENT);
183
184          // Цвета мишеней чередуются между белым и черным
185          int color = (n % 2 == 0) ?
186             getResources().getColor(R.color.dark,
187                getContext().getTheme()) :
188             getResources().getColor(R.color.light,
189                getContext().getTheme());
190
191          velocity *= -1; // Противоположная скорость следующей мишени
192
193          // Создание и добавление новой мишени в список
194          targets.add(new Target(this, color, HIT_REWARD, targetX, targetY,
195             (int) (TARGET_WIDTH_PERCENT * screenWidth),
196             (int) (TARGET_LENGTH_PERCENT * screenHeight),
197             (int) velocity));
198
199          // Увеличение координаты x для смещения
200          // следующей мишени вправо
201          targetX += (TARGET_WIDTH_PERCENT + TARGET_SPACING_PERCENT) *
202             screenWidth;
203       }
204
205       // Создание нового блока
206       blocker = new Blocker(this, Color.BLACK, MISS_PENALTY,
207          (int) (BLOCKER_X_PERCENT * screenWidth),
208          (int) ((0.5 - BLOCKER_LENGTH_PERCENT / 2) * screenHeight),
209          (int) (BLOCKER_WIDTH_PERCENT * screenWidth),
210          (int) (BLOCKER_LENGTH_PERCENT * screenHeight),
```

```
211          (float) (BLOCKER_SPEED_PERCENT * screenHeight));
212
213       timeLeft = 10; // Обратный отсчет с 10 секунд
214
215       shotsFired = 0; // Начальное количество выстрелов
216       totalElapsedTime = 0.0; // Обнулить затраченное время
217
218       if (gameOver) {// Начать новую игру после завершения предыдущей
219          gameOver = false; // Игра не закончена
220          cannonThread = new CannonThread(getHolder()); // Создать поток
221          cannonThread.start(); // Запуск потока игрового цикла
222       }
223
224       hideSystemBars();
225    }
226
```

В строке 165 создается новый объект `Random`, используемый для генерирования случайных скоростей мишеней. Строка 166 создает новый контейнер `ArrayList` с объектами мишеней `Target`. Строка 169 инициализирует `targetX` количеством пикселов, на которое первая мишень будет смещена от левого края (`TARGET_FIRST_X_PERCENT` процентов от ширины экрана). В строках 172–173 `targetY` инициализируется для вертикального выравнивания всех мишеней по центру экрана. Строки 176–203 строят `TARGET_PIECES` (9) новых мишеней и добавляют их в массив `targets`. В строках 180–182 скорость новой мишени выбирается как случайное значение в диапазоне между `TARGET_MIN_SPEED_PERCENT` и `TARGET_MAX_SPEED_PERCENT` процентов от высоты экрана. В строках 185–189 обеспечивается чередование цветов новых мишеней (между `R.color.dark` и `R.color.light`) и их скоростей (между положительными и отрицательными). Строка 191 меняет знак скорости каждой новой мишени, чтобы одни мишени двигались вниз, а другие вверх. Новая мишень создается и добавляется в `targets` (строки 194–197). Мишени назначается ширина `TARGET_WIDTH_PERCENT` процентов от ширины экрана и высота `TARGET_HEIGHT_PERCENT` от высоты экрана. Наконец, значение `targetX` увеличивается для определения позиции следующей мишени.

Новый объект `Blocker` создается и сохраняется в переменной `blocker` в строках 206–211. В начале игры блок устанавливается со смещением `BLOCKER_X_PERCENT` процентов от ширины экрана от левого края и выравнивается вертикально по центру экрана. Ширина блока составляет `BLOCKER_WIDTH_PERCENT` процентов от ширины экрана, а высота — `BLOCKER_HEIGHT_PERCENT` процентов от высоты экрана. Блок двигается со скоростью `BLOCKER_SPEED_PERCENT` процентов от высоты экрана.

Если переменная `gameOver` равна `true`, что происходит только после завершения игры, строка 219 сбрасывает флаг `gameOver`, а строки 220–221 создают новый объект `CannonThread` и вызывают его метод `start` для запуска игрового цикла,

управляющего ходом игры. Строка 224 вызывает метод `hideSystemBars` (раздел 6.13.16) для перевода приложения в режим погружения — системные панели скрываются, но пользователь может в любой момент вызвать их смахиванием от верха экрана.

6.13.7. Метод updatePositions

Метод `updatePositions` (листинг 6.21) вызывается методом `run` объекта `CannonThread` (раздел 6.13.15) для обновления позиций элементов на экране и простого обнаружения *столкновений*. Новые положения игровых элементов вычисляются по времени (в миллисекундах), прошедшему между предыдущим и текущим кадром анимации. Таким образом игра определяет расстояние, на которое перемещается каждый элемент, в соответствии с частотой обновления изображения на экране устройства. Эти вопросы будут разобраны более подробно при рассмотрении циклов игры в разделе 6.13.15.

Листинг 6.21. Метод updatePositions класса CannonView

```
227    // Многократно вызывается CannonThread для обновления элементов игры
228    private void updatePositions(double elapsedTimeMS) {
229       double interval = elapsedTimeMS / 1000.0; // Преобразовать в секунды
230
231       // Обновление позиции ядра
232       if (cannon.getCannonball() != null)
233          cannon.getCannonball().update(interval);
234
235       blocker.update(interval); // Обновление позиции блока
236
237       for (GameElement target : targets)
238          target.update(interval); // Обновление позиции мишени
239
240       timeLeft -= interval; // Уменьшение оставшегося времени
241
242       // Если счетчик достиг нуля
243       if (timeLeft <= 0) {
244          timeLeft = 0.0;
245          gameOver = true; // Игра закончена
246          cannonThread.setRunning(false); // Завершение потока
247          showGameOverDialog(R.string.lose); // Сообщение о проигрыше
248       }
249
250       // Если все мишени поражены
251       if (targets.isEmpty()) {
252          cannonThread.setRunning(false); // Завершение потока
253          showGameOverDialog(R.string.win); // Сообщение о выигрыше
254          gameOver = true;
255       }
256    }
257
```

Время от последнего кадра анимации

В строке 229 выполняется преобразование времени, прошедшего с момента последнего кадра анимации (из миллисекунд в секунды). Это значение используется для изменения позиций различных элементов игры.

Обновление позиций блока и мишени

В строках 232–238 изменяются позиции всех игровых элементов, для чего вызываются методы `update` объекта `Cannonball` (если ядро находится на экране), `Blocker` и всех оставшихся объектов `Target`. Метод `update` получает время, прошедшее с момента последнего обновления, чтобы изменение позиций соответствовало прошедшему времени.

Обновление оставшегося времени и проверка таймера

Значение переменной `timeLeft` уменьшается на время, которое прошло с момента воспроизведения предыдущего кадра анимации (строка 240). Если значение переменной `timeLeft` становится равным нулю, игра завершается. Переменной `timeLeft` присваивается значение 0.0 (если этого не сделать, на экране может отобразиться отрицательное значение времени). Затем переменной `gameOver` присваивается значение `true`, поток `CannonThread` завершается вызовом `setRunning` с аргументом `false`, а также вызывается метод `showGameOverDialog` с идентификатором строкового ресурса, используемого для вывода сообщения о проигрыше.

6.13.8. Метод alignAndFireCannonball

Если пользователь касается экрана, метод `onTouchEvent` (раздел 6.13.14) вызывает метод `alignAndFireCannonball` (листинг 6.22). В строках 267–272 вычисляется угол `angle`, необходимый для наведения пушки в точку касания. Строка 275 вызывает метод `align` класса `Cannon` для наведения пушки под углом `angle`. Наконец, если ядро существует и находится на экране, в строках 280–281 пушка стреляет, а значение переменной `shotsFired` увеличивается.

Листинг 6.22. Метод alignAndFireCannonball класса CannonView

```
258    // Метод определяет угол наклона ствола и стреляет из пушки,
259    // если ядро не находится на экране
260    public void alignAndFireCannonball(MotionEvent event) {
261       // Получение точки касания в этом представлении
262       Point touchPoint = new Point((int) event.getX(),
263          (int) event.getY());
264
265       // Вычисление расстояния точки касания от центра экрана
266       // по оси y
267       double centerMinusY = (screenHeight / 2 - touchPoint.y);
```

```
268
269       double angle = 0; // Инициализировать angle значением 0
270
271       // Вычислить угол ствола относительно горизонтали
272       angle = Math.atan2(touchPoint.x, centerMinusY);
273
274       // Ствол наводится в точку касания
275       cannon.align(angle);
276
277       // Пушка стреляет, если ядро не находится на экране
278       if (cannon.getCannonball() == null ||
279          !cannon.getCannonball().isOnScreen()) {
280          cannon.fireCannonball();
281          ++shotsFired;
282       }
283    }
284
```

6.13.9. Метод showGameOverDialog

После завершения игры метод `showGameOverDialog` (листинг 6.23) выводит фрагмент `DialogFragment` (см. раздел 4.7.10) с результатом игры (выигрыш или проигрыш), количеством сделанных выстрелов и затраченным временем. Вызов метода `setPositiveButton` (строки 301–311) создает кнопку для начала новой игры.

Листинг 6.23. Метод showGameOverDialog класса CannonView

```
285    // Отображение окна AlertDialog при завершении игры
286    private void showGameOverDialog(final int messageId) {
287       // Объект DialogFragment для вывода статистики и начала новой игры
288       final DialogFragment gameResult =
289          new DialogFragment() {
290             // Метод создает объект AlertDialog и возвращает его
291             @Override
292             public Dialog onCreateDialog(Bundle bundle) {
293                // Создание диалогового окна с выводом строки messageId
294                AlertDialog.Builder builder =
295                   new AlertDialog.Builder(getActivity());
296                builder.setTitle(getResources().getString(messageId));
297
298                // Вывод количества выстрелов и затраченного времени
299                builder.setMessage(getResources().getString(
300                   R.string.results_format, shotsFired, totalElapsedTime));
301                builder.setPositiveButton(R.string.reset_game,
302                   new DialogInterface.OnClickListener() {
303                      // Вызывается при нажатии кнопки "Reset Game"
304                      @Override
305                      public void onClick(DialogInterface dialog,
306                         int which) {
307                         dialogIsDisplayed = false;
308                         newGame(); // Создание и начало новой партии
```

```
309                     }
310                  }
311               );
312
313               return builder.create(); // Вернуть AlertDialog
314            }
315         };
316
317      // В UI-потоке FragmentManager используется для вывода DialogFragment
318      activity.runOnUiThread(
319         new Runnable() {
320           public void run() {
321              showSystemBars(); // Выход из режима погружения
322              dialogIsDisplayed = true;
323              gameResult.setCancelable(false); // Модальное окно
324              gameResult.show(activity.getFragmentManager(), "results");
325           }
326        }
327      );
328   }
```

Метод `onClick` слушателя кнопок сигнализирует о том, что диалоговое окно больше не отображается, и вызывает метод `newGame` для настройки и запуска новой игры. Диалоговое окно должно отображаться в потоке GUI, поэтому в строках 318–327 вызывается метод `runOnUiThread` класса `Activity` для определения объекта `Runnable`, который должен быть выполнен в потоке GUI как можно раньше. В аргументе передается объект анонимного внутреннего класса, реализующий `Runnable`. Метод `run` класса `Runnable` вызывает метод `showSystemBars` (раздел 6.13.16) для удаления приложения из режима погружения, устанавливает признак отображения диалогового окна и отображает его.

6.13.10. Метод drawGameElements

Метод `drawGameElements` (листинг 6.24) рисует пушку, пушечное ядро, блок и мишени с помощью класса `SurfaceView`. При этом используется компонент `Canvas`, полученный потоком `CannonThread` (раздел 6.13.15) из реализации `SurfaceHolder` объекта `SurfaceView`.

Листинг 6.24. Метод drawGameElements класса CannonView

```
330   // Рисование элементов игры
331   public void drawGameElements(Canvas canvas) {
332      // Очистка фона
333      canvas.drawRect(0, 0, canvas.getWidth(), canvas.getHeight(),
334         backgroundPaint);
335
336      // Вывод оставшегося времени
337      canvas.drawText(getResources().getString(
338         R.string.time_remaining_format, timeLeft), 50, 100, textPaint);
```

```
339
340       cannon.draw(canvas); // Рисование пушки
341
342       // Рисование игровых элементов
343       if (cannon.getCannonball() != null &&
344          cannon.getCannonball().isOnScreen())
345          cannon.getCannonball().draw(canvas);
346
347       blocker.draw(canvas); // Рисование блока
348
349       // Рисование всех мишеней
350       for (GameElement target : targets)
351          target.draw(canvas);
352    }
353
```

Очистка объекта Canvas методом drawRect

Сначала вызывается метод `drawRect` класса `Canvas` (строки 333–334), который стирает содержимое объекта `Canvas`, чтобы все элементы игры отображались в новых положениях. Этот метод получает в качестве аргументов координаты верхнего левого угла прямоугольника, ширину и высоту прямоугольника и объект `Paint`, определяющий характеристики рисования (`backgroundPaint` выбирает белый цвет рисования).

Вывод оставшегося времени методом drawText класса Canvas

Затем вызывается метод `drawText` класса `Canvas` (строки 337–338) для отображения оставшегося времени игры. В качестве аргументов передаются выводимая строка, координаты x и y ее вывода и компонент `textPaint` (сконфигурирован в строках 138–139), описывающий вывод текста (размер шрифта, цвет и другие атрибуты текста).

Рисование ствола, блока и мишени методом drawLine класса Canvas

В строках 339–350 рисуются ствол пушки, ядро (если оно находится на экране), блок и мишени. Каждый игровой элемент рисуется вызовом метода `draw` с передачей объекта `canvas`.

6.13.11. Метод testForCollisions

Метод `testForCollisions` (листинг 6.25) проверяет, столкнулось ли ядро с какой-либо мишенью или блоком, и применяет некоторые эффекты при обнаружении столкновения. Строки 359–360 проверяют, находится ли ядро на экране. Если проверка дает положительный результат, строка 362 вызывает метод `collidesWith` объекта `Cannonball` для определения того, столкнулось ли ядро с мишенью. При обнаружении столкновения строка 363 вызывает метод

playSound для воспроизведения звука попадания в мишень, строка 366 увеличивает timeLeft на величину «награды», связанной с мишенью, а строки 368–369 удаляют ядро и мишень с экрана. Строка 370 уменьшает n для проведения проверки столкновения для мишени, находящейся теперь в позиции n. В строке 376 уничтожается объект Cannonball, связанный с Cannon, если ядро вышло за пределы экрана. Если ядро все еще находится на экране, то строки 380–381 снова вызывают collidesWith для определения того, столкнулось ли ядро с блоком. В этом случае строка 382 вызывает метод playSound блока для воспроизведения звука попадания в блок, строка 385 меняет направление горизонтальной скорости ядра вызовом метода reverseVelocityX класса Cannonball, а строка 388 уменьшает timeLeft на величину штрафа, связанного с Blocker.

Листинг 6.25. Метод testForCollisions класса CannonView

```
354    // Проверка столкновений с блоком или мишенями
355    // и обработка столкновений
356    public void testForCollisions() {
357       // Удаление мишеней, с которыми
358       // сталкивается ядро
359       if (cannon.getCannonball() != null &&
360          cannon.getCannonball().isOnScreen()) {
361          for (int n = 0; n < targets.size(); n++) {
362             if (cannon.getCannonball().collidesWith(targets.get(n))) {
363                targets.get(n).playSound(); // Звук попадания в мишень
364
365                // Прибавление награды к оставшемуся времени
366                timeLeft += targets.get(n).getHitReward();
367
368                cannon.removeCannonball(); // Удаление ядра из игры
369                targets.remove(n); // Удаление пораженной мишени
370                --n; // Чтобы не пропустить проверку новой мишени
371                break;
372             }
373          }
374       }
375       else { // Удаление ядра, если оно не должно находиться на экране
376          cannon.removeCannonball();
377       }
378
379       // Проверка столкновения с блоком
380       if (cannon.getCannonball() != null &&
381          cannon.getCannonball().collidesWith(blocker)) {
382          blocker.playSound(); // play Blocker hit sound
383
384          // Изменение направления
385          cannon.getCannonball().reverseVelocityX();
386
387          // Уменьшение оставшегося времени на величину штрафа
388          timeLeft -= blocker.getMissPenalty();
389       }
390    }
391
```

6.13.12. Методы stopGame и releaseResources

Методы `onPause` и `onDestroy` (раздел 6.13) класса `MainActivityFragment` вызывают методы `stopGame` и `releaseResources` класса `CannonView` (листинг 6.26) соответственно. Метод `stopGame` (строки 393–396) вызывается из главного класса активности для остановки игры при вызове метода `onPause` класса `Activity` (для упрощения в этом примере не сохраняются сведения о состоянии игры). Метод `releaseResources` (строки 399–402) вызывает метод `release` класса `SoundPool`, с помощью которого выполняется освобождение ресурсов, связанных с `SoundPool`.

Листинг 6.26. Методы stopGame и releaseResources класса CannonView

```
392    // Остановка игры; вызывается методом onPause класса MainActivityFragment
393    public void stopGame() {
394       if (cannonThread != null)
395          cannonThread.setRunning(false); // Приказываем потоку завершиться
396    }
397
398    // Освобождение ресурсов; вызывается методом onDestroy класса CannonGame
399    public void releaseResources() {
400       soundPool.release(); // Освободить все ресурсы, используемые SoundPool
401       soundPool = null;
402    }
403
```

6.13.13. Реализация методов SurfaceHolder.Callback

В листинге 6.27 приведена реализация методов `surfaceChanged`, `surfaceCreated` и `surfaceDestroyed` интерфейса `SurfaceHolder.Callback`. Тело метода `surfaceChanged` в этом приложении остается пустым, потому что приложение *всегда* отображается в портретной ориентации. Этот метод вызывается при изменении размеров или ориентации `SurfaceView` и обычно используется для перерисовки графики для новой конфигурации.

Листинг 6.27. Реализация методов SurfaceHolder.Callback

```
404    // Вызывается при изменении размера поверхности
405    @Override
406    public void surfaceChanged(SurfaceHolder holder, int format,
407       int width, int height) { }
408
409    // Вызывается при создании поверхности
410    @Override
411    public void surfaceCreated(SurfaceHolder holder) {
412       if (!dialogIsDisplayed) {
413          newGame(); // Создание новой игры
```

```
414          cannonThread = new CannonThread(holder); // Создание потока
415          cannonThread.setRunning(true); // Запуск игры
416          cannonThread.start(); // Запуск потока игрового цикла
417       }
418    }
419
420    // Вызывается при уничтожении поверхности
421    @Override
422    public void surfaceDestroyed(SurfaceHolder holder) {
423       // Обеспечить корректную зависимость потока
424       boolean retry = true;
425       cannonThread.setRunning(false); // Завершение cannonThread
426
427       while (retry) {
428          try {
429             cannonThread.join(); // Ожидать завершения cannonThread
430             retry = false;
431          }
432          catch (InterruptedException e) {
433             Log.e(TAG, "Thread interrupted", e);
434          }
435       }
436    }
437
```

Метод `surfaceCreated` (строки 410–418) вызывается при создании `SurfaceView` — например, при первой загрузке приложения или при возвращении его из фонового режима. В нашем приложении `surfaceCreated` используется для создания и запуска потока `CannonThread`, инициирующего цикл игры. Метод `surfaceDestroyed` (строки 421–436) вызывается при уничтожении `SurfaceView` — например, при завершении приложения. Мы используем его для обеспечения корректного завершения `CannonThread`. Сначала в строке 425 метод `setRunning` класса `CannonThread` вызывается с аргументом `false`, который показывает, что поток должен остановиться, после чего строки 427–435 ожидают завершения потока. Тем самым предотвращаются возможные попытки рисования на `SurfaceView` после завершения `surfaceDestroyed`.

6.13.14. Переопределение метода onTouchEvent

В этом примере переопределяется метод `onTouchEvent` класса `View` (листинг 6.28) для получения информации о касании экрана пользователем. Параметр `MotionEvent` содержит информацию о произошедшем событии. В строке 442 метод `getAction` класса `MotionEvent` используется для определения типа события касания. Затем строки 445–446 определяют, коснулся ли пользователь экрана (`MotionEvent.ACTION_DOWN`) или провел пальцем по экрану (`MotionEvent.ACTION_MOVE`). В обоих случаях строка 529 вызывает метод `alignAndFireCannonball` представления `cannonView` для наведения пушки на точку касания и выполнения

выстрела. Затем строка 451 возвращает `true` — признак того, что событие касания было обработано.

Листинг 6.28. Переопределение метода onTouchEvent

```
438    // Вызывается при касании экрана в этой активности
439    @Override
440    public boolean onTouchEvent(MotionEvent e) {
441       // int представляет тип действия, вызвавшего данное событие
442       int action = e.getAction();
443
444       // Пользователь коснулся экрана или провел пальцем по экрану
445       if (action == MotionEvent.ACTION_DOWN ||
446          action == MotionEvent.ACTION_MOVE) {
447          // Выстрел в направлении точки касания
448          alignAndFireCannonball(e);
449       }
450
451       return true;
452    }
453
```

6.13.15. Поток CannonThread: использование потока для создания цикла игры

В листинге 6.29 определяется субкласс класса `Thread`, обновляющий состояние игры. Поток хранит ссылку на метод `SurfaceHolder` класса `SurfaceView` (строка 456) и логическое значение, указывающее, выполняется ли поток.

Листинг 6.29. Вложенный класс CannonThread управляет циклом игры и обновляет игровые элементы каждые TIME_INTERVAL миллисекунд

```
454    // Субкласс Thread для управления циклом игры
455    private class CannonThread extends Thread {
456       private SurfaceHolder surfaceHolder; // Для работы с Canvas
457       private boolean threadIsRunning = true; // По умолчанию
458
459       // Инициализация SurfaceHolder
460       public CannonThread(SurfaceHolder holder) {
461          surfaceHolder = holder;
462          setName("CannonThread");
463       }
464
465       // Изменение состояния выполнения
466       public void setRunning(boolean running) {
467          threadIsRunning = running;
468       }
469
```

```
470       // Управление игровым циклом
471       @Override
472       public void run() {
473          Canvas canvas = null; // Используется для рисования
474          long previousFrameTime = System.currentTimeMillis();
475
476          while (threadIsRunning) {
477             try {
478                // Получение Canvas для монопольного рисования из этого потока
479                canvas = surfaceHolder.lockCanvas(null);
480
481                // Блокировка surfaceHolder для рисования
482                synchronized(surfaceHolder) {
483                   long currentTime = System.currentTimeMillis();
484                   double elapsedTimeMS = currentTime - previousFrameTime;
485                   totalElapsedTime += elapsedTimeMS / 1000.0;
486                   updatePositions(elapsedTimeMS); // Обновление состояния игры
487                   testForCollisions(); // Проверка столкновений с GameElement
488                   drawGameElements(canvas); // Рисование на объекте canvas
489                   previousFrameTime = currentTime; // Обновление времени
490                }
491             }
492             finally {
493                // Вывести содержимое canvas на CannonView
494                // и разрешить использовать Canvas другим потокам
495                if (canvas != null)
496                   surfaceHolder.unlockCanvasAndPost(canvas);
497             }
498          }
499       }
500    }
```

Метод класса `run` (строки 471–499) управляет покадровыми анимациями (этот процесс называется *циклом игры*). Каждое обновление элементов игры на экране выполняется с учетом количества миллисекунд, прошедших с момента последнего обновления. В строке 474 считывается текущее системное время в миллисекундах, прошедшее с момента начала выполнения потока. В строках 476–498 цикл выполняется до тех пор, пока значение `threadIsRunning` не станет равным `false`.

Сначала мы получаем объект `Canvas`, используемый для рисования на `SurfaceView`, вызовом метода `lockCanvas` класса `SurfaceHolder` (строка 479). На `SurfaceView` в любой момент времени должен рисовать только один поток, поэтому сначала нужно *заблокировать* объект `SurfaceHolder`, указав его в круглых скобках блока `synchronized` (строка 482). Затем программа получает текущее время (в миллисекундах), вычисляет прошедшее время и прибавляет его к счетчику времени — это значение применяется для вывода оставшегося времени игры. В строке 486 вызывается метод `updatePositions`, перемещающий все элементы игры; в аргументе этого метода передается время выполнения приложения в миллисекундах. Таким

образом обеспечивается постоянная скорость игры *независимо от используемого устройства*. Если интервал времени между соседними кадрами увеличивается (на медленном устройстве), элементы игры будут перемещаться на большее расстояние при отображении очередного кадра анимации. Если же интервал времени между кадрами меньше (на быстром устройстве), то элементы игры будут перемещаться на меньшее расстояние при отображении очередного кадра. В строке 487 вызывается метод `testForCollisions` для проверки столкновения ядра с блоком или мишенью.

- Если ядро столкнулось с блоком, то метод `testForCollisions` меняет скорость ядра на противоположную.
- Если ядро столкнулось с мишенью, то метод `testForCollisions` удаляет ядро.

И наконец, в строке 488 рисуются элементы игры с использованием объекта `Canvas` класса `SurfaceView`, а в строке 489 значение `currentTime` заменяет предыдущее значение `previousFrameTime` для подготовки вычисления времени между текущим и *следующим* кадром анимации.

6.13.16. Методы hideSystemBars и showSystemBars

В этом приложении используется *режим погружения* — игрок может в любой момент вызвать системные панели, смахивая вниз от верхнего края экрана. Режим погружения доступен только на устройствах с Android 4.4 и выше. По этой причине методы `hideSystemBars` и `showSystemBars` (листинг 6.30) сначала проверяют, что версия Android устройства `Build.VERSION_SDK_INT` больше либо равна `Build.VERSION_CODES_KITKAT` — константы, представляющей Android 4.4 (API уровня 19). Если проверка дает положительный результат, то оба метода используют метод `setSystemUiVisibility` класса `View` для настройки системных панелей и панели приложения (хотя мы уже скрыли панель приложения при изменении темы). Чтобы скрыть системные панели с панелью приложения и перевести интерфейс в режим погружения, следует передать `setSystemUiVisibility` набор констант, объединенных поразрядным оператором OR (`|`) в строках 505–510. Чтобы отобразить системные панели и панель приложения, передайте `setSystemUiVisibility` набор констант в строках 517–519. С этими константами `View` приложение не будет изменяться в размерах при каждом сокрытии и появлении системных панелей и панели приложения. Вместо этого системные панели и панель приложения будут накладываться на `CannonView` — то есть часть `CannonView` будет временно скрываться при появлении системных панелей на экране. За дополнительной информацией о режиме погружения обращайтесь по адресу

http://developer.android.com/training/system-ui/immersive.html

Листинг 6.30. Методы hideSystemBars и showSystemBars

```
501    // Сокрытие системных панелей и панели приложения
502    private void hideSystemBars() {
503       if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
504          setSystemUiVisibility(
505             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
506             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
507             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN |
508             View.SYSTEM_UI_FLAG_HIDE_NAVIGATION |
509             View.SYSTEM_UI_FLAG_FULLSCREEN |
510             View.SYSTEM_UI_FLAG_IMMERSIVE);
511    }
512
513    // Вывести системные панели и панель приложения
514    private void showSystemBars() {
515       if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
516          setSystemUiVisibility(
517             View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
518             View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
519             View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
520    }
521 }
```

6.14. Резюме

В этой главе вы создали приложение `Cannon Game`, в котором игрок должен разрушить состоящую из семи секций мишень до истечения 10-секундного интервала. Пользователь наводит на цель и стреляет из пушки, касаясь экрана. Чтобы рисование выполнялось в отдельном потоке, мы создали отдельное представление расширением класса `SurfaceView`. Вы узнали о том, что имена классов пользовательских компонентов должны быть полностью заданы в элементе разметки XML, представляющем компонент. Мы рассмотрели дополнительные методы «жизненного цикла» класса `Activity`: метод `onPause` вызывается из текущей активности в случае, если другая деятельность становится активной, а метод `onDestroy` — при завершении активности системой. Переопределение метода `onTouchEvent` класса `Activity` было использовано для обработки касаний. Мы добавили звуковые эффекты в папку приложения `res/raw` и управляли ими с помощью класса `SoundPool`, а также использовали системную службу `AudioManager` для получения текущего значения громкости звучания музыки и использования этого значения для установки громкости воспроизведения звука.

Это приложение реализует анимации вручную, путем обновления элементов игры на `SurfaceView` из отдельного потока выполнения. Мы расширили класс `Thread` и создали метод `run`, который выводит графику с использованием методов класса `Canvas`. Мы использовали объект `SurfaceHolder` класса `SurfaceView` для получения объекта `Canvas`, а также научились создавать циклы, управляющие

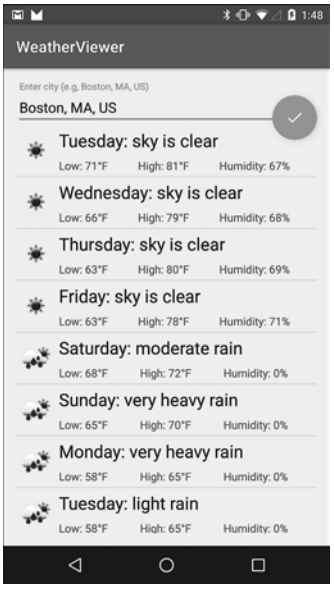
игрой на основе информации о времени, которое прошло между кадрами анимации. В результате игра выполняется с одной и той же скоростью на различных устройствах. Наконец, мы использовали режим погружения, для того чтобы приложение занимало весь экран.

В главе 7 мы построим приложение `WeatherViewer`. Наше приложение будет использовать веб-сервисы для обращения к 16-дневному прогнозу погоды на сайте *OpenWeatherMap.org*. Как и многие современные веб-сервисы, *OpenWeatherMap.org* возвращает данные прогнозов в формате *JSON* (JavaScript Object Notation). Полученные данные будут обрабатываться с помощью классов `JSONObject` и `JSONArray` из пакета `org.json`, после чего дневной прогноз будет отображаться в компоненте `ListView`.

7 Приложение WeatherViewer

REST-совместимые веб-сервисы, AsyncTask, HttpUrlConnection, обработка данных в формате JSON, JSONObject, JSONArray, ListView, ArrayAdapter, паттерн View-Holder, TextInputLayout, FloatingActionButton

В этой главе…



- Использование бесплатных веб-сервисов OpenWeatherMap.org для получения 16-дневного прогноза погоды для города, заданного пользователем
- Использование классов AsyncTask и HttpUrlConnection для обращения к веб-сервису или загрузки изображения в отдельном потоке с последующей передачей результата потоку GUI
- Обработка ответа в формате JSON с помощью классов JSONObject и JSONArray org.json
- Определение объекта ArrayAdapter, определяющего данные для отображения в ListView
- Применение паттерна View-Holder для повторного использования представлений, выходящих за границы экрана (вместо создания новых представлений)
- Использование компонентов материального дизайна TextInputLayout, Snackbar и FloatingActionButton из библиотеки Android Design Support Library

7.1. Введение

Приложение `WeatherViewer` (рис. 7.1) использует бесплатные REST-совместимые веб-сервисы *OpenWeatherMap.org* для получения 16-дневного прогноза погоды для заданного города. Приложение получает данные в формате *JSON* (JavaScript Object Notation). Результаты отображаются в `ListView` — компоненте для вывода списка с поддержкой прокрутки. В этом приложении будет использоваться пользовательский формат элементов списка:

- значок погодных условий,
- день недели с текстовым описанием погоды в этот день,
- самая высокая и самая низкая температура за день (по шкале Фаренгейта) и
- влажность в процентах.

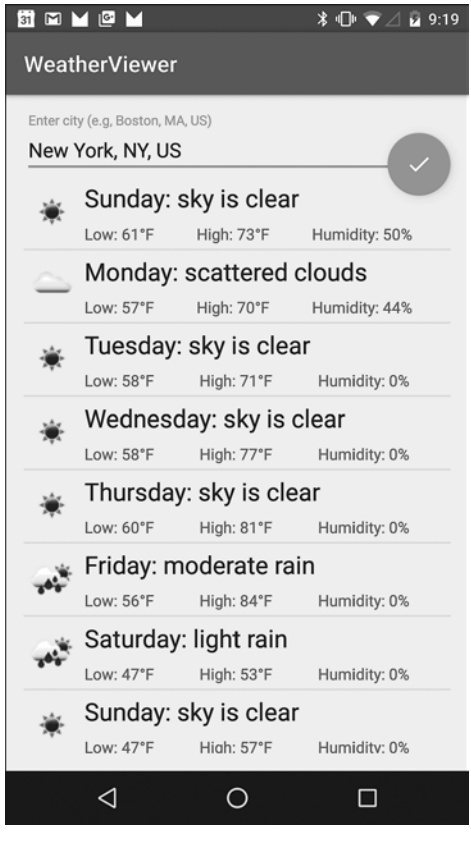


Рис. 7.1. Приложение WeatherViewer с прогнозом погоды для Нью-Йорка (США)

Эти элементы составляют небольшое подмножество возвращаемых данных прогноза. За подробной информацией о данных, возвращаемых API 16-дневного прогноза погоды, обращайтесь по адресу

http://openweathermap.org/forecast16

Полный список погодных API, предоставляемых *OpenWeatherMap.org*, доступен по адресу

http://openweathermap.org/api

7.2. Тестирование приложения WeatherViewer

Запустите Android Studio и откройте приложение `WeatherViewer` из папки `WeatherViewer` в примерах книги. Прежде чем запускать приложение, необходимо добавить собственный ключ API для *OpenWeatherMap.org*. О том, как получить ключ и где он должен храниться в проекте, рассказано в разделе 7.3.1. Этот шаг обязательно должен быть выполнен до запуска приложения. После того как ключ API будет добавлен в проект, запустите приложение в эмуляторе AVD или на устройстве.

Просмотр 16-дневного прогноза погоды

При запуске приложения фокус передается полю `EditText` в верхней части интерфейса пользователя, и на экране появляется виртуальная клавиатура для ввода названия города (рис. 7.2). Например, для получения прогноза для Нью-Йорка была введена строка `New York, NY, US`. После того как название города будет введено, коснитесь круглой кнопки `FloatingActionButton` со значком ✓ для передачи данных приложению. Приложение запрашивает 16-дневный прогноз погоды для выбранного вами города (рис. 7.2).

7.3. Обзор применяемых технологий

В этом разделе представлены новые технологии, которые будут использоваться для создания приложения `WeatherViewer`.

7.3.1. Веб-сервисы

В этой главе мы начнем использовать веб-сервисы, обеспечивающие портируемость и расширяющие возможности повторного использования кода в приложениях, работающих по Интернету. *Веб-сервис* представляет собой программный компонент, к которому можно обращаться по сети.

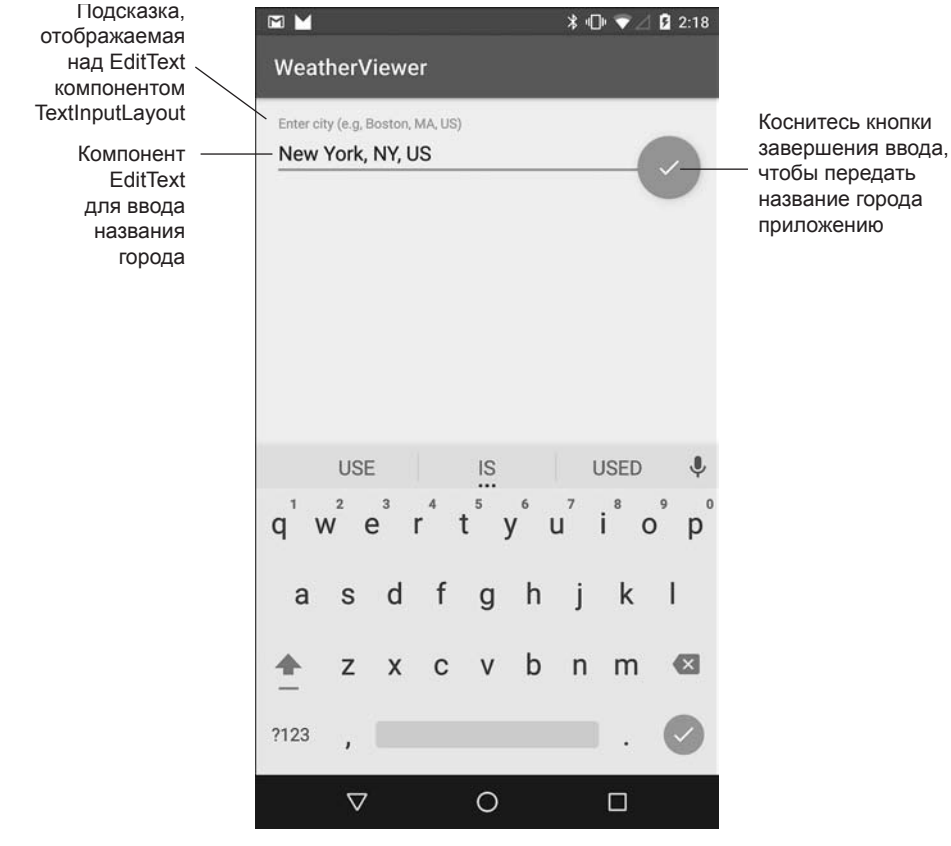


Рис. 7.2. Ввод названия города

Машина, на которой работает веб-сервис, называется *хостом*. *Клиент* — в данном случае приложение `WeatherViewer` — отправляет запрос хосту по сети. Хост обрабатывает запрос и возвращает ответ клиенту по сети. Распределенные вычисления обладают рядом преимуществ. Например, приложение может запрашивать данные по мере необходимости через веб-сервис, вместо того чтобы хранить их прямо на устройстве. Или, скажем, приложение в системе, не обеспечивающей необходимых вычислительных мощностей, может воспользоваться превосходящими ресурсами другой системы.

REST-совместимые веб-сервисы

Сокращением REST (Representational State Transfer) обозначается архитектурный стиль реализации веб-сервисов — такие сервисы часто называются «REST-совместимыми» (RESTful). Многие из популярных веб-сервисов — как бесплатных, так и коммерческих — являются REST-совместимыми. Хотя стиль

REST сам по себе не является стандартом, REST-совместимые веб-сервисы используют веб-стандарты — такие как протокол HTTP, используемый браузерами для взаимодействия с веб-серверами. Каждый метод REST-совместимого веб-сервиса представляется уникальным URL-адресом. Таким образом, когда сервер получает запрос, он немедленно узнает, какую операцию следует выполнить. Такие веб-сервисы могут использоваться в приложениях, но иногда их адреса можно вводить прямо в адресной строке браузера.

Веб-сервисам часто требуется ключ API

Для использования веб-сервиса часто необходимо получить уникальный ключ API у поставщика веб-сервиса. Когда ваше приложение обращается с запросом к веб-сервису, ключ API позволяет поставщику:

- убедиться в том, что вам разрешено использовать данный веб-сервис, и
- следить за интенсивностью использования — многие веб-сервисы ограничивают количество запросов за заданный промежуток времени (в секунду, в минуту, в час и т. д.).

Некоторые веб-сервисы требуют выполнения аутентификации перед получением ключа API — фактически вы должны подключиться к веб-сервису на программном уровне, прежде чем вам будет разрешено ее использовать.

Веб-сервисы OpenWeatherMap.org

Веб-сервисы *OpenWeatherMap.org*, которые будут использоваться в приложении `WeatherViewer`, бесплатны, но *OpenWeatherMap.org* ограничивает количество запросов к веб-сервису — в настоящее время эти ограничения составляют 1200 запросов в минуту и 1,7 миллиона запросов в день. Сервис *OpenWeatherMap.org* является условно-бесплатным — наряду с бесплатным уровнем, который будет использоваться в нашем приложении, существуют платные услуги с более высокими порогами запросов, более частыми обновлениями и другими возможностями. За дополнительной информацией о веб-сервисах *OpenWeatherMap.org* обращайтесь по адресу:

http://openweathermap.org/api

Лицензия на использование веб-сервиса OpenWeatherMap.org

Доступ к веб-сервисам *OpenWeatherMap.org* предоставляется на условиях открытой лицензии Creative Commons. За условиями лицензии обращайтесь по адресу

http://creativecommons.org/licenses/by-sa/2.0/

Дополнительная информация об условиях лицензии доступна по адресу

http://openweathermap.org/terms

Получение ключа API OpenWeatherMap.org

Прежде чем запускать это приложение, необходимо получить собственный ключ API OpenWeatherMap.org по адресу

http://openweathermap.org/register

После регистрации скопируйте шестнадцатеричный ключ API со страницы подтверждения, а затем замените строку `YOUR_API_KEY` в файле `strings.xml` скопированным ключом.

7.3.2. Формат JSON и пакет org.json

Формат JSON (JavaScript Object Notation) может использоваться для представления данных вместо XML. JSON — текстовый формат обмена данных, используемый для представления объектов в JavaScript в виде коллекций пар «имя—значение», представленных в строковом виде. Это простой формат, в котором легко создавать, читать и разбирать объекты; к тому же он существенно компактнее XML, что повышает эффективность передачи данных по Интернету. Каждый объект JSON представляется в виде списка имен и значений свойств, заключенных в фигурные скобки, в следующем формате:

{свойствоИмя1: значение1, свойствоИмя2: значение2}

Каждое имя свойства представляет собой `String`. Массивы в JSON заключаются в квадратные скобки в следующем формате:

[значение1, значение2, значение3]

Каждый элемент массива может быть строкой, числом, объектом JSON, `true`, `false` или `null`.

В листинге 7.1 приведен пример данных в формате JSON, возвращаемых сервисом прогноза погоды *OpenWeatherMap.org*. Этот конкретный пример содержит погодные данные за два дня (строки 15–57).

Листинг 7.1. Пример JSON из OpenWeatherMap.org — веб-сервиса прогноза погоды

```
 1 {
 2    "city": {
 3       "id": 5128581,
 4       "name": "New York",
 5       "coord": {
 6          "lon": -74.005966,
 7          "lat": 40.714272
 8       },
 9       "country": "US",
10       "population": 0
11    },
12    "cod": "200",
13    "message": 0.0102,
14    "cnt": 2,
```

```
{
   "list": [{ // Мы будем использовать этот массив объектов
       "dt": 1442419200,
       "temp": {
         "day": 79.9,
          "min": 71.74,
          "max": 82.53,
         "night": 71.85,
         "eve": 82.53,
         "morn": 71.74
      },
      "pressure": 1037.39,
      "humidity": 64,
      "weather": [{
         "id": 800,
         "main": "Clear",
         "description": "sky is clear",
         "icon": "01d"
      }],
      "speed": 0.92,
      "deg": 250,
      "clouds": 0
   }, { // Конец первого элемента массива, начало второго
      "dt": 1442505600,
      "temp": {
         "day": 79.92,
         "min": 66.72,
         "max": 83.1,
         "night": 70.79,
         "eve": 81.99,
         "morn": 66.72
      },
      "pressure": 1032.46,
      "humidity": 62,
      "weather": [{
         "id": 800,
         "main": "Clear",
         "description": "sky is clear",
         "icon": "01d"
      }],
      "speed": 1.99,
      "deg": 224,
      "clouds": 0
   }] // Конец второго элемента и конец массива
}
```

Объект JSON, возвращаемый в прогнозе погоды, содержит много свойств. Мы будем использовать только свойство `"list"` — массив объектов JSON, представляющих прогнозы на время до 16 дней (7 по умолчанию, если в запросе не указано обратное). Каждый элемент массива `"list"` также содержит много свойств, из которых мы будем использовать:

- `"dt"` — целое значение типа `long` с временной меткой, представленной в виде количества секунд, прошедших с 1 января 1970 года (GMT). Мы преобразуем это значение в название дня;

- `"temp"` — объект JSON со свойствами `double`, представляющими дневные температуры. В приложении используется только минимальная (`"min"`) и максимальная (`"max"`) температуры, но веб-сервис также возвращает среднюю дневную (`"day"`), ночную (`"night"`), вечернюю (`"eve"`) и утреннюю (`"morn"`) температуру;
- `"humidity"` — значение `int`, представляющее влажность в процентах;
- `"weather"` — объект JSON с несколькими свойствами, включая описание погодных условий (`"description"`) и имя значка, представляющего условия (`"icon"`).

Пакет org.json

Для обработки данных JSON, получаемых приложением, будут использоваться следующие классы из пакета `org.json` (раздел 7.7.6):

- `JSONObject` — один из конструкторов класса преобразует строку данных JSON в объект `JSONObject`, который содержит коллекцию `Map<String, Object>`, связывающую ключи JSON со значениями. Для обращения к свойствам JSON в коде используются методы `get` класса `JSONObject`, что позволяет получить значение, ассоциированное с ключом, в одном из типов `JSONObject`, `JSONArray`, `Object`, `boolean`, `double`, `int`, `long` или `String`;
- `JSONArray` — класс представляет массив данных JSON и предоставляет методы для обращения к его элементам. В приложении `WeatherViewer` свойство `"list"` ответа *OpenWeatherMap.org* будет обрабатываться как `JSONArray`.

7.3.3. HttpUrlConnection и обращение к REST-совместимым веб-сервисам

Чтобы передать вызов веб-сервису прогноза погоды *OpenWeatherMap.org*, мы преобразуем строку URL-адреса веб-сервиса в объект `URL`, а затем используем `URL` для открытия `HttpUrlConnection` (раздел 7.7.5). При этом выдается запрос HTTP к веб-сервису. Чтобы получить ответ JSON, мы прочитаем все данные из потока `InputStream` объекта `HttpUrlConnection` и поместим их в `String`. Мы покажем, как преобразовать их в `JSONObject` для обработки.

7.3.4. Использование AsyncTask для обработки сетевых запросов вне потока GUI

Продолжительные операции, а также операции, которые блокируют выполнение приложения до их завершения (сетевые операции, обращения к файлам

и базам данных), должны выполняться вне потока графического интерфейса. Тем самым обеспечивается быстрый отклик приложения и предотвращается появление диалоговых окон ANR (Activity Not Responding), которые выводятся системой Android, если она считает, что графический интерфейс перестал реагировать на действия пользователя. Однако вспомните, о чем говорилось в главе 6: обновления интерфейса пользователя должны происходить в потоке GUI, потому что компоненты GUI не обладают потоковой безопасностью.

Для выполнения долгих операций, результатом которых является обновление графического интерфейса, Android предоставляет класс `AsyncTask` (пакет `android.os`). Этот класс выполняет продолжительную операцию в одном потоке и передает результаты потоку GUI. Класс `AsyncTask` берет на себя все подробности создания и управления потоками, как и передачу результатов от `AsyncTask` потоку GUI. В приложении будут использоваться два субкласса `AsyncTask` — один будет обращаться с вызовом к веб-сервису *OpenWeatherMap.org* (раздел 7.7.5), а другой загружает изображение, представляющее погодные условия (раздел 7.6.5).

7.3.5. ListView, ArrayAdapter и паттерн View-Holder

Приложение выводит погодные данные в компоненте `ListView` (пакет `android.widget`) — списке с поддержкой прокрутки. Класс `ListView` является субклассом класса `AdapterView` (пакет `android.widget`), представляющего получение данных от источника через объект `Adapter` (пакет `android.widget`). В этом приложении субкласс `ArrayAdapter` (пакет `android.widget`) используется для создания объекта, заполняющего `ListView` данными из объекта коллекции `ArrayList` (раздел 7.6). Когда приложение обновляет `ArrayList` погодными данными, мы вызываем метод `notifyDataSetChanged` класса `ArrayAdapter`, чтобы сообщить об изменении данных в `ArrayList`. Адаптер сообщает `ListView` о необходимости изменить список отображаемых данных. Этот механизм называется *связыванием данных* (data binding). Существует несколько типов `AdapterView`, которые могут связываться с данными с использованием адаптера. В главе 9 вы узнаете, как связать компонент `ListView` с информацией из базы данных. Дополнительную информацию о связывании данных в Android, а также некоторые обучающие примеры можно найти по адресу

http://developer.android.com/guide/topics/ui/binding.html

Паттерн View-Holder

По умолчанию компонент `ListView` может отображать один или два компонента `TextView`. В этом приложении мы настроим элементы `ListView` так, чтобы в них отображался компонент `ImageView` и несколько компонентов `TextView`

в пользовательском (нестандартном) макете. Создание пользовательских элементов `ListView` сопряжено с высокими затратами ресурсов на динамическое создание новых объектов. В больших списках со сложными макетами элементов, в которых пользователь быстро прокручивает содержимое, это может привести к нарушению плавности прокрутки. Для сокращения затрат ресурсов при выходе элементов за границы экрана Android повторно использует элементы списка для новых объектов, появляющихся на экране.

В паттерне `View-Holder` создается класс (обычно с именем `ViewHolder`), который содержит переменные экземпляров для представлений, отображающих данные элементов `ListView`. При создании элемента `ListView` также создается объект `ViewHolder`, переменные которого инициализируются ссылками на вложенные представления элемента. Затем объект `ViewHolder` сохраняется вместе с элементом `ListView`, который относится к классу `View`. Метод `setTag` класса `View` позволяет добавить к `View` произвольный объект. В дальнейшем этот объект может быть получен методом `getTag` класса `View`. С этими вызовами используется объект `ViewHolder`, содержащий ссылки на вложенные представления элемента `ListView`.

Когда новый элемент готовится к появлению на экране, компонент `ListView` проверяет наличие представления, доступного для повторного использования. Если такого представления нет, мы заполняем представление нового элемента на основании XML-файла макета, а затем сохраняем ссылки на компоненты GUI в объекте `ViewHolder`. Затем этот объект `ViewHolder` связывается с элементом `ListView` вызовом `setTag`. Если элемент, пригодный для повторного использования, существует, мы получаем ассоциированный с ним объект `ViewHolder` вызовом `getTag`; вызов возвращает существующий объект `ViewHolder`, созданный ранее для элемента `ListView`. Независимо от того, как был получен объект `ViewHolder`, данные выводятся в представлениях, ссылки на которые хранятся в `ViewHolder`.

7.3.6. FloatingActionButton

Пользователи нажимают кнопки для выполнения операций. С появлением концепции материального дизайна в Android 5.0 компания Google представила кнопку `FloatingActionButton`, «плавающую» над интерфейсом пользователя (иначе говоря, эта кнопка находится на большей высоте, чем остальные элементы интерфейса) и обозначающую некоторое важное действие. Например, в адресной книге плавающая кнопка со значком + может выделять действие добавления нового контакта. В нашем приложении плавающая кнопка со значком ✓ будет использоваться для передачи приложению названия города и получения прогноза для этого города. С выходом Android 6.0 и новой библиотеки Android Design Support Library компания Google формализовала плавающую

кнопку в классе FloatingActionButton (пакет android.support.design.widget). В Android Studio 1.4 были введены новые реализации шаблонов приложений, использующие материальный дизайн, и во многие новые шаблоны кнопка FloatingActionButton включалась по умолчанию.

Класс FloatingActionButton является субклассом ImageView, что позволяет выводить изображение на FloatingActionButton. Рекомендации материального дизайна советуют размещать кнопки FloatingActionButton по крайней мере в 16dp от границ экрана на телефонах и по крайней мере в 24dp от границ экрана на планшетах — шаблоны по умолчанию настраивают эти интервалы за вас. За дополнительной информацией о том, когда и как следует использовать кнопку FloatingActionButton, обращайтесь по адресу

https://www.google.com/design/spec/components/buttons-floating-action-button.html

7.3.7. Компонент TextInputLayout

В этом приложении компонент EditText будет использоваться для ввода названия города, для которого запрашивается прогноз погоды. Чтобы пользователю было проще понять назначение EditText, можно создать подсказку, которая будет отображаться, пока в EditText нет символов. Когда пользователь начинает вводить текст, подсказка исчезает — и пользователь может забыть, для чего нужно данное поле.

Компонент TextInputLayout из библиотеки поддержки Android (пакет android.support.design.widget) решает эту проблему. При получении фокуса EditText компонент TextInputLayout уменьшает текст подсказки и размещает его над EditText, чтобы подсказка была видна и в процессе ввода данных (см. рис. 7.2). В нашем приложении EditText получает фокус в начале выполнения приложения, так что TextInputLayout немедленно перемещает подсказку над EditText.

7.3.8. Компонент Snackbar

Компонент материального дизайна Snackbar (пакет android.support.design.widget) по своему назначению напоминает временные оповещения Toast. Он не только появляется на экране на заданный промежуток времени, но и поддерживает интерактивные взаимодействия. Пользователь может провести пальцем по экрану, чтобы закрыть оповещение. С компонентом Snackbar также может быть связано действие, выполняемое при касании Snackbar. В приложении WeatherViewer компонент Snackbar будет использоваться для вывода информационных сообщений.

7.4. Создание графического интерфейса приложения и файлов ресурсов

В этом разделе рассматриваются новые возможности графического интерфейса и файлов ресурсов приложения `WeatherViewer`.

7.4.1. Создание проекта

Создайте новый проект на базе шаблона `Blank Activity`. На шаге `New Project` диалогового окна `Create New Project` укажите следующие значения:

- `Application name`: `WeatherViewer`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

На остальных шагах диалогового окна `Create New Project` используйте те же настройки, что и в разделе 2.3. Также добавьте в проект значок приложения так, как описано в разделе 2.5.2, и настройте поддержку Java SE 7 в соответствии с разделом 4.4.3.

7.4.2. AndroidManifest.xml

Приложение `WeatherViewer` рассчитано на работу только в портретной ориентации. Задайте свойству `android:screenOrientation` значение `portrait` (см. описание в разделе 3.7). Кроме того, добавьте в элемент `<manifest>` разрешение на доступ к Интернету перед вложенным элементом `<application>`:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Тем самым вы разрешаете приложению доступ к Интернету, необходимый для обращения к веб-сервису.

Разрешения, автоматически предоставляемые в Android 6.0

Новая модель разрешений Android 6.0 (см. главу 6) автоматически предоставляет разрешение на доступ к Интернету во время установки, так как доступ к Интернету считается неотъемлемой частью современных приложений. В Android 6.0 это и другие разрешения, которые, по мнению Google, «не создают большого риска для конфиденциальности или безопасности пользователя», предоставляются автоматически во время установки — эти разрешения объединяются в категорию `PROTECTION_NORMAL`. Полный список таких разрешений доступен по адресу

https://developer.android.com/preview/features/runtime-permissions.html#best-practices

Android не предлагает пользователям предоставлять такие разрешения, и пользователи не могут отзывать их у приложений. По этой причине вашему коду не нужно проверять, обладает ли приложение разрешением `PROTECTION_NORMAL`. Тем не менее вы все равно должны запросить эти разрешения в файле `AndroidManifest.xml` для обеспечения совместимости с более ранними версиями Android.

7.4.3. strings.xml

Сделайте двойной щелчок на файле `strings.xml` из папки `res/values` и щелкните на ссылке `Open editor`, чтобы запустить редактор `Translations Editor` для создания строковых ресурсов из табл. 7.1.

Таблица 7.1. Строковые ресурсы, используемые в приложении WeatherViewer

Ключ	Значение
api_key	Используйте свое значение ключа API
web_service_url	http://api.openweathermap.org/data/2.5/forecast/daily?q=
invalid_url	Invalid URL
weather_condition_image	A graphical representation of the weather conditions
high_temp	High: %s
low_temp	Low: %s
day_description	%1$s: %2$s
humidity	Humidity: %s
hint_text	Enter city (e.g, Boston, MA, US)
read_error	Unable to read weather data
connect_error	Unable to connect to OpenWeatherMap.org

7.4.4. colors.xml

Шаблон Android Studio `Blank Activity` настраивает основной цвет приложения, темный основной и акцентный цвета. В этом приложении следует заменить акцентный цвет шаблона (`colorAccent`) синим оттенком (шестнадцатеричный код `#448AFF`) в файле `colors.xml`.

7.4.5. activity_main.xml

Шаблон Android Studio `Blank Activity` разбивает графический интерфейс `MainActivity` на два файла:

- `activity_main.xml` определяет объект `Toolbar` (замена панели приложения для `AppCompatActivity`) и объект `FloatingActionButton`, по умолчанию находящийся в правом нижнем углу;
- `content_main.xml` определяет оставшуюся часть графического интерфейса `MainActivity` и включаетсяч в файл `activity_main.xml` при помощи элемента `<include>`.

Внесите в файл `activity_main.xml` следующие изменения.

1. Задайте свойству `id` компонента `CoordinatorLayout` значение `coordinator-Layout` — оно будет использоваться для определения макета, в котором должен отображаться компонент `Snackbar`.
2. Добавьте в проект кнопку `Done` (✓) из Vector Asset Studio (как это было сделано в разделе 4.4.9), затем задайте этот новый значок свойству `src` компонента `FloatingActionButton`.
3. Отредактируйте разметку XML макета, чтобы настроить некоторые свойства `FloatingActionButton`, отсутствующие в окне свойств. Замените значение `layout_gravity` с `bottom|end` на `top|end`, чтобы кнопка `FloatingActionButton` отображалась в правом верхнем углу интерфейса пользователя.
4. Чтобы кнопка перекрывала правую сторону `EditText`, определите новый ресурс метрики с именем `fab_margin_top` и значением `90dp`. Используйте этот ресурс и ресурс `fab_margin`, определяемый шаблоном `Blank Activity`, для определения следующих отступов `FloatingActionButton`:

```
android:layout_marginTop="@dimen/fab_margin_top"
android:layout_marginEnd="@dimen/fab_margin"
android:layout_marginBottom="@dimen/fab_margin"
android:layout_marginStart="@dimen/fab_margin"
```

5. Удалите значение `layout_margin` компонента `FloatingActionButton`, определенное шаблоном `Blank Activity`.

7.4.6. content_main.xml

Макет включается в файл `activity_main.xml` и определяется основной графический интерфейс `MainActivity`. Выполните следующие действия.

1. Удалите компонент `TextView`, включаемый по умолчанию шаблоном `Blank Activity`, и замените компонент `RelativeLayout` вертикальным компонентом `LinearLayout`.
2. Вставьте компонент `TextInputLayout`. В режиме представления `Design` макетного редактора щелкните на компоненте `CustomView` в разделе `Custom`. В открывшемся диалоговом окне начните вводить имя `TextInputLayout`, чтобы провести поиск по списку компонентов GUI. Когда IDE выделит компонент `TextInputLayout`, нажмите кнопку `OK`, а затем в окне `Component Tree` щелкните на компоненте `LinearLayout`, чтобы вставить `TextInputLayout` как вложенный макет.
3. Чтобы добавить компонент `EditText` в `TextInputLayout`, перейдите на вкладку `Text` макетного редактора, замените символы `/>` в конце элемента `TextInputLayout` на `>`, установите курсор справа от `>`, нажмите Enter и введите `</`. IDE автоматически вставляет завершающий тег. Между начальным и конечным тегами `TextInputLayout` введите `<EditText`. IDE открывает окно автозаполнения с выделенным компонентом `EditText`. Нажмите Enter, чтобы вставить `EditText`, задайте его свойству `layout_width` значение `match_parent`, а свойству `layout_height` значение `wrap_content`. В режиме `Design` задайте свойству `id` компонента `EditText` значение `locationEditText`, установите флажок свойства `singleLine` и задайте свойству `hint` строковый ресурс `hint_text`.
4. Чтобы завершить построение макета, перетащите компонент `ListView` на `LinearLayout` в окне `Component Tree`. Задайте его свойству `layout:width` значение `match_parent`, свойству `layout:height` — значение `0dp`, свойству `layout:weight` — значение 1 и свойству `id` — значение `weatherListView`. Напомним, что IDE рекомендует использовать значение `0dp` свойства `layout:height` для ускорения прорисовки, когда высота представления вычисляется с использованием `layout:weight`.

7.4.7. list_item.xml

Теперь мы добавим в проект макет `list_item.xml` и определим пользовательский макет для отображения погодных данных в элементе `ListView` (рис. 7.3). Этот макет будет заполняться классом `WeatherArrayAdapter` для создания интерфейса новых элементов `ListView` (раздел 7.6.4).

Шаг 1: Создание файла макета и настройка ориентации LinearLayout

Выполните следующие действия, чтобы создать файл макета `list_item.xml`.

1. Щелкните правой кнопкой мыши на папке `layout` проекта и выберите команду `New ▸ Layout resource file`.

Рис. 7.3. Макет прогноза погоды на день в элементе ListView

2. Введите текст `list_item.xml` в поле `File name` диалогового окна `New Resource File`.
3. Убедитесь в том, что в поле `Root element` выбрано значение `LinearLayout`, и нажмите кнопку `OK`. Файл `list_item.xml` появляется в папке `layout` окна `Project` и открывается в макетном редакторе.
4. Выберите компонент `LinearLayout` и задайте его свойству `orientation` значение `horizontal` — макет будет состоять из компонентов `ImageView` и `GridLayout`, содержащих другие представления.

Шаг 2: Добавление ImageView для значка погодных условий

Выполните следующие действия, чтобы добавить и настроить компонент `ImageView`.

1. Перетащите компонент `ImageView` с палитры на компонент `LinearLayout` в окне `Component Tree`.
2. Задайте свойству `id` значение `conditionImageView`.
3. Задайте `layout:width` значение `50dp` — определите для этого значения ресурс метрики `image_side_length`.
4. Задайте `layout:height` значение `match_parent` — высота `ImageView` должна определяться высотой элемента `ListView`.
5. Задайте свойству `contentDescription` строковый ресурс `weather_condition_image`, созданный на шаге 7.4.3.
6. Задайте свойству `scaleType` значение `fitCenter` — значок будет размещаться в границах `ImageView` с выравниванием по центру (по горизонтали и вертикали).

Шаг 3: Добавление GridLayout для отображения TextView

Выполните следующие действия, чтобы добавить и настроить компонент `GridLayout`.

1. Перетащите компонент `GridLayout` с палитры на компонент `LinearLayout` в окне `Component Tree`.
2. Задайте свойству `columnCount` значение 3, а свойству `rowCount` — значение 2.
3. Задайте `layout:width` значение `0dp` — ширина `GridLayout` должна определяться `layout:weight`.
4. Задайте `layout:height` значение `match_parent` — высота `GridLayout` должна определяться высотой элемента `ListView`.
5. Задайте `layout:weight` значение 1 — компонент `GridLayout` должен занимать все оставшееся горизонтальное пространство в родительском компоненте `LinearLayout`.
6. Установите флажок `useDefaultMargins`, чтобы ячейки `GridLayout` разделялись стандартными интервалами.

Шаг 4: Добавление TextView

Выполните следующие действия, чтобы добавить и настроить четыре компонента `TextView`.

1. Перетащите компонент `Large Text` на `GridLayout` в окне `Component Tree`. Задайте его свойству `id` значение `dayTextView`, свойству `layout:column` — значение 0, а свойству `layout:columnSpan` — значение 3.
2. Перетащите три компонента `Plain TextView` на `GridLayout` в окне `Component Tree` и задайте их свойствам `id` значения `lowTextView`, `hiTextView` и `humidityTextView` соответственно. Задайте свойству `layout:row` всех четырех `TextView` значение 1, а свойству `layout:columnWeight` — значение 1. Эти компоненты `TextView` будут отображаться во второй строке `GridLayout`, а поскольку все они имеют одинаковые значения `layout:columnWeight`, столбцы будут иметь одинаковые размеры.
3. Задайте свойству `layout:column` компонента `lowTextView` значение 0, свойству `layout:column` компонента `hiTextView` — значение 1 и свойству `layout:column` компонента `humidityTextView` — значение 2.

На этом построение макета `list_item.xml` завершено. Изменять свойство `text` компонентов `TextView` не нужно — содержащийся в них текст будет определяться на программном уровне.

7.5. Класс Weather

Приложение состоит из трех классов, описания которых приведены в разделах 7.5–7.7.

- Класс `Weather` (этот раздел) представляет погодные данные за один день. Класс `MainActivity` преобразует погодные данные в формате JSON в `ArrayList<Weather>`.
- Класс `WeatherArrayAdapter` (раздел 7.6) определяет пользовательский субкласс `ArrayAdapter` для связывания данных `ArrayList<Weather>` с компонентом `ListView` из `MainActivity`. Индексы элементов `ListView` начинаются с 0, а все вложенные представления каждого элемента `ListView` заполняются данными из объекта `Weather` с соответствующим индексом из `ArrayList<Weather>`.
- Класс `MainActivity` (раздел 7.7) определяет пользовательский интерфейс приложения, логику взаимодействия с веб-сервисом *OpenWeatherMap.org* и обработки ответа JSON.

В этом разделе рассматривается класс `Weather`.

7.5.1. Команды package, import и переменные экземпляров

В листинге 7.2 представлены команды `package` и `import`, а также переменные экземпляров класса `Weather`. Классы из пакетов `java.text` и `java.util` (строки 5–8) используются для преобразования временной метки каждого дня в название дня недели. Переменные экземпляров объявлены с ключевым словом `final`, потому что они не должны изменяться после инициализации. Переменные также были объявлены открытыми (`public`) — напомним, что объекты `String` в Java неизменяемы (immutable), поэтому несмотря на такое объявление, их значения измениться не могут.

Листинг 7.2. Команда package, import и переменные экземпляров класса Weather

```
1 // Weather.java
2 // Информация о погоде за один день
3 package com.deitel.weatherviewer;
4
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.Calendar;
8 import java.util.TimeZone;
9
10 class Weather {
11    public final String dayOfWeek;
```

```
12    public final String minTemp;
13    public final String maxTemp;
14    public final String humidity;
15    public final String description;
16    public final String iconURL;
17
```

7.5.2. Конструктор

Конструктор `Weather` (листинг 7.3) инициализирует поля класса.

- Объект `NumberFormat` создает объекты `String` из числовых значений. В строках 22–23 этот объект настраивается для округления вещественных значений до целых.
- В строке 25 вызывается вспомогательный метод `convertTimeStampToDay` (раздел 7.5.3) для получения названия дня недели и инициализации `dayOfWeek`.
- Строки 26–27 форматируют минимальную и максимальную температуру в виде целых чисел при помощи объекта `numberFormat`. К концу каждой отформатированной строки присоединяется суффикс `°F`, так как приложение должно выводить температуру по шкале Фаренгейта — служебная последовательность `\u00B0` в Юникоде обозначает символ градуса (°). *OpenWeatherMap.org* API также поддерживает шкалы Кельвина (по умолчанию) и Цельсия.
- Строки 28–29 получают объект `NumberFormat` для форматирования процентов по правилам локального контекста, а затем используют его для форматирования влажности. Веб-сервис возвращает процент в виде целого числа, поэтому значение делится на 100 для форматирования — по правилам локального контекста США значение 1,00 форматируется как 100%, 0,5 — как 50% и т. д.
- Строка 30 инициализирует описание погодных условий.
- В строках 31–32 создается строка с URL-адресом, представляющим изображение погодных условий для погоды, — она будет использоваться для загрузки изображения.

Листинг 7.3. Конструктор класса Weather

```
18    // Конструктор
19    public Weather(long timeStamp, double minTemp, double maxTemp,
20       double humidity, String description, String iconName) {
21       // NumberFormat для форматирования температур в целое число
22       NumberFormat numberFormat = NumberFormat.getInstance();
23       numberFormat.setMaximumFractionDigits(0);
24
25       this.dayOfWeek = convertTimeStampToDay(timeStamp);
26       this.minTemp = numberFormat.format(minTemp) + "\u00B0F";
27       this.maxTemp = numberFormat.format(maxTemp) + "\u00B0F";
```

```
28       this.humidity =
29          NumberFormat.getPercentInstance().format(humidity / 100.0);
30       this.description = description;
31       this.iconURL =
32          "http://openweathermap.org/img/w/" + iconName + ".png";
33    }
34
```

7.5.3. Метод convertTimeStampToDay

В аргументе вспомогательного метода `convertTimeStampToDay` (листинг 7.4) передается длинное целое число, представляющее количество секунд, прошедших с 1 января 1970 года (GMT), — стандартное представление времени в системах Linux (система Android создана на базе Linux). Преобразование выполняется следующим образом.

- Строка 37 получает объект `Calendar` для работы с датой и временем, после чего в строке 38 вызывается метод `setTimeInMillis` для инициализации времени с использованием аргумента `timestamp`. Значение `timestamp` задается в секундах, поэтому оно умножается на 1000 для преобразования в миллисекунды.
- Строка 39 задает объект `TimeZone` по умолчанию. Этот объект используется для внесения поправки в значение времени с учетом часового пояса устройства (строки 42–43).
- Строка 46 создает объект `SimpleDateFormat` для форматирования объекта `Date`. Аргумент конструктора `"EEEE"` форматирует как дату, так и название дня недели (`Monday`, `Tuesday` и т. д.). Полный список форматов доступен по адресу

  http://developer.android.com/reference/java/text/SimpleDateFormat.html
- Строка 47 форматирует и возвращает название дня недели. Метод `getTime` класса `Calendar` возвращает объект `Date`, содержащий время. Этот объект `Date` передается методу `format` класса `SimpleDateFormat` для получения названия дня.

Листинг 7.4. Метод convertTimeStampToDay

```
35    // Преобразование временной метки в название дня недели (Monday, ...)
36    private static String convertTimeStampToDay(long timeStamp) {
37       Calendar calendar = Calendar.getInstance(); // Объект Calendar
38       calendar.setTimeInMillis(timeStamp * 1000); // Получение времени
39       TimeZone tz = TimeZone.getDefault(); // Часовой пояс устройства
40
41       // Поправка на часовой пояс устройства
42       calendar.add(Calendar.MILLISECOND,
43          tz.getOffset(calendar.getTimeInMillis()));
44
45       // Объект SimpleDateFormat, возвращающий название дня недели
46       SimpleDateFormat dateFormatter = new SimpleDateFormat("EEEE");
```

```
47       return dateFormatter.format(calendar.getTime());
48    }
49 }
```

7.6. Класс WeatherArrayAdapter

Класс `WeatherArrayAdapter` определяет субкласс `ArrayAdapter` для связывания `ArrayList<Weather>` с компонентом `ListView` активности `MainActivity`.

7.6.1. Команды package и import

В листинге 7.5 содержатся команды `package` и `import` класса `WeatherArrayAdapter`. Импортируемые типы будут рассматриваться по мере того, как они будут встречаться в тексте.

Элементы `ListView` этого приложения используют нестандартный макет. Каждый элемент списка содержит изображение (значок погодных условий) и текст с описанием дня, погоды, минимальной температуры, максимальной температуры и влажности. Чтобы связать погодные данные с элементами списка `ListView`, мы расширяем класс `ArrayAdapter` (строка 23); это позволит переопределить метод `getView` класса `ArrayAdapter` для создания пользовательского макета для каждого элемента `ListView`.

Листинг 7.5. Команды package и import класса WeatherArrayAdapter

```
 1 // WeatherArrayAdapter.java
 2 // Объект ArrayAdapter для отображения элементов List<Weather> в ListView
 3 package com.deitel.weatherviewer;
 4
 5 import android.content.Context;
 6 import android.graphics.Bitmap;
 7 import android.graphics.BitmapFactory;
 8 import android.os.AsyncTask;
 9 import android.view.LayoutInflater;
10 import android.view.View;
11 import android.view.ViewGroup;
12 import android.widget.ArrayAdapter;
13 import android.widget.ImageView;
14 import android.widget.TextView;
15
16 import java.io.InputStream;
17 import java.net.HttpURLConnection;
18 import java.net.URL;
19 import java.util.HashMap;
20 import java.util.List;
21 import java.util.Map;
22
23 class WeatherArrayAdapter extends ArrayAdapter<Weather> {
```

7.6.2. Вложенный класс ViewHolder

Вложенный класс `ViewHolder` (листинг 7.6) определяет переменные экземпляров, к которым класс `WeatherArrayAdapter` обращается напрямую при работе с объектами `ViewHolder`. При создании элемента `ListView` мы связываем с ним новый объект `ViewHolder`. Если имеется существующий элемент `ListView`, который может использоваться повторно, мы просто получаем объект `ViewHolder` этого элемента.

Листинг 7.6. Вложенный класс ViewHolder

```
24    // Класс для повторного использования представлений списка при прокрутке
25    private static class ViewHolder {
26       ImageView conditionImageView;
27       TextView dayTextView;
28       TextView lowTextView;
29       TextView hiTextView;
30       TextView humidityTextView;
31    }
32
```

7.6.3. Переменные экземпляров и конструктор

В листинге 7.7 определяется переменная экземпляра класса `WeatherArrayAdapter` и конструктор. Переменная экземпляра `bitmaps` (строка 34) — `Map<String, Bitmap>` — используется для кэширования ранее загруженных изображений погодных условий, чтобы их не приходилось загружать заново при прокрутке прогноза. Кэшированные изображения остаются в памяти до тех пор, пока Android не завершит приложение. Конструктор (строки 37–39) просто вызывает конструктор суперкласса с тремя аргументами; в первом и третьем аргументах передаются объект `Context` (то есть активность, в которой отображается `ListView`) и `List<Weather>` (список выводимых данных). Второй аргумент конструктора суперкласса представляет идентификатор ресурса макета, содержащего компонент `TextView`, в котором отображаются данные `ListView`. Аргумент –1 означает, что в приложении используется пользовательский макет, чтобы элемент списка не ограничивался одним компонентом `TextView`.

Листинг 7.7. Поле класса WeatherArrayAdapter и конструктор

```
33    // Кэш для уже загруженных объектов Bitmap
34    private Map<String, Bitmap> bitmaps = new HashMap<>();
35
36    // Конструктор для инициализации унаследованных членов суперкласса
37    public WeatherArrayAdapter(Context context, List<Weather> forecast) {
38       super(context, -1, forecast);
39    }
40
```

7.6.4. Переопределенный метод getView

Метод `getView` (листинг 7.8) вызывается для получения объекта `View`, используемого для отображения данных элемента `ListView`. Переопределение этого метода позволяет связать данные с нестандартным элементом `ListView`. В аргументах метод получает позицию элемента `ListView`, элемент `View` (`convertView`), представляющий элемент `ListView`, и родителя элемента `ListView`. Выполняя необходимые операции с `convertView`, вы можете настроить содержимое элемента `ListView`. В строке 45 унаследованный от `ArrayAdapter` метод `getItem` вызывается для получения от `List<Weather>` отображаемого объекта `Weather`.

В строке 47 определяется переменная `ViewHolder`, которой будет присвоен новый или существующий объект `ViewHolder` — в зависимости от того, равен ли `null` аргумент `convertView` метода `getView`.

Листинг 7.8. Поле класса WeatherArrayAdapter и конструктор

```
41    // Создание пользовательских представлений для элементов ListView
42    @Override
43    public View getView(int position, View convertView, ViewGroup parent)  {
44       // Получение объекта Weather для заданной позиции ListView
45       Weather day = getItem(position);
46
47       ViewHolder viewHolder; // Объект, содержащий ссылки
48                              // на представления элемента списка
49       // Проверить возможность повторного использования ViewHolder
50       // для элемента, вышедшего за границы экрана
51       if (convertView == null) { // Объекта ViewHolder нет, создать его
52          viewHolder = new ViewHolder();
53          LayoutInflater inflater = LayoutInflater.from(getContext());
54          convertView =
55             inflater.inflate(R.layout.list_item, parent, false);
56          viewHolder.conditionImageView =
57             (ImageView) convertView.findViewById(R.id.conditionImageView);
58          viewHolder.dayTextView =
59             (TextView) convertView.findViewById(R.id.dayTextView);
60          viewHolder.lowTextView =
61             (TextView) convertView.findViewById(R.id.lowTextView);
62          viewHolder.hiTextView =
63             (TextView) convertView.findViewById(R.id.hiTextView);
64          viewHolder.humidityTextView =
65             (TextView) convertView.findViewById(R.id.humidityTextView);
66          convertView.setTag(viewHolder);
67       }
68       else { // Существующий объект ViewHolder используется заново
69          viewHolder = (ViewHolder) convertView.getTag();
70       }
71
72       // Если значок погодных условий уже загружен, использовать его;
73       // в противном случае загрузить в отдельном потоке
74       if (bitmaps.containsKey(day.iconURL)) {
75          viewHolder.conditionImageView.setImageBitmap(
```

```
76              bitmaps.get(day.iconURL));
77        }
78        else {
79           // Загрузить и вывести значок погодных условий
80           new LoadImageTask(viewHolder.conditionImageView).execute(
81              day.iconURL);
82        }
83
84        // Получить данные из объекта Weather и заполнить представления
85        Context context = getContext(); // Для загрузки строковых ресурсов
86        viewHolder.dayTextView.setText(context.getString(
87           R.string.day_description, day.dayOfWeek, day.description));
88        viewHolder.lowTextView.setText(
89           context.getString(R.string.low_temp, day.minTemp));
90        viewHolder.hiTextView.setText(
91           context.getString(R.string.high_temp, day.maxTemp));
92        viewHolder.humidityTextView.setText(
93           context.getString(R.string.humidity, day.humidity));
94
95        return convertView; // Вернуть готовое представление элемента
96     }
97
```

Если `convertView` содержит `null`, то строка 52 создает новый объект `ViewHolder` для хранения ссылок на представления нового элемента `ListView`. Затем строка 53 получает объект `LayoutInflator`, который используется в строках 54–55 для заполнения макета элемента `ListView`. В первом аргументе передается заполняемый макет (`R.layout.list_item`), во втором — родительский объект `ViewGroup` макета, к которому будут присоединены его представления, и в последнем аргументе передается флаг автоматического присоединения представлений. В данном случае третий аргумент равен `false`, потому что `ListView` вызывает метод `getView` для получения представления `View` элемента списка, а затем присоединяет его к `ListView`. Строки 56–65 получают ссылки на представления во вновь заполненном макете и задают переменные экземпляров `ViewHolder`. Строка 66 связывает новый объект `ViewHolder` с элементом `ListView` для использования в будущем.

Если значение `convertView` отлично от `null`, то `ListView` повторно использует элемент `ListView`, вышедший за границы экрана. В этом случае строка 69 получает значение, ассоциированное с текущим элементом `ListView`, а им является объект `ViewHolder`, ранее присоединенный к элементу `ListView`.

После создания или получения объекта `ViewHolder` в строках 74–93 задаются данные для представлений `ListItem`. Строки 74–82 определяют, был ли значок погодных условий загружен ранее; в этом случае объект `bitmaps` будет содержать ключ `iconURL` объекта `Weather`. Если это так, то строки 75–76 получают существующий объект `Bitmap` из `bitmaps` и задают изображение `conditionImageView`. В противном случае строки 80–81 создают новый объект `LoadImageTask` (раздел 7.6.5) для загрузки изображения в отдельном потоке. Метод `execute` задачи

получает iconURL и запускает задачу на выполнение. В строках 86–93 назначается текст компонентов TextView элемента ListView. Наконец, строка 95 возвращает готовое представление View для элемента списка ListView.

РАЗРАБОТЧИКУ НА ЗАМЕТКУ 7.1

Каждый раз, когда вам потребуется объект AsyncTask, приходится создавать новый объект — каждая задача AsyncTask может быть выполнена только один раз.

7.6.5. Субкласс AsyncTask для загрузки изображений в отдельном потоке

Вложенный класс LoadImageTask (листинг 7.9) расширяет класс AsyncTask. Он определяет, как должна происходить загрузка значка погодных условий в отдельном потоке, а затем возвращает изображение потоку GUI для отображения в компоненте ImageView элемента ListView.

Листинг 7.9. Субкласс AsyncTask для загрузки изображения в отдельном потоке

```
98    // AsyncTask для загрузки изображения в отдельном потоке
99    private class LoadImageTask extends AsyncTask<String, Void, Bitmap>  {
100       private ImageView imageView; // Для вывода миниатюры
101
102       // Сохранение ImageView для загруженного объекта Bitmap
103       public LoadImageTask(ImageView imageView) {
104          this.imageView = imageView;
105       }
106
107       // загрузить изображение; params[0] содержит URL-адрес изображения
108       @Override
109       protected Bitmap doInBackground(String... params)  {
110          Bitmap bitmap = null;
111          HttpURLConnection connection = null;
112
113          try {
114             URL url = new URL(params[0]); // Создать URL для изображения
115
116             // Открыть объект HttpURLConnection, получить InputStream
117             // и загрузить изображение
118             connection = (HttpURLConnection) url.openConnection();
119
120             try (InputStream inputStream = connection.getInputStream()) {
121                  bitmap = BitmapFactory.decodeStream(inputStream);
122                  bitmaps.put(params[0], bitmap); // Кэширование
123             }
124             catch (Exception e) {
125                e.printStackTrace();
126             }
127          }
```

```
128          catch (Exception e) {
129             e.printStackTrace();
130          }
131          finally {
132             connection.disconnect(); // Закрыть HttpURLConnection
133          }
134
135          return bitmap;
136       }
137
138       // Связать значок погодных условий с элементом списка
139       @Override
140       protected void onPostExecute(Bitmap bitmap)  {
141          imageView.setImageBitmap(bitmap);
142       }
143    }
144 }
```

`AsyncTask` представляет собой обобщенный тип с тремя параметрами-типами.

- Первый — тип списка параметров переменной длины (`String`) для метода `doInBackground` класса `AsyncTask`, который необходимо переопределить (строки 108–136). Когда вы вызываете метод `execute`, он создает поток, в котором метод `doInBackground` выполняет свою операцию. Приложение передает строку URL значка погодных условий в аргументе метода `execute` (листинг 7.8, строки 80–81).
- Второй — тип списка параметров переменной длины для метода `onProgressUpdate` класса `AsyncTask`. Этот метод выполняется в потоке GUI и используется для получения промежуточных обновлений заданного типа от выполняемой задачи. Переопределение этого метода не является обязательным. В нашем примере метод не используется, поэтому мы передаем `Void` и игнорируем этот параметр-тип.
- Третий — тип результата задачи (`Bitmap`), передаваемого методу `onPostExecute` класса `AsyncTask` (139–143). Этот метод выполняется в потоке GUI и позволяет вывести результаты `AsyncTask` в компоненте `ImageView` элемента `ListView`. Компонент `ImageView` передается в аргументе конструктора класса `LoadImageTask` (строки 103–105) и сохраняется в переменной экземпляра в строке 100.

Важнейшее преимущество класса `AsyncTask` заключается в том, что он берет на себя технические подробности создания потоков и выполнения своих методов в соответствующих потоках, так что разработчику не нужно взаимодействовать с потоковым механизмом напрямую.

Загрузка значка погодных условий

Метод `doInBackground` использует объект `HttpURLConnection` для загрузки значка погодных условий. Строка 114 преобразует строку с URL, переданную методу `execute` класса `AsyncTask` (`params[0]`), в объект `URL`. Затем строка 118 вызывает

метод openConnection класса URL для получения объекта HttpURLConnection — преобразование типа необходимо, потому что метод возвращает URLConnection. Метод openConnection запрашивает контент, определяемый объектом URL. Строка 120 получает объект InputStream класса HttpURLConnection, который передается методу decodeStream класса BitmapFactory для чтения байтов изображения и возвращается объекту Bitmap с изображением (строка 121). В строке 122 загруженное изображение кэшируется в bitmaps для возможного повторного использования, а строка 132 вызывает унаследованный метод disconnect для закрытия подключения и освобождения ресурсов. Строка 135 возвращает загруженный объект Bitmap, который затем передается onPostExecute — в потоке GUI — для вывода изображения.

7.7. Класс MainActivity

Класс MainActivity определяет интерфейс приложения, логику взаимодействия с веб-сервисом *OpenWeatherMap.org* и логику обработки ответа в формате JSON, полученного от веб-сервиса. Вложенный субкласс AsyncTask с именем GetWeatherTask выполняет запрос к веб-сервису в отдельном потоке (раздел 7.7.5). В этом приложении активности MainActivity меню не требуется, поэтому мы удалили методы onCreateOptionsMenu и onOptionsItemSelected из автоматически сгенерированного кода.

7.7.1. Команды package и import класса MainActivity

Листинг 7.10. Команды package и import класса MainActivity

```
1 // MainActivity.java
2 // Вывод 16-дневного прогноза погоды для заданного города
3 package com.deitel.weatherviewer;
4
5 import android.content.Context;
6 import android.os.AsyncTask;
7 import android.os.Bundle;
8 import android.support.design.widget.FloatingActionButton;
9 import android.support.design.widget.Snackbar;
10 import android.support.v7.app.AppCompatActivity;
11 import android.support.v7.widget.Toolbar;
12 import android.view.View;
13 import android.view.inputmethod.InputMethodManager;
14 import android.widget.EditText;
15 import android.widget.ListView;
16
17 import org.json.JSONArray;
18 import org.json.JSONException;
19 import org.json.JSONObject;
20
```

```
21 import java.io.BufferedReader;
22 import java.io.IOException;
23 import java.io.InputStreamReader;
24 import java.net.HttpURLConnection;
25 import java.net.URL;
26 import java.net.URLEncoder;
27 import java.util.ArrayList;
28 import java.util.List;
29
```

7.7.2. Переменные экземпляров

Класс `MainActivity` (листинг 7.11) расширяет класс `AppCompatActivity` и определяет три переменные экземпляров.

- `weatherList` (строка 32) — коллекция `ArrayList<Weather>`, содержащая объекты `Weather`. Каждый объект представляет один день в прогнозе погоды.
- `weatherArrayAdapter` — ссылка на объект `WeatherArrayAdapter` (раздел 7.6), связывающий `weatherList` с элементами `ListView`.
- `weatherListView` — ссылка на компонент `ListView` активности `MainActivity`.

Листинг 7.11. Переменные экземпляров класса MainActivity

```
30 public class MainActivity extends AppCompatActivity {
31    // Список объектов Weather, представляющих прогноз погоды
32    private List<Weather> weatherList = new ArrayList<>();
33
34    // ArrayAdapter связывает объекты Weather с элементами ListView
35    private WeatherArrayAdapter weatherArrayAdapter;
36    private ListView weatherListView; // Для вывода информации
37
```

7.7.3. Переопределенный метод onCreate

Переопределенный метод `onCreate` (листинг 7.12) настраивает графический интерфейс `MainActivity`. Строки 41–45 генерируются Android Studio при выборе шаблона `Blank Activity` во время создания проекта. Они заполняют графический интерфейс, создают объект `Toolbar` и связывают его с активностью. Как упоминалось ранее, класс `AppCompatActivity` должен предоставлять собственный объект `Toolbar`, потому что панели приложений (ранее называвшиеся панелями действий) не поддерживались в ранних версиях Android.

В строках 48–50 настраивается объект `ListAdapter`, связанный с `weatherListView`, — в данном случае объект `WeatherArrayAdapter` (субкласс `ArrayAdapter`). Метод `setAdapter` класса `ListView` связывает `WeatherArrayAdapter` с `ListView` для заполнения элементов `ListView`.

Листинг 7.12. Переопределенный метод onCreate

```
38    // Настройка Toolbar, ListView и FAB
39    @Override
40    protected void onCreate(Bundle savedInstanceState) {
41       super.onCreate(savedInstanceState);
42       // Сгенерированный код для заполнения макета и настройки Toolbar
43       setContentView(R.layout.activity_main);
44       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
45       setSupportActionBar(toolbar);
46
47       // ArrayAdapter для связывания weatherList с weatherListView
48       weatherListView = (ListView) findViewById(R.id.weatherListView);
49       weatherArrayAdapter = new WeatherArrayAdapter(this, weatherList);
50       weatherListView.setAdapter(weatherArrayAdapter);
51
52       // FAB скрывает клавиатуру и выдает запрос к веб-сервису
53       FloatingActionButton fab =
54          (FloatingActionButton) findViewById(R.id.fab);
55       fab.setOnClickListener(new View.OnClickListener()  {
56          @Override
57          public void onClick(View view) {
58             // Получить текст из locationEditText и создать URL веб-сервисы
59             EditText locationEditText =
60                (EditText) findViewById(R.id.locationEditText);
61             URL url = createURL(locationEditText.getText().toString());
62
63             // Скрыть клавиатуру и запустить GetWeatherTask для получения
64             // погодных данных от OpenWeatherMap.org в отдельном потоке
65             if (url != null) {
66                dismissKeyboard(locationEditText);
67                GetWeatherTask getLocalWeatherTask = new GetWeatherTask();
68                getLocalWeatherTask.execute(url);
69             }
70             else {
71                Snackbar.make(findViewById(R.id.coordinatorLayout),
72                   R.string.invalid_url, Snackbar.LENGTH_LONG).show();
73             }
74          }
75       });
76    }
77
```

В строках 53–75 настраивается кнопка `FloatingActionButton` из шаблона `Blank Activity`. Метод слушателя `onClick` был сгенерирован Android Studio, но мы задаем собственную реализацию в этом приложении. Эта реализация сначала получает ссылку на компонент `EditText` приложения, а затем использует ее в строке 61 для получения текста, введенного пользователем. Текст передается методу `createURL` (раздел 7.7.4) для создания URL-адреса запроса к веб-сервису, возвращающего прогноз погоды.

Если создание URL прошло успешно, строка 66 скрывает клавиатуру вызовом метода `dismissKeyboard` (раздел 7.7.4). Затем строка 67 создает объект

GetWeatherTask для получения прогноза погоды в отдельном потоке, а строка 68 выполняет задачу с передачей URL запроса к веб-сервису в аргументе метода execute класса AsyncTask. Если в процессе создания URL произошла ошибка, строки 71–72 создают объект Snackbar с сообщением о недействительном URL-адресе.

7.7.4. Методы dismissKeyboard и createURL

В листинге 7.13 содержатся методы dismissKeyboard и createURL класса MainActivity. Вызов метода dismissKeyboard (строки 79–83) скрывает экранную клавиатуру, когда пользователь касается кнопки FloatingActionButtion для передачи города приложению. Android предоставляет средства управления клавиатурой на программном уровне. Чтобы получить ссылку на эту системную службу (и многие другие службы Android), вызовите метод getSystemService класса Context с соответствующей константой — Context.INPUT_METHOD_SERVICE в данном случае. Этот метод может возвращать объекты многих разных типов, поэтому возвращаемое значение необходимо преобразовать к нужному типу — InputMethodManager (пакет android.view.inputmethod). Клавиатура скрывается вызовом метода hideSoftInputFromWindow класса InputMethodManager (строка 82).

Листинг 7.13. Методы dismissKeyboard и createURL

```
78    // Клавиатура закрывается при касании кнопки FAB
79    private void dismissKeyboard(View view) {
80       InputMethodManager imm = (InputMethodManager) getSystemService(
81          Context.INPUT_METHOD_SERVICE);
82       imm.hideSoftInputFromWindow(view.getWindowToken(), 0);
83    }
84
85    // Создание URL веб-сервисы openweathermap.org для названия города
86    private URL createURL(String city) {
87       String apiKey = getString(R.string.api_key);
88       String baseUrl = getString(R.string.web_service_url);
89
90       try {
91          // Создание URL для заданного города и температурной шкалы (Фаренгейт)
92          String urlString = baseUrl + URLEncoder.encode(city, "UTF-8") +
93             "&units=imperial&cnt=16&APPID=" + apiKey;
94          return new URL(urlString);
95       }
96       catch (Exception e) {
97          e.printStackTrace();
98       }
99
100       return null; // Некорректный URL
101    }
102
```

Метод `createURL` (строки 86–101) формирует строковое представление URL запроса к веб-сервису (строки 92–93). Затем строка 94 пытается создать и вернуть объект `URL`, инициализированный строкой URL-адреса. Строка 93 добавляет параметры к запросу

&units=imperial&cnt=16&APPID=

Параметр `units` может принимать значения `imperial` (для шкалы Фаренгейта), `metric` (для шкалы Цельсия) или `standard` (для шкалы Кельвина) — если параметр `units` не указан, по умолчанию используется значение `standard`. Параметр `cnt` определяет количество дней в прогнозе. Максимальное значение равно 16, значение по умолчанию равно 7 (при некорректном количестве дней возвращается прогноз на 7 дней). Наконец, параметр `APPID` предназначен для ключа API *OpenWeatherMap.org*, который загружается в приложение из строкового ресурса `api_key`. По умолчанию прогноз возвращается в формате JSON, хотя вы можете добавить параметр `mode` со значением `XML` или `HTML`, чтобы получить данные в формате XML или веб-страницы соответственно.

7.7.5. Субкласс AsyncTask для обращения к веб-сервису

Вложенный класс `GetWeatherTask` (листинг 7.14) выполняет запрос к веб-сервису и обрабатывает ответ в отдельном потоке, после чего передает информацию прогноза в виде `JSONObject` потоку GUI для отображения.

Листинг 7.14. Субкласс AsyncTask для обращения к веб-сервису

```
103    // Обращение к REST-совместимому веб-сервису за погодными данными
104    // и сохранение данных в локальном файле HTML
105    private class GetWeatherTask
106         extends AsyncTask<URL, Void, JSONObject> {
107
108       @Override
109          protected JSONObject doInBackground(URL... params) {
110          HttpURLConnection connection = null;
111
112          try {
113             connection = (HttpURLConnection) params[0].openConnection();
114             int response = connection.getResponseCode();
115
116             if (response == HttpURLConnection.HTTP_OK)  {
117                StringBuilder builder = new StringBuilder();
118
119                try (BufferedReader reader = new BufferedReader(
120                   new InputStreamReader(connection.getInputStream()))) {
121
122                   String line;
123
124                   while ((line = reader.readLine()) != null) {
125                      builder.append(line);
```

```
126                  }
127               }
128               catch (IOException e) {
129                  Snackbar.make(findViewById(R.id.coordinatorLayout),
130                     R.string.read_error, Snackbar.LENGTH_LONG).show();
131                  e.printStackTrace();
132               }
133
134               return new JSONObject(builder.toString());
135            }
136            else {
137               Snackbar.make(findViewById(R.id.coordinatorLayout),
138                  R.string.connect_error, Snackbar.LENGTH_LONG).show();
139            }
140         }
141         catch (Exception e) {
142            Snackbar.make(findViewById(R.id.coordinatorLayout),
143               R.string.connect_error, Snackbar.LENGTH_LONG).show();
144            e.printStackTrace();
145         }
146         finally {
147            connection.disconnect(); // Закрыть HttpURLConnection
148         }
149
150         return null;
151      }
152
153      // Обработка ответа JSON и обновление ListView
154      @Override
155         protected void onPostExecute(JSONObject weather) {
156         convertJSONtoArrayList(weather); // Заполнение weatherList
157         weatherArrayAdapter.notifyDataSetChanged(); // Связать с ListView
158         weatherListView.smoothScrollToPosition(0); // Прокрутить до верха
159      }
160   }
161
```

Класс `GetWeatherTask` получает три параметра-типа.

- `URL` — тип списка параметров переменной длины метода `doInBackground` класса `AsyncTask` (строки 108–151) — URL запроса к веб-сервису передается в единственном аргументе метода `execute` класса `GetWeatherTask`.
- `Void` — тип списка параметров переменной длины метода `onProgressUpdate` — как уже было сказано, в приложении этот метод не используется.
- `JSONObject` — тип результата задачи — передается методу `onPostExecute` (строки 154–159) в потоке GUI для отображения результатов.

Строка 113 в `doInBackground` создает объект `HttpURLConnection`, который используется для запроса к REST-совместимому веб-сервису. Как и в разделе 7.6.5, для выдачи запроса достаточно открыть объект подключения. Строка 114 получает код ответа от веб-сервера. Если код ответа равен `HttpURLConnection.HTTP_OK`, значит,

REST-совместимый веб-сервис был вызван правильно и вернул данные, которые необходимо обработать. В этом случае строки 119–126 получают поток `InputStream` класса `HttpURLConnection`, «упаковывают» его в `BufferedReader`, читают каждую строку текста из ответа и присоединяют ее к `StringBuilder`. Затем строка 134 преобразует строку JSON, содержащуюся в `StringBuilder`, в `JSONObject` и возвращает ее потоку GUI. Строка 147 закрывает подключение `HttpURLConnection`.

Если при чтении погодных данных или подключении к веб-сервису будет обнаружена ошибка, в строках 129–130, 137–138 или 142–143 отображается уведомление `Snackbar` с описанием проблемы. Проблемы могут возникнуть из-за того, что устройство потеряло доступ к сети на середине передачи запроса или сеть с самого начала была недоступна — например, если устройство было переведено в авиарежим.

При вызове `onPostExecute` в потоке GUI строка 156 вызывает метод `convertJSONtoArrayList` (раздел 7.7.6) для извлечения погодных данных из `JSONObject` и их включения в `weatherList`. Затем строка 157 вызывает метод `notifyDataSetChanged` класса `ArrayAdapter`, в результате чего `weatherListView` обновляется новыми данными.

Строка 158 вызывает метод `smoothScrollToPosition` класса `ListView` для перевода `ListView` к первому элементу в самом начале списка — это делается для того, чтобы прогноз отображался с первого дня.

7.7.6. Метод convertJSONtoArrayList

В разделе 7.3.2 рассматривался формат JSON, возвращаемый погодным веб-сервисом *OpenWeatherMap.org*. Метод `convertJSONtoArrayList` (листинг 7.15) извлекает эти данные из аргумента `JSONObject`. Сначала строка 164 удаляет из `weatherList` все существующие объекты `Weather`. При обработке данных JSON в `JSONObject` или `JSONArray` могут возникнуть исключения `JSONException`, поэтому строки 168–188 включены в блок `try`.

Листинг 7.15. Метод convertJSONToArrayList

```
162    // Создание объектов Weather на базе JSONObject с прогнозом
163    private void convertJSONtoArrayList(JSONObject forecast) {
164       weatherList.clear(); // Стирание старых погодных данных
165
166       try {
167          // Получение свойства "list" JSONArray
168          JSONArray list = forecast.getJSONArray("list");
169
170          // Преобразовать каждый элемент списка в объект Weather
171          for (int i = 0; i < list.length(); ++i) {
172             JSONObject day = list.getJSONObject(i); // Данные за день
173
```

```
174             // Получить JSONObject с температурами дня ("temp")
175             JSONObject temperatures = day.getJSONObject("temp");
176
177             // Получить JSONObject с описанием и значком ("weather")
178             JSONObject weather =
179                day.getJSONArray("weather").getJSONObject(0);
180
181             // Добавить новый объект Weather в weatherList
182             weatherList.add(new Weather(
183                day.getLong("dt"), // Временная метка даты/времени
184                temperatures.getDouble("min"), // Мин. температура
185                temperatures.getDouble("max"), // Макс. температура
186                day.getDouble("humidity"), // Процент влажности
187                weather.getString("description"), // Погодные условия
188                weather.getString("icon"))); // Имя значка
189          }
190       }
191       catch (JSONException e) {
192          e.printStackTrace();
193       }
194    }
195 }
```

Строка 168 получает свойство `"list"` `JSONArray`, вызывая метод `getJSONArray` класса `JSONObject` с передачей в аргументе имени свойства массива. Затем строки 171–189 создают объект `Weather` для каждого элемента `JSONArray`. Метод `length` класса `JSONArray` возвращает количество элементов в массиве (строка 171).

Затем строка 172 получает объект `JSONObject`, который представляет прогноз за один день из `JSONArray`; для этого вызывается метод `getJSONObject`, получающий индекс в аргументе. Строка 175 получает объект JSON `"temp"` с температурными данными за день. Строки 178–179 получают массив JSON `"weather"`, после чего получают первый элемент массива, который содержит описание погоды и значок.

В строках 182–188 создается объект `Weather`, который добавляется в `weatherList`. Строка 183 использует метод `getLong` класса `JSONObject` для получения временной метки (`"dt"`), которую конструктор `Weather` преобразует в название дня недели. Строки 184–186 вызывают метод `getDouble` класса `JSONObject` для получения минимальной (`"min"`) и максимальной (`"max"`) температур из объекта `temperatures`, а также процента влажности `"humidity"` из объекта `day`. Наконец, в строках 187–188 метод `getString` используется для получения строкового описания погоды и значка погодных условий из объекта `weather`.

7.8. Резюме

В этой главе мы построили приложение `WeatherViewer`. Приложение получает 16-дневный прогноз погоды от веб-сервиса, предоставляемого сайтом

OpenWeatherMap.org, и отображает прогноз в списке `ListView`. Вы познакомились с архитектурным стилем реализации веб-сервисов REST (REpresentational State Transfer) и узнали, что приложения используют веб-стандарты (такие как протокол HTTP) для обращения к REST-совместимым веб-сервисам и получения от них ответов.

Веб-сервис *OpenWeatherMap.org*, используемый в этом приложении, возвращает прогноз в формате JSON (JavaScript Object Notation). Вы узнали, что JSON представляет собой текстовый формат, в котором объекты представлены в виде коллекций пар «имя/значение». Для обработки данных JSON использовались классы `JSONObject` и `JSONArray` из пакета `org.json`.

Для обращения к веб-сервису строка URL была преобразована в объект `URL`. Затем объект `URL` использовался для открытия объекта `HttpUrlConnection`, в результате чего происходило обращение к веб-сервису через запрос HTTP. Приложение читало все данные из потока `InputStream` класса `HttpUrlConnection` и помещало их в строку, а затем преобразовывало эту строку в `JSONObject` для обработки. Вы увидели, как организуется выполнение продолжительных операций за пределами потока GUI и получение результатов в потоке GUI с использованием объектов `AsyncTask`. Это особенно важно для обращений к веб-сервисам, время выполнения которых не детерминировано. Погодные данные выводятся в компоненте `ListView`, при этом данные для каждого элемента `ListView` поставляются при помощи субкласса `ArrayAdapter`. Вы узнали, как повысить быстродействие `ListView` за счет использования паттерна View-Holder, при котором представления существующих элементов `ListView` используются заново при выходе элементов за границы экрана.

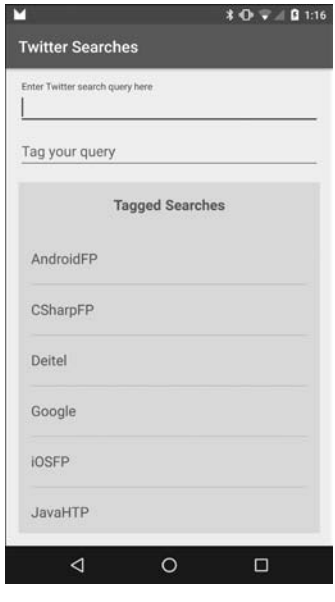
Наконец, мы использовали некоторые средства материального дизайна из библиотеки Android Design Support Library — компонент `TextInputLayout` оставляет подсказку `EditText` на экране даже после того, как пользователь начал вводить текст; плавающая кнопка `FloatingActionButton` используется для отправки данных, введенных пользователем; компонент `Snackbar` отображает информационное сообщение для пользователя.

В следующей главе будет построено приложение `Twitter® Searches`. Многие мобильные приложения отображают списки, как и в приложении этой главы. В главе 8 для решения этой задачи будет использоваться компонент `RecyclerView`, получающий данные от `ArrayList<String>`. Для больших наборов данных он работает более эффективно, чем `ListView`. Также будут рассмотрены средства сохранения данных приложения в настройках пользователя и запуска браузера для отображения веб-страницы.

8 Приложение Twitter® Searches

Настройки SharedPreferences, SharedPreferences.Editor, неявные интенты, RecyclerView, RecyclerView.Adapter, RecyclerView.ViewHolder, RecyclerView.ItemDecoration

В этой главе…

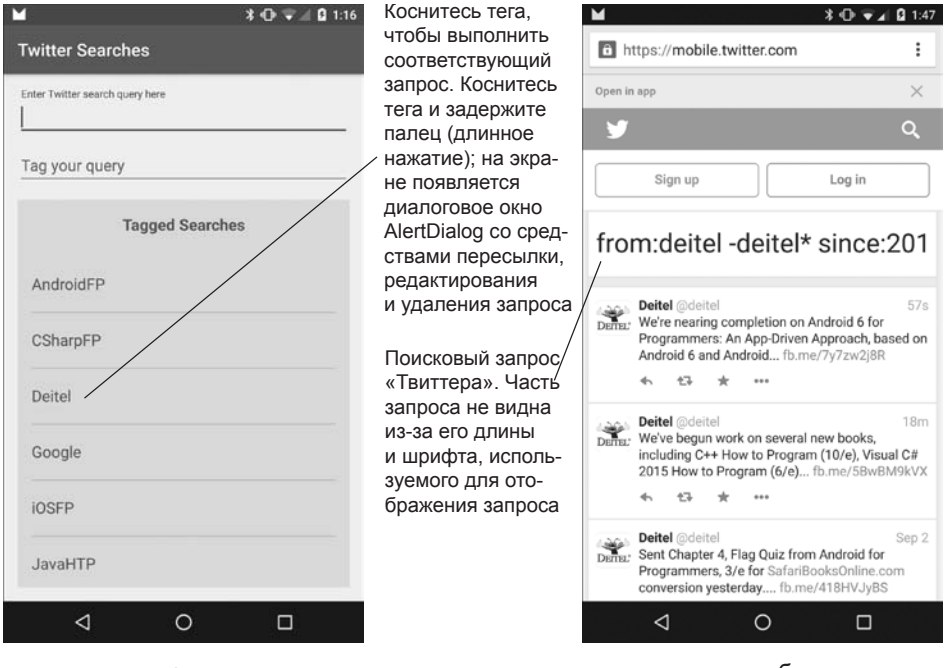


- Использование SharedPreferences для хранения данных, связанных с приложением, в виде пар «ключ—значение»
- Использование неявного интента для открытия сайта в браузере
- Использование неявного интента для вывода списка приложений, способных обмениваться текстом
- Вывод списка с поддержкой прокрутки и компонент RecyclerView
- Использование субкласса RecyclerView.Adapter для определения данных RecyclerView
- Использование субкласса RecyclerView.ViewHolder для реализации паттерна View-Holder
- Использование субкласса RecyclerView.ItemDecoration для вывода линий между строками RecyclerView
- Создание диалогового окна AlertDialog с помощью объекта AlertDialog.Builder

8.1. Введение

Поисковый механизм «Твиттера» упрощает отслеживание актуальных тем, обсуждаемых более 300 миллионами активных пользователей[1] (общее количество учетных записей превышает 1 миллиард[2]). Поисковые запросы «Твиттера» могут настраиваться с помощью *поисковых операторов* «Твиттера» (см. раздел 8.2), при этом следует иметь в виду, что более сложные запросы имеют большую длину, и вводить их на мобильных устройствах трудно и неудобно. Приложение `Twitter® Searches` (рис. 8.1) позволяет сохранить избранные поисковые запросы пользователя на устройстве с короткими, легко запоминающимися именами (тегами) (рис. 8.1, *а*). Касаясь кнопки поиска, вы можете быстро и легко отслеживать сообщения по заданной теме (рис. 8.1, *б*). Как вы вскоре увидите, приложение также позволяет пересылать, редактировать и удалять сохраненные запросы.

Приложение поддерживает как портретную, так и альбомную ориентацию устройства. В приложении `Flag Quiz` мы предоставляли отдельный макет для



а *б*

Рис. 8.1. Приложение Twitter Searches: *а* — приложение с несколькими сохраненными поисковыми запросами; *б* — приложение после касания кнопки Deitel

[1] *https://about.twitter.com/company*.

[2] *http://www.businessinsider.com/twitter-monthly-active-users-2015-7?r=UK&IR=T*.

каждой ориентации, а в приложении `Doodlz` ориентация выбиралась на программном уровне. В этом приложении мы обеспечим поддержку обеих ориентаций, спроектировав графический интерфейс с возможностью динамического изменения размеров компонентов GUI для текущей ориентации.

Начнем с тестового запуска приложения, а затем приведем краткий обзор технологий, использованных для его построения. Далее будет спроектирован графический интерфейс приложения. Глава завершается описанием полного исходного кода приложения, при этом основное внимание будет уделяться его новым возможностям.

8.2. Тестирование приложения

Запустите среду разработки Android Studio и откройте приложение `Twitter Searches` из папки `TwitterSearches` в примерах книги. Запустите приложение в AVD или на устройстве. Среда разработки строит проект и запускает приложение (рис. 8.2).

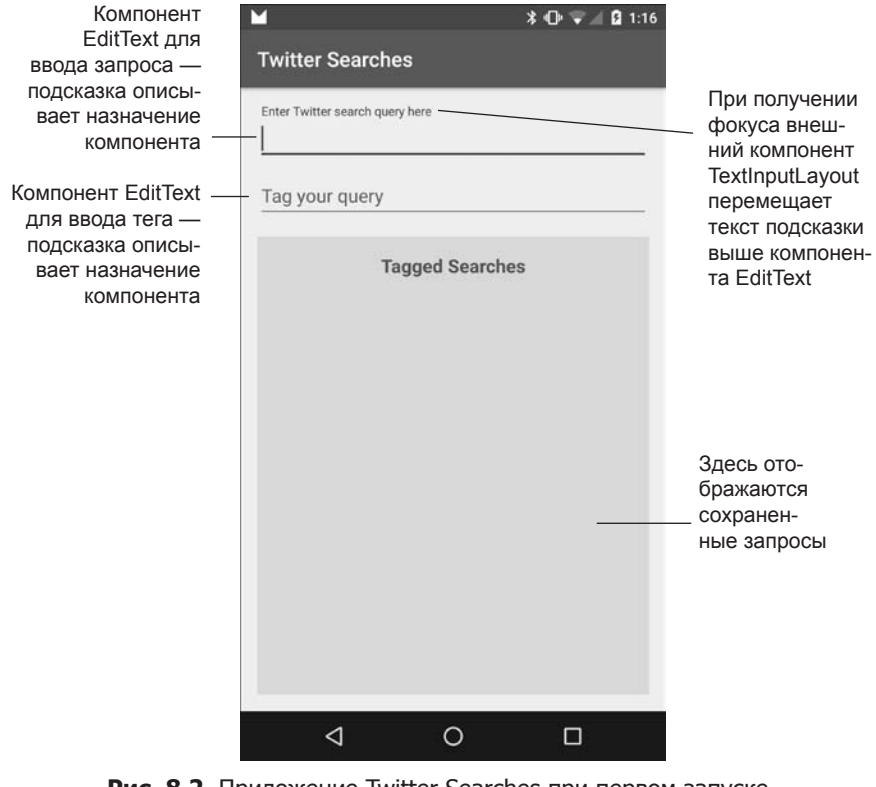


Рис. 8.2. Приложение Twitter Searches при первом запуске

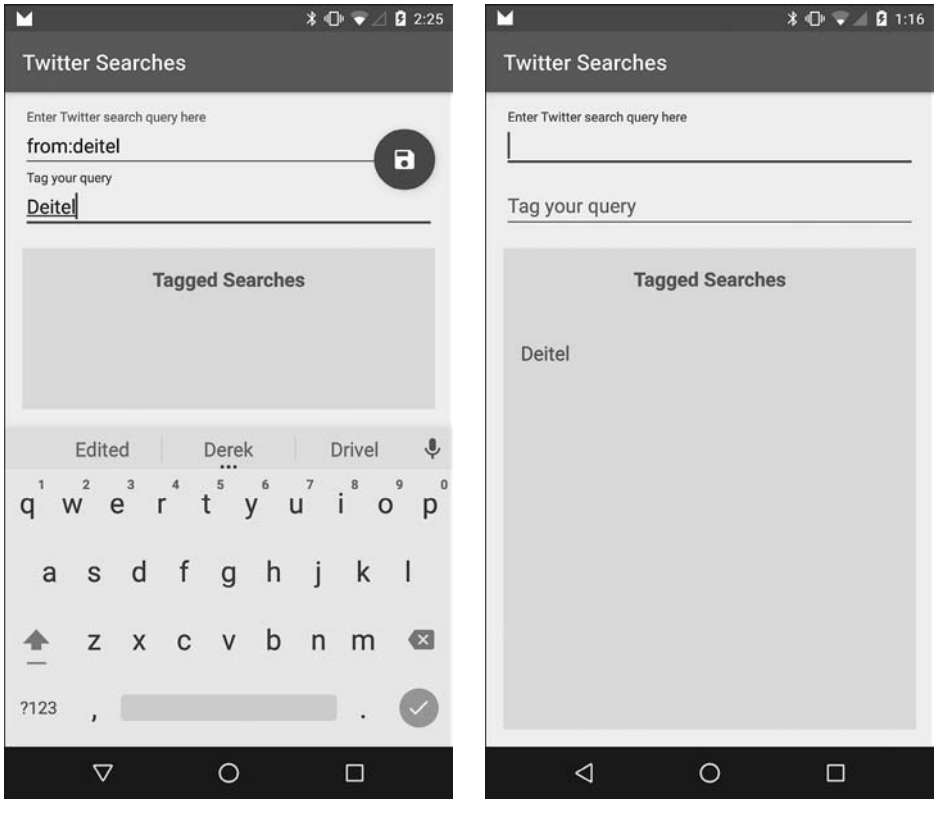
8.2.1. Добавление нового запроса

Коснитесь верхнего компонента `EditText` и введите поисковый запрос *from:deitel* — оператор *from:* включает поиск сообщений от заданного пользователя «Твиттера». В табл. 8.1 представлены примеры использования поисковых операторов «Твиттера»; в сложных запросах можно задействовать сразу несколько операторов. Полный список доступен по адресу *http://bit.ly/TwitterSearchOperators*.

Таблица 8.1. Некоторые поисковые операторы «Твиттера»

Пример	Критерий поиска сообщений
google android	Неявный оператор «логическое И» — находит сообщения, содержащие текст google *и* текст android
google OR android	Логический оператор «ИЛИ» — находит сообщения, содержащие текст google *или* текст android (или оба текста сразу)
"how to program"	Строка в кавычках — находит сообщения, содержащие текст в заданном виде ("how to program")
android ?	Вопросительный знак — находит сообщения с текстом android, в которых задается вопрос
google -android	Знак «минус» — находит сообщения, содержащие текст google, но не содержащие android
android :)	:) — находит *позитивные* сообщения, содержащие текст android
android :(	:(— находит *негативные* сообщения, содержащие текст android
since:2015-10-01	Находит сообщения, опубликованные в указанный день или после него (дата задается в формате ГГГГ-ММ-ДД)
near:"New York City"	Находит сообщения, отправленные рядом с заданным местом
from:GoogleCode	Находит сообщения от пользователя «Твиттера» @GoogleCode
to:GoogleCode	Находит сообщения, адресованные пользователю «Твиттера» @GoogleCode

В нижнем компоненте `EditText` введите тег запроса `Deitel` (рис. 8.3, *а*). Это короткое имя будет отображаться в списке сохраненных запросов. Коснитесь кнопки , чтобы сохранить запрос — тег «Deitel» появляется в списке под заголовком `Tagged Searches` (рис. 8.3, *б*). При сохранении запроса виртуальная клавиатура исчезает, чтобы список был виден на экране — о том, как скрыть виртуальную клавиатуру, рассказано в разделе 8.5.5.



а *б*

Рис. 8.3. Ввод поискового запроса: *а* — ввод запроса и тега (короткого имени); *б* — приложение после сохранения запроса и тега

8.2.2. Просмотр результатов поиска

Чтобы просмотреть результаты поиска, коснитесь тега `Deitel`. На устройстве запускается браузер, которому передается URL-адрес, представляющий сохраненный поисковый запрос. «Твиттер» получает поисковый запрос из URL и возвращает сообщения, соответствующие запросу (если они обнаружены), в виде веб-страницы. Затем браузер выводит страницу результатов (рис. 8.4). Просмотрев результаты, коснитесь кнопки `Back` (◀), чтобы вернуться к приложению `Twitter Searches`, в котором вы можете сохранить новые запросы, а также отредактировать, удалить и опубликовать ранее сохраненные запросы. Для запроса `"from:deitel"` «Твиттер» выдает учетные записи пользователей, в имя которых входит строка `deitel`, а также последние сообщения от этих пользователей.

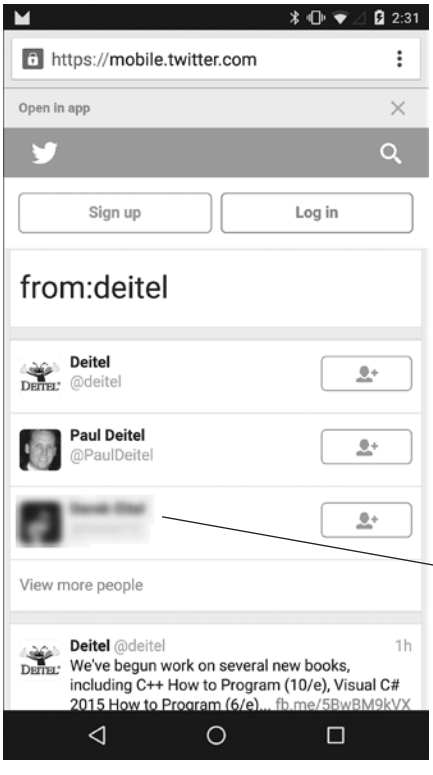
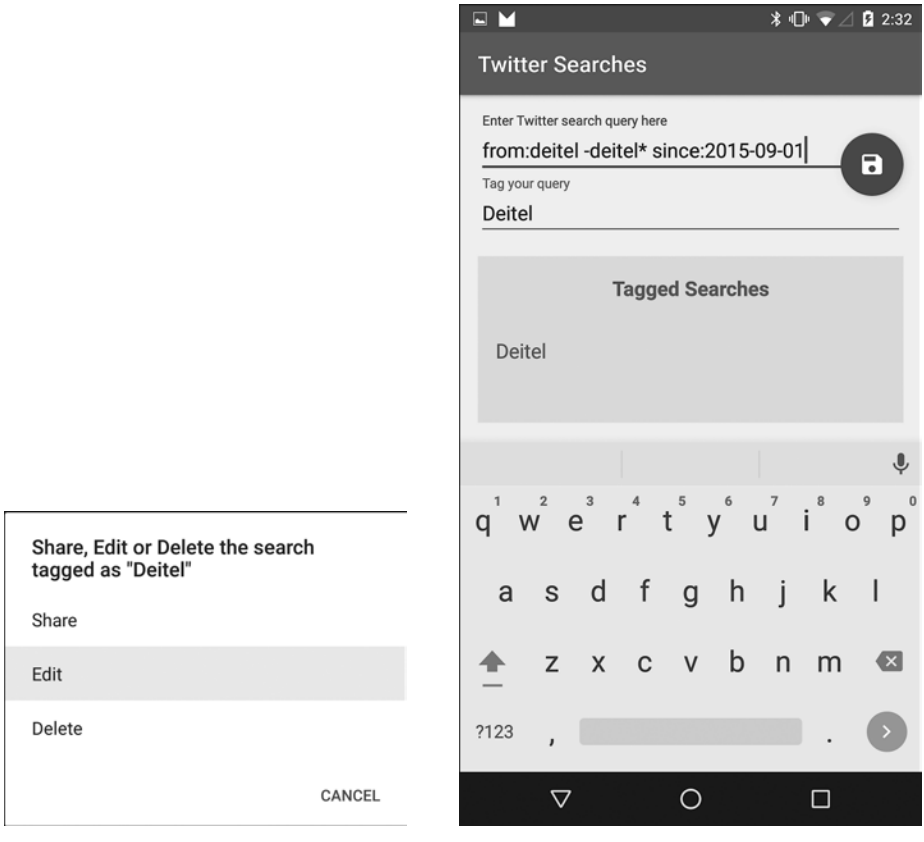


Рис. 8.4. Просмотр результатов поиска по запросу from:deitel

8.2.3. Редактирование запроса

Запросы также можно *пересылать*, *редактировать* и *удалять*. Чтобы увидеть эти команды, выполните длинное нажатие на теге запроса (коснитесь тега и задержите палец у экрана). Если вы используете AVD, щелкните и удерживайте левую кнопку мыши на теге, чтобы имитировать длинное нажатие. При длинном нажатии на теге `Deitel` открывается диалоговое окно `AlertDialog` (рис. 8.5, *а*) с командами `Share`, `Edit` и `Delete` для поискового запроса с тегом `Deitel`. Если вы не хотите выполнить ни одну из этих операций, коснитесь кнопки `CANCEL`.

Чтобы изменить запрос с тегом `Deitel`, коснитесь команды `Edit` в диалоговом окне. Приложение загружает запрос и тег в компоненты `EditText` для редактирования. Чтобы ограничить поиск сообщениями, отправленными после 1 сентября 2015 года, добавьте пробел и фрагмент `since:2015-09-01` в конец запроса (рис. 8.5, *б*) в верхнем компоненте `EditText`. Оператор `since:` ограничивает результаты поиска сообщениями, отправленными в заданный день или после него (в формате *ГГГГ-ММ-ДД*). Коснитесь кнопки , чтобы обновить сохраненный



а *б*

Рис. 8.5. Редактирование сохраненного запроса: *а* — выбор команды Edit для редактирования существующего запроса; *б* — редактирование сохраненного запроса Deitel

запрос, и просмотрите обновленные результаты (рис. 8.6), коснувшись тега `Deitel` в разделе `Tagged Searches` приложения. (Примечание: изменение имени тега приводит к созданию новой кнопки поиска — эта возможность удобна для создания новых запросов на базе ранее сохраненного запроса.)

8.2.4. Пересылка запроса

Android упрощает распространение разных видов информации из приложений по электронной почте, через системы мгновенной передачи сообщений (SMS), Facebook, Google+ и т. д. Наше приложение позволяет переслать информацию

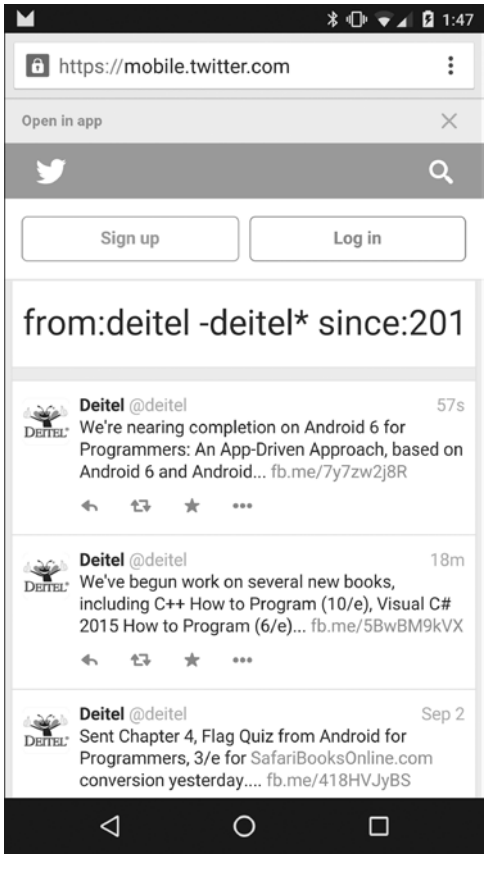
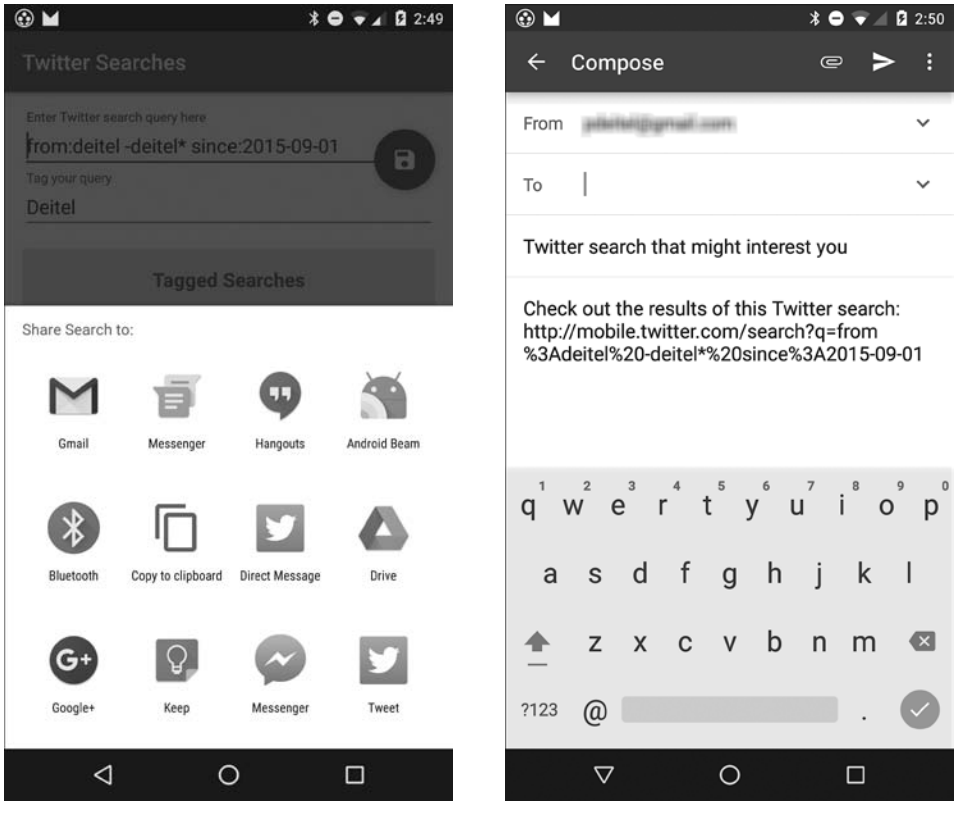


Рис. 8.6. Просмотр обновленных результатов поиска «Deitel»

о поисковом запросе — сделайте длинное нажатие на теге запроса и выберите команду `Share` в открывшемся окне `AlertDialog`. На экране появляется окно выбора (рис. 8.7, *а*), содержимое которого зависит от типа передаваемой информации и приложений, способных эту информацию обрабатывать. В нашем приложении публикуется текстовая информация, а окно выбора на нашем телефоне содержит список приложений, способных обрабатывать текст. Если информация не может быть обработана ни одним приложением, в окне выбора выводится соответствующее сообщение. Если информация обрабатывается только одним приложением, это приложение запускается без отображения промежуточного окна выбора. В тестовом запуске было выбрано приложение `Gmail`. На рис. 8.7, *б* показан экран `Compose` приложения `Gmail` с заполненным полем темы и телом сообщения. (Мы удалили адрес электронной почты в поле `From` на экранном снимке по соображениям конфиденциальности.)



а *б*

Рис. 8.7. Пересылка запроса по электронной почте: *а* — окно выбора с вариантами пересылки информации; *б* — экран Compose приложения Gmail для поискового запроса «Deitel»

8.2.5. Удаление запроса

Чтобы удалить запрос, выполните длинное нажатие на теге запроса и выберите в открывшемся окне `AlertDialog` команду `Delete`. Приложение предлагает подтвердить удаление запроса (рис. 8.8) — кнопка `CANCEL` возвращает приложение к главному экрану без удаления, а кнопка `DELETE` удаляет запрос.

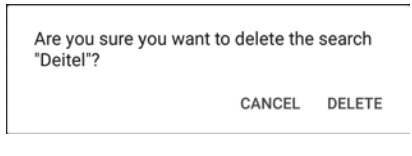


Рис. 8.8. Окно AlertDialog с предложением подтвердить удаление

8.2.6. Прокрутка списка сохраненных запросов

На рис. 8.9 изображено приложение после сохранения 10 запросов, только шесть из которых видны в настоящий момент. Приложение дает возможность прокрутить список запросов, если он не помещается на экране полностью. В отличие от настольных приложений, приложения для устройств с сенсорным экраном обычно не отображают полосы прокрутки. Чтобы прокрутить список, проведите пальцем (или мышью в AVD) по списку `Tagged Searches`. Также попробуйте повернуть устройство в альбомную ориентацию и убедитесь в том, что графический интерфейс динамически подстраивается под новые размеры.

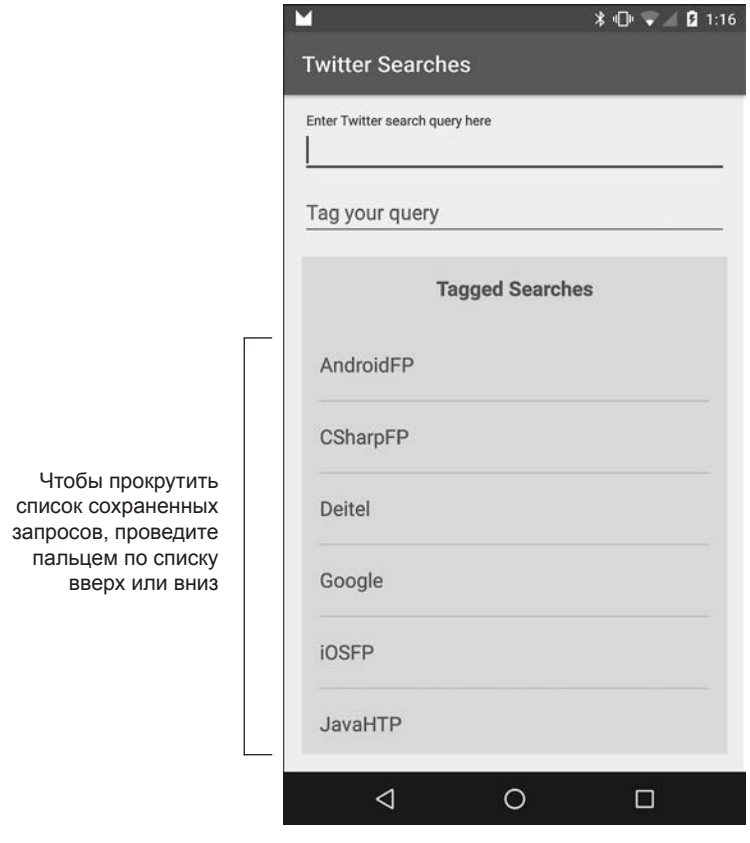


Рис. 8.9. Приложение с длинным списком запросов, не помещающимся на экране

8.3. Обзор применяемых технологий

В этом разделе представлены некоторые функции, используемые в приложении.

8.3.1. Хранение пар «ключ—значение» в файле SharedPreferences

С приложением могут быть связаны один или несколько файлов, содержащих пары «ключ—значение» (ключ обеспечивает быстрый доступ к соответствующему значению). В приложении `Flag Quiz` главы 4 настройки приложения хранились в файле `SharedPreferences` на устройстве. Фрагмент `PreferenceFragment` этого приложения создавал файл `SharedPreferences` за вас. В этом приложении используется файл с именем `searches`, в котором хранятся пары из тегов (ключи) и поисковых запросов «Твиттера» (значения), созданных пользователем. Для работы с парами «ключ—значение» в этом файле используются объекты `SharedPreferences.Editor`.

ПОВЫШЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ 8.1

В этом приложении объем хранимых данных относительно невелик, поэтому сохраненные запросы читаются с устройства в методе onCreate класса MainActivity. Продолжительные операции с данными не должны выполняться в UI-потоке; в противном случае приложение выдаст диалоговое окно ANR (Application Not Responding) — обычно это происходит через пять секунд бездействия приложения. За информацией о разработке приложений, реагирующих на действия пользователей, обращайтесь по адресу http://developer.android.com/training/articles/perf-anr.html. Также рассмотрите возможность использования объектов AsyncTask (см. главу 7).

8.3.2. Неявные интенты и выбор интентов

В главе 4 *явный интент* использовался для запуска конкретной активности в том же приложении. Android также поддерживает *неявные* (implicit) интенты, для которых разработчик *не указывает*, какой именно компонент должен обрабатывать интент. В этом приложении будут использоваться два неявных интента:

- один запускает браузер устройства по умолчанию для вывода результатов поиска в «Твиттере» на основании поискового запроса, встроенного в URL, и
- другой разрешает пользователю выбрать приложение для передачи текста, чтобы пользователь мог поделиться с другими результатами поиска.

В обоих случаях, если система не может найти активность для обработки действия, метод `startActivity` инициирует исключение `ActivityNotFoundException`. В общем

случае рекомендуется предусмотреть обработку этого исключения в программах. За дополнительной информацией об интентах обращайтесь по адресу

http://developer.android.com/guide/components/intents-filters.html

При получении неявного интента Android находит все установленные приложения, содержащие активность, способную обработать заданное действие и тип данных. Если такое приложение только одно, Android запускает соответствующую активность в приложении. Если интент может быть обработан несколькими активностями, система выводит диалоговое окно, в котором пользователь выбирает приложение. Например, если пользователь выбирает сохраненный поиск и на устройстве установлен только один браузер, Android немедленно запускает браузер для выполнения поиска и отображения результатов. Если же в системе установлены два и более браузера, пользователь должен выбрать, какой браузер должен использоваться для выполнения операции.

8.3.3. RecyclerView

В главе 7 компонент `ListView` использовался для вывода прогноза погоды — относительно небольшого набора данных. Например, в почтовом клиенте он может использоваться для вывода списка сообщений электронной почты, в адресной книге — для вывода списка контактов, в приложении для чтения новостей — для вывода списка заголовков и т. д. Во всех случаях пользователь касается элемента в списке для просмотра дополнительной информации — содержимого выбранного сообщения, подробной информации о выбранном контакте, текста выбранной новости и т. д.

RecyclerView и ListView

В этом приложении для отображения списка поисковых запросов с поддержкой прокрутки будет использоваться компонент `RecyclerView` (пакет `android.support.v7.widget`) — гибкое представление с широкими возможностями настройки и управления выводом списка данных. Компонент `RecyclerView` был разработан как улучшенная версия `ListView`. Он помогает отделить представление данных от механизма повторного использования представлений `RecyclerView` (раздел 8.3.4), а также обладает более широкими возможностями настройки (раздел 8.3.5) для представления элементов `RecyclerView`. Например, элементы `ListView` всегда отображаются в вертикальном списке, тогда как `RecyclerView` позволяет выбрать между отображением в виде вертикального списка или таблицы. Вы даже можете определить собственный менеджер макета.

Менеджеры макетов RecyclerView

В этом приложении компонент `RecyclerView` будет использовать менеджер макета `LinearLayoutManager` (субкласс `RecyclerView.LayoutManager`) для определения

того, что элементы должны отображаться в вертикальном списке, а в каждом элементе списка поисковый запрос будет отображаться как строка в текстовом представлении. Также можно определить пользовательский макет для элементов `RecyclerView`.

8.3.4. RecyclerView.Adapter и RecyclerView.ViewHolder

В главе 7 субкласс `Adapter` использовался для связывания данных с `ListView`. Также был представлен паттерн View-Holder для повторного использования представлений, выходящих за границы экрана. Вспомните, что мы создали класс `ViewHolder` (раздел 8.6.2) для хранения ссылок на представления в элементе `ListView`. Субкласс `Adapter` хранит с каждым элементом списка `ListView` объект `ViewHolder`, чтобы мы могли повторно использовать представления элемента `ListView`. Субкласс `Adapter` хранит объект `ViewHolder` с каждым элементом `ListView`, чтобы мы могли повторно использовать представления элемента `ListView`. Использовать этот паттерн не обязательно, но желательно, потому что он повышает скорость прокрутки `ListView`.

В компоненте `RecyclerView` паттерн View-Holder формализируется и становится обязательным. Разработчик создает субкласс `RecyclerView.Adapter` для связывания элементов списка `RecyclerView` с данными в списке `List` (раздел 8.6). У каждого элемента `RecyclerView` имеется соответствующий объект субкласса `RecyclerView.ViewHolder` (раздел 8.6.2), который хранит ссылки на представления элемента. Взаимодействие `RecyclerView` и `RecyclerView.Adapter` обеспечивает повторное использование представлений элементов, выходящих за границы экрана.

8.3.5. RecyclerView.ItemDecoration

Класс `ListView` автоматически выводит горизонтальные линии между элементами, но `RecyclerView` никаких декоративных элементов по умолчанию не предоставляет. Для рисования горизонтальных разделительных линий между элементами мы определяем субкласс `RecyclerView.ItemDecoration` (раздел 8.7).

8.3.6. Отображение списка команд в AlertDialog

В этом приложении пользователь может выполнить длинное нажатие на элементе `RecyclerView`, чтобы вызвать окно `AlertDialog`. Окно содержит список команд, из которых пользователь может выбрать только одну. Метод `setItems` класса `AlertDialog.Builder` задает ресурс строкового массива с именами отображаемых команд и назначает обработчик события, который вызывается при касании команды пользователем.

8.4. Создание графического интерфейса приложения и файлов ресурсов

В этом разделе мы займемся созданием графического интерфейса приложения `Twitter Searches` и файлов ресурсов. Как говорилось в разделе 8.3.3, компонент `RecyclerView` не определяет способ отображения своих элементов списка. Это означает, что разработчик также должен создать макет, определяющий графический интерфейс элемента списка. `RecyclerView` заполняет этот макет при создании элементов списка.

8.4.1. Создание проекта

Создайте новый проект на базе шаблона `Blank Activity`. Фрагменты в данном приложении не используются, поэтому при настройке шаблона не устанавливайте флажок `Use a Fragment`. На шаге `New Project` диалогового окна `Create New Project` укажите следующие значения:

- `Application name`: `Twitter Searches`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

Добавьте в проект значок приложения так, как было описано ранее. Удалите текстовое поле `Hello World!` из файла `content_main.xml`, оно в приложении не используется. Настройте поддержку Java SE 7 в соответствии с разделом 4.4.3.

8.4.2. AndroidManifest.xml

Многие пользователи запускают приложение, чтобы выполнить сохраненный ранее поисковый запрос. Если первым компонентом GUI этой активности, способным получить фокус, является `EditText`, то Android передает этому компоненту фокус при отображении активности. При получении фокуса компонентом `EditText` отображается соответствующая экранная клавиатура, если только устройство не оснащено физической клавиатурой. В нашем приложении следует запретить отображение экранной клавиатуры, если только пользователь не коснулся одного из полей `EditText` приложения. Для этого задайте значение свойства `windowSoftInputMode` так, как описано в разделе 3.7, но выберите значение `stateAlwaysHidden`.

8.4.3. Добавление библиотеки RecyclerView

В этом приложении используются новые компоненты материального оформления из библиотеки Android Design Support Library, включая `TextInputLayout`,

FloatingActionButton и RecyclerView. В новых шаблонах приложений Android Studio уже настроена поддержка Android Design Support Library для работы с TextInputLayout и FloatingActionButton. Однако чтобы использовать компонент RecyclerView, необходимо обновить зависимости приложения и включить в них библиотеку RecyclerView.

1. Щелкните правой кнопкой мыши на папке app приложения и выберите команду Open Module Settings, чтобы открыть окно Project Structure.
2. Откройте вкладку Dependencies, щелкните на значке ➕ и выберите Library Dependency, чтобы открыть диалоговое окно Choose Library Dependency.
3. Выберите в списке библиотеку recyclerview-v7 и нажмите кнопку OK. Библиотека появляется в списке на вкладке Dependencies.
4. В окне Project Structure нажмите кнопку OK.

IDE обновляет файл build.gradle проекта (файл, отображаемый в узле Gradle Scripts проекта под именем build.gradle (Module: app)) и включает в него новую зависимость. Система сборки Gradle открывает доступ к библиотеке в вашем проекте.

8.4.4. colors.xml

В этом приложении мы изменим акцентный цвет приложения по умолчанию (используемый для EditText, TextInputLayout и FloatingActionButton) и добавим цветовой ресурс для фона области Tagged Searches. Откройте файл colors.xml и замените шестнадцатеричное значение ресурса colorAccent на #FF5722, после чего добавьте новый цветовой ресурс с именем colorTaggedSearches и значением #BBDEFB.

8.4.5. strings.xml

Добавьте строковые ресурсы из табл. 8.2 в файл strings.xml.

Таблица 8.2. Строковые ресурсы, используемые в приложении Twitter Searches

Ключ	Значение
query_prompt	Enter Twitter search query here
tag_prompt	Tag your query
save_description	Touch this button to save your tagged search
tagged_searches	Tagged Searches

Ключ	Значение
search_URL	http://mobile.twitter.com/search?q=
share_edit_delete_title	Share, Edit or Delete the search tagged as \"%s\"
cancel	Cancel
share_subject	Twitter search that might interest you
share_message	Check out the results of this Twitter search: %s
share_search	Share Search to:
confirm_message	Are you sure you want to delete the search \"%s\"?
delete	Delete

8.4.6. arrays.xml

Как упоминалось в главе 4, ресурсы массивов обычно определяются в `arrays.xml`. Создайте файл `arrays.xml` так, как описано в разделе 4.4.6, а затем добавьте в файл ресурс из табл. 8.3.

Таблица 8.3. Ресурсы строковых массивов, определяемые в файле arrays.xml

Ключ	Значение
dialog_items	Share, Edit, Delete

8.4.7. dimens.xml

Добавьте ресурс метрики из табл. 8.4 в файл `dimens.xml`.

Таблица 8.4. Ресурсы метрик в файле dimens.xml

Ключ	Значение
fab_margin_top	90dp

8.4.8. Добавление значка для кнопки сохранения

Добавьте в проект значок сохранения (из группы `Content`) из Vector Asset Studio (как это было сделано в разделе 4.4.9) — он будет использоваться как

значок FloatingActionButton. После добавления векторного значка перейдите в папку res/drawable, откройте файл XML значка и замените значение атрибута android:fillColor элемента <path> на

```
"@android:color/white"
```

Так значок будет лучше виден на фоне акцентного цвета приложения, который назначается FloatingActionButton темой приложения.

8.4.9. activity_main.xml

В этом разделе мы настроим кнопку FloatingActionButton, встроенную в шаблон приложения Blank Activity. По умолчанию на кнопке отображается значок электронной почты, и она располагается в правом нижнем углу макета MainActivity. Мы заменим значок на кнопке значком сохранения, добавленным в разделе 8.4.8, и переместим кнопку в правый верхний угол макета. Выполните следующие действия.

1. Откройте файл activity_main.xml. В режиме представления Design выберите компонент FloatingActionButton в окне Component Tree.
2. Задайте свойству contentDescription ресурс save_description, а свойству src — ресурс ic_save_24dp.

   На момент написания книги среда Android Studio не отображала свойства layout для компонентов из Android Design Support Library, поэтому все изменения таких свойств должны осуществляться непосредственно в разметке XML макета. Перейдите в режим представления Text.
3. Замените значение свойства layout_gravity с "bottom|end" на "top|end", чтобы кнопка FloatingActionButton перешла в верх макета.
4. Измените имя свойства layout_margin на layout_marginEnd, чтобы оно применялось только к правой стороне FloatingActionButton (или левой стороне для языков с направлением письма справа налево).
5. Добавьте следующую строку в элемент XML FloatingActionButton, чтобы задать новое значение верхнего поля — в результате кнопка смещается вниз от верха макета на часть графического интерфейса, определяемую в файле content_main.xml:

   ```
   android:layout_marginTop="@dimen/fab_margin_top"
   ```

8.4.10. content_main.xml

Компонент `RelativeLayout` в файле `content_main.xml` этого приложения содержит два компонента `TextInputLayout` и компонент `LinearLayout`, который в свою очередь содержит компоненты `TextView` и `RecyclerView`. При помощи макетного редактора и окна `Component Tree` сформируйте структуру макета, изображенную на рис. 8.10. В процессе создания компонентов задайте значения их свойств `id` в соответствии с иллюстрацией. Макет содержит несколько компонентов, которым значения `id` не нужны, так как код Java не обращается к ним напрямую.

Рис. 8.10. Компоненты графического интерфейса Twitter Searches со значениями свойства id

Шаг 1: Добавление компонента queryTextInputLayout и вложенного компонента EditText

Добавьте компонент `queryTextInputLayout` и его вложенный компонент `EditText`.

1. Вставьте компонент `TextInputLayout`. В режиме представления `Design` макетного редактора щелкните на компоненте `CustomView` в разделе `Custom` палитры. В открывшемся диалоговом окне начните вводить имя `TextInputLayout`, чтобы провести поиск по списку компонентов GUI. Когда IDE выделит компонент `TextInputLayout`, нажмите кнопку `OK`, а затем в окне `Component Tree` щелкните на компоненте `RelativeLayout`, чтобы вставить `TextInputLayout` как вложенный макет. Выделите компонент `TextInputLayout` и задайте его свойству `id` значение `queryTextInputLayout`.
2. Чтобы добавить компонент `EditText` в `TextInputLayout`, перейдите на вкладку `Text` макетного редактора, замените символы `/>` в конце элемента `TextInputLayout` на `>`, установите курсор справа от `>`, нажмите Enter и введите `</`. IDE автоматически вставляет завершающий тег. Между начальным и конечным тегами `TextInputLayout` введите `<EditText`. IDE открывает окно автозаполнения с выделенным компонентом `EditText`. Нажмите Enter, чтобы вставить `EditText`, задайте его свойству `layout_width` значение `match_parent`, а свойству `layout_height` значение `wrap_content`.
3. Вернитесь в режим `Design`. В окне `Component Tree` выберите компонент `EditText` и задайте его свойству `imeOptions` значение `actionNext` (на клавиатуре отображается кнопка › для перехода к следующему компоненту `EditText`). Задайте свойству `hint` строковый ресурс `query_prompt` и установите флажок свойства `singleLine`. Чтобы увидеть свойство `imeOptions`, необходимо нажать кнопку `Show expert properties` ( ) в верхней части окна свойств.

Шаг 2: Добавление компонента tagTextInputLayout и вложенного компонента EditText

Используя описание из предыдущего шага, добавьте компонент `tagTextInputLayout` и его вложенный компонент `EditText` со следующими изменениями.

1. После добавления компонента `TextInputLayout` задайте его свойству `id` значение `tagTextInputLayout`.
2. На вкладке `Text` добавьте следующую строку в элемент XML `tagTextInputLayout`, чтобы показать, что компонент `TextImputLayout` должен отображаться под `queryTextInputLayout`:

```
android:layout_below="@id/queryTextInputLayout"
```

3. На вкладке Design назначьте строковый ресурс tag_prompt свойству hint компонента EditText.
4. Задайте свойству imeOptions компонента EditText значение actionDone (на клавиатуре отображается кнопка для закрытия клавиатуры).

Шаг 3: Добавление компонента LinearLayout

Затем добавьте компонент LinearLayout под tagTextInputLayout.

1. На вкладке Design перетащите компонент LinearLayout (vertical) на узел RelativeLayout в окне Component Tree.
2. В окне свойств раскройте узел свойства layout:alignComponent, щелкните на поле значения справа от top:bottom и выберите tagTextInputLayout. Это означает, что верхний край LinearLayout должен располагаться под низом tagTextInputLayout.

Шаг 4: Добавление компонентов TextView и RecyclerView, вложенных в LinearLayout

Осталось добавить компоненты TextView и RecyclerView, вложенные в LinearLayout.

1. Перетащите компонент Medium Text на узел LinearLayout (vertical) в окне Component Tree, задайте его свойству layout:width значение match_parent, свойству text — строковый ресурс с именем tagged_searches, свойству gravity — значение center_horizontal и свойству textStyle — значение bold. Также откройте раздел padding и задайте свойствам top и bottom ресурс метрики с именем activity_vertical_margin.
2. Добавьте компонент RecyclerView. На вкладке Design макетного редактора щелкните на компоненте CustomView в разделе Custom. В открывшемся диалоговом окне начните вводить имя RecyclerView, чтобы провести поиск по списку компонентов GUI. Когда IDE выделит компонент RecyclerView, нажмите кнопку OK, а затем в окне Component Tree щелкните на компоненте LinearLayout, чтобы вставить RecyclerView как вложенное представление.
3. Выделите компонент RecyclerView в окне Component Tree. Задайте его свойству id значение recyclerView, свойству layout:width — значение match_parent, свойству layout:height — значение 0dp и свойству layout:weight — значение 1 (компонент RecyclerView должен заполнять все оставшееся вертикальное пространство в LinearLayout). Откройте раздел padding и задайте свойствам left и right ресурс метрики с именем activity_horizontal_margin.

8.4.11. Макет элемента RecyclerView: list_item.xml

При заполнении `RecyclerView` данными разработчик должен задать макет каждого элемента списка. В нашем приложении в каждом элементе списка отображается короткое имя одного сохраненного запроса. Теперь мы создадим новый макет, который содержит только `TextView` с соответствующим форматированием.

1. В окне `Project` откройте папку `res`, щелкните правой кнопкой мыши на папке `layout` и выберите команду `New ▸ Layout resource file`. На экране появляется диалоговое окно `New Resource File`.
2. В поле `File name` введите текст `list_item.xml`.
3. В поле `Root element` введите `TextView`.
4. Нажмите кнопку `OK`. Новый файл `list_item.xml` появляется в папке `res/layout`.

IDE открывает новый макет в макетном редакторе. Выделите компонент `TextView` в окне `Component Tree`, задайте его свойству `id` значение `textView`, а затем задайте следующие свойства:

- `layout:width` — `match_parent`;
- `layout:height` — `?android:attr/listPreferredItemHeight` — предопределенный ресурс Android, представляющий желательную высоту области элемента списка, реагирующей на касания[1];

СОВЕТ ПО ОФОРМЛЕНИЮ 8.1

В рекомендациях по дизайну Android указано, что минимальный размер экранной области, реагирующей на касания, составляет 48dp×48dp. За дополнительной информацией о размерах и расстояниях между элементами графического интерфейса обращайтесь по адресу https://www.google.com/design/spec/layout/metrics-keylines.html.

- `gravity` — `center_vertical`;
- `textAppearance` — `?android:attr/textAppearanceMedium` — предопределенный ресурс темы, представляющий размер шрифта для среднего текста.

[1] На момент написания книги это значение приходилось задавать напрямую в XML из-за ошибки в Android Studio, присоединявшей суффикс «dp» к значению свойства при его задании в окне свойств.

Другие предопределенные ресурсы Android

Существует много других предопределенных ресурсов Android — вроде тех, которые использовались для назначения высоты и размера шрифта элемента списка. Полный список доступен по адресу

http://developer.android.com/reference/android/R.attr.html

Чтобы использовать значение в своем макете, укажите его в формате

```
?android:attr/имяРесурса
```

8.5. Класс MainActivity

Приложение состоит из трех классов.

- Класс `MainActivity` (который будет рассматриваться в этом разделе) настраивает графический интерфейс и определяет логику приложения.
- Класс `SearchesAdapter` (субкласс `RecyclerView.Adapter` — раздел 8.6) определяет, как теги (короткие имена) запросов должны связываться с элементами `RecyclerView`. Метод `onCreate` класса `MainActivity` создает объект класса `SearchesAdapter` и назначает его адаптером `RecyclerView`.
- Класс `ItemDivider` (субкласс `RecyclerView.ItemDecoration` — раздел 8.7) используется для рисования горизонтальных линий между элементами.

В разделах 8.5.1–8.5.10 подробно рассматривается класс `MainActivity`. Этому приложению меню не нужно, поэтому мы удалили методы `MainActivity` `onCreateOptionsMenu` и `onOptionsItemSelected`, а также соответствующий ресурс меню из папки `res/menu`.

8.5.1. Команды package и import

В листинге 8.1 приведены команды `package` и `import` класса `MainActivity`. Импортируемые типы рассматриваются в разделе 8.3 по мере того, как они будут встречаться нам в классе `MainActivity`.

Листинг 8.1. Команды package и import класса MainActivity

```
1 // MainActivity.java
2 // Управление поисковыми запросами «Твиттера»
3 // и вывод результатов в браузере
4 package com.deitel.twittersearches;
5
6 import android.app.AlertDialog;
```

```
 7 import android.content.Context;
 8 import android.content.DialogInterface;
 9 import android.content.Intent;
10 import android.content.SharedPreferences;
11 import android.net.Uri;
12 import android.os.Bundle;
13 import android.support.design.widget.FloatingActionButton;
14 import android.support.design.widget.TextInputLayout;
15 import android.support.v7.app.AppCompatActivity;
16 import android.support.v7.widget.LinearLayoutManager;
17 import android.support.v7.widget.RecyclerView;
18 import android.support.v7.widget.Toolbar;
19 import android.text.Editable;
20 import android.text.TextWatcher;
21 import android.view.View;
22 import android.view.View.OnClickListener;
23 import android.view.View.OnLongClickListener;
24 import android.view.inputmethod.InputMethodManager;
25 import android.widget.EditText;
26 import android.widget.TextView;
27
28 import java.util.ArrayList;
29 import java.util.Collections;
30 import java.util.List;
31
```

8.5.2. Поля MainActivity

Как и в приложении `WeatherViewer`, класс `MainActivity` (листинг 8.2) расширяет класс `AppCompatActivity` (строка 32); это позволяет ему использовать панель приложения и другие возможности библиотеки AppCompat на устройствах с более старыми версиями Android. Статическая строковая константа `SEARCHES` (строка 34) представляет имя файла `SharedPreferences`, в котором на устройстве хранятся пары «тег—запрос».

Листинг 8.2. Поля MainActivity

```
32 public class MainActivity extends AppCompatActivity {
33    // Имя файла SharedPreferences с сохраненными запросами
34    private static final String SEARCHES = "searches";
35
36    private EditText queryEditText; // Для ввода запроса
37    private EditText tagEditText; // Для ввода тега
38    private FloatingActionButton saveFloatingActionButton; // Для сохранения
39    private SharedPreferences savedSearches; // Сохраненные запросы
40    private List<String> tags; // Список тегов сохраненных запросов
41    private SearchesAdapter adapter; // Для связывания данных с RecyclerView
42
```

В строках 36–41 определяются переменные экземпляров `MainActivity`.

- В строках 36–37 объявляются компоненты `EditTexts`, которые будут использоваться для ввода тегов и запросов.
- В строке 38 объявляется кнопка `FloatingActionButton` для сохранения запроса. В шаблоне `Blank Activity` она объявляется как локальная переменная в методе `onCreate` (раздел 8.5.3) — мы переименовали ее и сделали переменной экземпляра, чтобы кнопку можно было скрыть, когда поля `EditText` пусты, и отобразить при появлении данных.
- В строке 39 объявляется переменная экземпляра `savedSearchers` класса `SharedPreferences`. Эта переменная будет использоваться для работы с коллекцией пар «тег—запрос», представляющих сохраненные запросы пользователя.
- В строке 40 объявляется контейнер `List<String> tags` для хранения отсортированных тегов запросов.
- В строке 41 объявляется переменная экземпляра `adapter` класса `SearchesAdapter`. В ней хранится ссылка на объект субкласса `RecyclerView.Adapter`, поставляющего данные `RecyclerView`.

8.5.3. Переопределенный метод onCreate

Переопределенный метод `onCreate` класса `Activity` (листинг 8.3) инициализирует переменные экземпляров `Activity` и настраивает компоненты GUI. Строки 52–57 получают ссылки на компоненты `queryEditText` и `tagEditText`; для каждого компонента регистрируется объект `TextWatcher` (раздел 8.5.4), который получает оповещения о вводе или удалении символов в `EditText`.

Листинг 8.3. Переопределенный метод onCreate

```
43    // Настройка графического интерфейса и регистрация слушателей
44    @Override
45    protected void onCreate(Bundle savedInstanceState) {
46       super.onCreate(savedInstanceState);
47       setContentView(R.layout.activity_main);
48       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
49       setSupportActionBar(toolbar);
50
51       // Получить ссылки на EditText и добавить слушателей
52       queryEditText = ((TextInputLayout) findViewById(
53          R.id.queryTextInputLayout)).getEditText();
54       queryEditText.addTextChangedListener(textWatcher);
55       tagEditText = ((TextInputLayout) findViewById(
56          R.id.tagTextInputLayout)).getEditText();
57       tagEditText.addTextChangedListener(textWatcher);
58
59       // Получить объект SharedPreferences с сохраненными запросами
```

```
60        savedSearches = getSharedPreferences(SEARCHES, MODE_PRIVATE);
61
62        // Получить сохраненные теги в ArrayList и отсортировать их
63        tags = new ArrayList<>(savedSearches.getAll().keySet());
64        Collections.sort(tags, String.CASE_INSENSITIVE_ORDER);
65
66        // Получить ссылку на RecyclerView и настроить его
67        RecyclerView recyclerView =
68           (RecyclerView) findViewById(R.id.recyclerView);
69
70        // Получить LinearLayoutManager для вертикального списка
71        recyclerView.setLayoutManager(new LinearLayoutManager(this));
72
73        // Создать RecyclerView.Adapter для связывания тегов с RecyclerView
74        adapter = new SearchesAdapter(
75           tags, itemClickListener, itemLongClickListener);
76        recyclerView.setAdapter(adapter);
77
78        // Назначить ItemDecorator для рисования линий между элементами
79        recyclerView.addItemDecoration(new ItemDivider(this));
80
81        // Зарегистрировать слушателя для сохранения или редактирования
82        saveFloatingActionButton =
83           (FloatingActionButton) findViewById(R.id.fab);
84        saveFloatingActionButton.setOnClickListener(saveButtonListener);
85        updateSaveFAB(); // Скрыть кнопку, потому что поля EditText пусты
86     }
87
```

Получение объекта SharedPreferences

В строке 60 метод `getSharedPreferences` (унаследованный косвенно от класса `Context`) используется для получения объекта `SharedPreferences`, который позволяет читать из файла `searches` пары «тег—запрос» (если они существуют). Первый аргумент определяет имя файла с данными. Второй аргумент определяет уровень доступа к файлу и может принимать следующие значения:

- `MODE_PRIVATE` — файл доступен только для этого приложения. В большинстве случаев используется именно этот режим;
- `MODE_WORLD_READABLE` — любое приложение на устройстве может читать данные из файла;
- `MODE_WORLD_WRITABLE` — любое приложение на устройстве может записывать данные в файл.

Эти константы могут объединяться поразрядным оператором OR (|).

Получение ключей, хранящихся в объекте SharedPreferences

Теги поисковых запросов удобнее выводить в алфавитном порядке, чтобы пользователю было проще найти нужный запрос. Сначала строка 63 получает

представления ключей из объекта SharedPreferences и сохраняет их в tags (ArrayList<String>). Метод getAll класса SharedPreferences возвращает все сохраненные запросы в виде Map (пакет java.util) — коллекции пар «ключ—значение». Вызов метода keyset для объекта Map возвращает все ключи в виде Set<String> (пакет java.util) — коллекции уникальных значений. Результат используется для инициализации tags.

Сортировка списка тегов

В строке 64 содержимое tags сортируется вызовом Collections.sort. Так как пользователь может вводить теги с использованием символов как верхнего, так и нижнего регистра, сортировка проводится без учета регистра символов. Для этого во втором аргументе Collections.sort передается предопределенный объект String.CASE_INSENSITIVE_ORDER.

Настройка RecyclerView

В строках 67–79 происходит настройка RecyclerView.

- Строки 67–68 получают ссылку на RecyclerView.
- Компонент RecyclerView может разместить свои отображаемые элементы многими разными способами. В этом приложении будет использоваться объект LinearLayoutManager для отображения элементов в вертикальном списке. Конструктор LinearLayoutManager получает объект Context (в данном случае MainActivity). Строка 71 создает объект LinearLayoutManager и вызывает для него метод setLayoutManager класса RecyclerView, чтобы назначить новый объект менеджером макета RecyclerView.
- Строки 74–75 создают объект SearchesAdapter (субкласс RecyclerView.Adapter — раздел 8.6), который будет поставлять данные для отображения в RecyclerView. Строка 76 вызывает метод setAdapter класса RecyclerView, указывая тем самым, что SearchesAdapter будет поставлять данные для RecyclerView.
- Строка 79 создает субкласс RecyclerView.ItemDecoration с именем ItemDivider (раздел 8.7) и передает его методу addItemDecoration класса RecyclerView. Это делается для того, чтобы компонент RecyclerView рисовал горизонтальные линии между элементами списков.

Регистрация слушателя для FloatingActionButton

Строки 82–85 получает ссылку на saveFloatingActionButton и регистрируют слушателя OnClickListener. Переменная экземпляра saveButtonListener содержит ссылку на объект анонимного внутреннего класса, реализующий интерфейс View.OnClickListener (раздел 8.5.5). Строка 85 вызывает метод updateSaveFAB

(раздел 8.5.4), который изначально скрывает кнопку `saveFloatingActionButton`, потому что поля `EditText` пусты при создании `onCreate` — кнопка отображается только в том случае, если оба поля содержат введенные данные.

8.5.4. Обработчик событий TextWatcher и метод updateSaveFAB

Листинг 8.4 определяет анонимный внутренний класс, реализующий интерфейс `TextWatcher` (строки 89–103). Метод `onTextChanged` класса `TextWatcher` вызывает `updateSaveFAB` при изменении содержимого любого из полей `EditText`. Строки 54 и 57 (листинг 8.3) регистрируют переменную экземпляра `textWatcher` как слушателя событий `EditText`.

Листинг 8.4. Обработчик события TextWatcher и метод updateSaveFAB

```
88    // Управление состоянием кнопки saveFloatingActionButton
89    private final TextWatcher textWatcher = new TextWatcher() {
90       @Override
91       public void beforeTextChanged(CharSequence s, int start, int count,
92          int after) { }
93
94       // Проверка состояния saveFloatingActionButton после ввода данных
95       @Override
96       public void onTextChanged(CharSequence s, int start, int before,
97          int count) {
98          updateSaveFAB();
99       }
100
101       @Override
102       public void afterTextChanged(Editable s) { }
103    };
104
105    // Управление видимостью saveFloatingActionButton
106    private void updateSaveFAB() {
107       // Проверить, присутствуют ли данные в обоих компонентах EditText
108       if (queryEditText.getText().toString().isEmpty() ||
109          tagEditText.getText().toString().isEmpty())
110          saveFloatingActionButton.hide();
111       else
112          saveFloatingActionButton.show();
113    }
114
```

Метод `updatedSaveFAB` (листинг 8.4, строки 106–113) проверяет, присутствует ли текст в обоих компонентах `EditText` (строки 108–109). Если хотя бы одно из двух полей `EditText` пусто, строка 110 скрывает кнопку вызовом метода `hide` класса `FloatingActionButton`, потому что для сохранения пары «тег—запрос» должны быть введены обе составляющие. Если оба поля содержат текст,

строка 112 вызывает метод `show` класса `FloatingActionButton` для отображения кнопки. Касаясь кнопки, пользователь сохраняет пару «тег—запрос».

8.5.5. Слушатель OnClickListener

Листинг 8.5 определяет переменную экземпляра `saveButtonListener`, которая содержит ссылку на объект анонимного внутреннего класса, реализующий интерфейс `OnClickListener`. Строка 84 (листинг 8.3) регистрирует `saveButtonListener` как обработчик события `saveFloatingActionButton`. Строки 119–135 (листинг 8.5) переопределяют метод `onClick` интерфейса `OnClickListener`. Строки 121–122 получают строки из компонентов `EditText`. Если пользователь ввел запрос и тег (строка 124):

- строки 126–128 скрывают экранную клавиатуру;
- строка 130 вызывает метод `addTaggedSearch` (раздел 8.5.6) для сохранения пары «тег—запрос»;
- строки 131–132 очищают содержимое двух компонентов `EditText`;
- строка 133 вызывает метод `requestFocus` компонента `queryEditText` для передачи курсора в `queryEditText`.

Листинг 8.5. Анонимный внутренний класс, реализующий интерфейс OnClickListener кнопки saveButton для сохранения нового или обновленного запроса

```
115    // saveButtonListener сохраняет пару "тег—запрос" в SharedPreferences
116    private final OnClickListener saveButtonListener =
117       new OnClickListener() {
118          // добавить/обновить данные, если оба поля содержат данные
119          @Override
120          public void onClick(View view) {
121             String query = queryEditText.getText().toString();
122             String tag = tagEditText.getText().toString();
123
124             if (!query.isEmpty() && !tag.isEmpty()) {
125                // Скрыть экранную клавиатуру
126                ((InputMethodManager) getSystemService(
127                  Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
128                      view.getWindowToken(), 0);
129
130                addTaggedSearch(tag, query); // добавить/обновить запрос
131                queryEditText.setText(""); // очистить queryEditText
132                tagEditText.setText(""); // очистить tagEditText
133                queryEditText.requestFocus(); // queryEditText получает фокус
134             }
135          }
136       };
137
```

8.5.6. Метод addTaggedSearch

Обработчик события в листинге 8.5 вызывает метод `addTaggedSearch` (листинг 8.6) для добавления нового поиска в `savedSearches` или изменения существующего поиска.

Листинг 8.6. Метод addTaggedSearch класса MainActivity

```
138    // Добавление нового запроса с обновлением всех кнопок
139    private void addTaggedSearch(String tag, String query) {
140       // Получение SharedPreferences.Editor для сохранения новой пары
141       SharedPreferences.Editor preferencesEditor = savedSearches.edit();
142       preferencesEditor.putString(tag, query); // Сохранение текущего запроса
143       preferencesEditor.apply(); // Сохранение обновленных настроек
144
145       // Если тег новый, добавить его, отсортировать и вывести список
146       if (!tags.contains(tag)) {
147          tags.add(tag); // Добавить новый тег
148          Collections.sort(tags, String.CASE_INSENSITIVE_ORDER);
149          adapter.notifyDataSetChanged(); // Обновление тегов в RecyclerView
150       }
151    }
152
```

Редактирование данных в объекте SharedPreferences

Вспомните, о чем говорилось в разделе 4.6.7: чтобы изменить содержимое объекта `SharedPreferences`, необходимо сначала вызвать метод `edit` для получения объекта `SharedPreferences.Editor` (листинг 8.5, строка 141). В этот объект можно добавлять пары «ключ—значение», удалять из него пары «ключ—значение», а также изменять значение, связанное с конкретным ключом в файле `SharedPreferences`. Строка 142 вызывает метод `putString` объекта `SharedPreferences.Editor` для сохранения тега (ключ) и запроса (ассоциированное значение) — если тег уже существует в `SharedPreferences`, это приводит к обновлению значения. Строка 143 закрепляет изменения вызовом метода `apply` класса `SharedPreferences.Editor`.

Оповещение RecyclerView.Adapter об изменении данных

Когда пользователь добавляет новый запрос, необходимо обновить компонент `RecyclerView`, чтобы он появился в списке. Строка 146 проверяет, был ли добавлен новый тег. В этом случае строки 147–148 добавляют тег нового запроса в `tags`, а затем сортируют содержимое `tags`. Строка 149 вызывает метод `notifyDataSetChanged` класса `RecyclerView.Adapter`, чтобы сообщить об изменении данных в `tags`. Как и в случае с адаптером `ListView`, `RecyclerView.Adapter` затем оповещает `RecyclerView` для обновления списка отображаемых элементов.

8.5.7. Анонимный внутренний класс, реализующий View.OnClickListener для вывода результатов поиска

В листинге 8.7 определяется переменная экземпляра `itemClickListener`, содержащая ссылку на объект анонимного внутреннего класса, реализующий интерфейс `OnClickListener` (вложенный интерфейс класса `View`). В строках 156–168 переопределяется метод `onClick` интерфейса. В аргументе метода передается объект `View`, к которому притронулся пользователь, — в данном случае компонент `TextView`, отображающий тег поискового запроса в `RecyclerView`.

Листинг 8.7. Анонимный внутренний класс, реализующий интерфейс View.OnClickListener для вывода результатов поиска

```
153    // itemClickListener запускает браузер для вывода результатов поиска
154    private final OnClickListener itemClickListener =
155       new OnClickListener() {
156          @Override
157          public void onClick(View view) {
158             // Получение строки запроса и создание URL для этого запроса
159             String tag = ((TextView) view).getText().toString();
160             String urlString =  getString(R.string.search_URL) +
161                Uri.encode(savedSearches.getString(tag, ""), "UTF-8");
162
163             // Создание интента для запуска браузера
164             Intent webIntent = new Intent(Intent.ACTION_VIEW,
165                Uri.parse(urlString));
166
167             startActivity(webIntent); // Вывести результаты в браузере
168          }
169       };
170
```

Получение строковых ресурсов

Строка 159 получает текст объекта `View`, к которому прикоснулся пользователь в компоненте `RecyclerView` — этот текст содержит тег поискового запроса. В строках 160–161 создается объект `String` с поисковым URL-адресом «Твиттера» и выполняемым запросом. Строка 160 вызывает метод `getString`, унаследованный от `Activity`, передавая ему строковый ресурс `search_URL`.

Получение строк из объекта SharedPreferences

Результат строки 161 присоединяется к поисковому URL-адресу, и образуется строка `urlString`. Метод `getString` класса `SharedPreferences` возвращает запрос, связанный с тегом. Если тег еще не существует, то возвращается второй аргумент (`""` в данном случае). Строка 161 передает строку запроса методу `encode` класса `Uri`, который *экранирует* (escapes) все специальные символы (такие как `?`, `/`, `:` и т. д.) и возвращает так называемую *строку в URL-кодировке*. Класс `Uri`

(Uniform Resource Identifier) из пакета `android.net` позволяет преобразовать URL в формат, необходимый интенту для запуска браузера на устройстве[1]. Очень важно, чтобы веб-сервер «Твиттера» смог правильно разобрать URL и получить поисковый запрос.

Создание интента для запуска браузера на устройстве

В строках 164–165 создается объект `Intent`, который будет использоваться для запуска браузера на устройстве и вывода результатов поиска. В главе 4 явный интент использовался для запуска другой активности в том же приложении. На этот раз мы используем неявный интент для запуска другой активности. В первом аргументе конструктора `Intent` передается константа, описывающая выполняемое действие. Константа `Intent.ACTION_VIEW` означает, что мы хотим вывести представление данных интента. Класс `Intent` определяет много других констант, описывающих такие действия, как поиск, выбор, отправка и воспроизведение:

http://developer.android.com/reference/android/content/Intent.html

Второй аргумент (строка 165) содержит объект `Uri`, представляющий данные, с которыми выполняется действие. Метод `parse` класса `Uri` преобразует строку, представляющую URL (Uniform Resource Locator), в `Uri`.

Запуск активности для интента

Строка 167 передает объект `Intent` методу `startActivity`, унаследованному от `Activity`. Метод запускает активность, которая может выполнить указанное действие для указанных данных. В данной ситуации, поскольку было выбрано действие просмотра URI, интент запускает браузер устройства для отображения веб-страницы с результатами переданного поискового запроса «Твиттера».

8.5.8. Анонимный внутренний класс, реализующий интерфейс View.OnLongClickListener для пересылки, изменения и удаления запросов

В листинге 8.8 определяется переменная экземпляра `itemLongClickListener`, которая содержит ссылку на объект анонимного внутреннего класса, реализующий интерфейс `OnLongClickListener`. В строках 175–216 переопределяется метод `onLongClick` интерфейса `OnLongClickListener`.

---

[1] URI однозначно идентифицирует сетевой ресурс. Одной из распространенных разновидностей URI является URL-адрес (Uniform Resource Locator) для идентификации объектов Всемирной паутины: веб-страниц, файлов изображений, методов веб-сервисов и т. д.

Листинг 8.8. Анонимный внутренний класс, реализующий интерфейс View.OnLongClickListener

```
171    // itemLongClickListener отображает диалоговое окно для пересылки,
172    // изменения или удаления сохраненного запроса
173    private final OnLongClickListener itemLongClickListener =
174       new OnLongClickListener() {
175          @Override
176          public boolean onLongClick(View view) {
177             // Получение тега, на котором было сделано длинное нажатие
178             final String tag = ((TextView) view).getText().toString();
179
180             // Создание нового объекта AlertDialog
181             AlertDialog.Builder builder =
182                new AlertDialog.Builder(MainActivity.this);
183
184             // Назначение заголовка AlertDialog
185             builder.setTitle(
186                getString(R.string.share_edit_delete_title, tag));
187
188             // Назначение списка вариантов и создание обработчика
189             builder.setItems(R.array.dialog_items,
190                new DialogInterface.OnClickListener() {
191                   @Override
192                   public void onClick(DialogInterface dialog, int which) {
193                         switch (which) {
194                         case 0: // share
195                            shareSearch(tag);
196                            break;
197                         case 1: // edit
198                            // Заполнение EditText для тега и запроса
199                            tagEditText.setText(tag);
200                            queryEditText.setText(
201                               savedSearches.getString(tag, ""));
202                            break;
203                         case 2: // delete
204                            deleteSearch(tag);
205                            break;
206                      }
207                   }
208                }
209             );
210
211             // Назначение негативной кнопки AlertDialog
212             builder.setNegativeButton(getString(R.string.cancel), null);
213
214             builder.create().show(); // Отображение AlertDialog
215             return true;
216          }
217       };
218
```

Локальные переменные для анонимных внутренних классов

Строка 178 получает текст элемента списка, на котором пользователь сделал *длинное нажатие*, и присваивает его локальной переменной `tag` со

спецификатором `final`. Любые локальные переменные или параметры методов, которые будут использоваться в анонимном внутреннем классе, должны быть объявлены со спецификатором `final`.

Окно AlertDialog для вывода списка

В строках 181–186 создается объект `AlertDialog.Builder`, а заголовком диалогового окна назначается отформатированная строка, в которой значение `tag` заменяет спецификатор формата. Строка 186 вызывает унаследованный от `Activity` метод `getString`, который получает несколько аргументов. Первый аргумент содержит идентификатор строкового ресурса, представляющего форматную строку, а в остальных передаются значения, которые должны заменить спецификаторы формата в форматной строке. Кроме кнопок, окно `AlertDialog` может вывести список вариантов. Строки 189–209 используют метод `setItem` класса `AlertDialog.Builder`, чтобы указать, что в окне должно отображаться содержимое массива строк `R.array.dialog_items`; кроме того, в них определяется анонимный внутренний класс для обработки касаний к элементам списка.

Обработчик событий для списка элементов диалогового окна

Анонимный внутренний класс в строках 190–208 определяет, какой элемент был выбран пользователем из списка диалогового окна, и выполняет соответствующее действие. Если пользователь выбрал команду `Share`, то вызывается функция `shareSearch` (строка 195). Если пользователь выбрал команду `Edit`, то в строках 199–201 выводится запрос и тег в компонентах `EditText`. Если пользователь выбрал команду `Delete`, то вызывается `deleteSearch` (строка 204).

Настройка негативной кнопки и отображение диалогового окна

В строке 212 настраивается негативная кнопка диалогового окна. Если обработчик события негативной кнопки равен `null`, негативная кнопка просто закрывает диалоговое окно. Строка 214 создает и отображает диалоговое окно.

8.5.9. Метод shareSearch

Метод `shareSearch` (листинг 8.9) вызывается при выборе команды пересылки запроса. В строках 222–223 создается объект `String`, представляющий пересылаемый запрос. Строки 226–232 создают и настраивают объект `Intent` для отправки URL-адреса с использованием активности, которая может обработать `Intent.ACTION_SEND`.

Листинг 8.9. Метод shareSearch класса MainActivity

```
219    // Выбор приложения для пересылки URL-адреса сохраненного запроса
220    private void shareSearch(String tag) {
221       // Создание URL-адреса, представляющего поисковый запрос
222       String urlString = getString(R.string.search_URL) +
223          Uri.encode(savedSearches.getString(tag, ""), "UTF-8");
224
225       // Создание объекта Intent для пересылки
226       Intent shareIntent = new Intent();
227       shareIntent.setAction(Intent.ACTION_SEND);
228       shareIntent.putExtra(Intent.EXTRA_SUBJECT,
229          getString(R.string.share_subject));
230       shareIntent.putExtra(Intent.EXTRA_TEXT,
231          getString(R.string.share_message, urlString));
232       shareIntent.setType("text/plain");
233
234       // Вывод списка приложений с возможностью пересылки текста
235       startActivity(Intent.createChooser(shareIntent,
236          getString(R.string.share_search)));
237    }
238
```

Включение дополнительной информации в интент

Объект `Intent` содержит контейнер `Bundle` с *дополнениями* (extras), которые должны передаваться объекту `Activity`, обрабатывающему интент. Например, активность электронной почты может получать дополнения, представляющие собой тему сообщения, адреса CC и BCC, а также текст сообщения. В строках 228–231 метод `putExtra` класса `Intent` используется для добавления пары «ключ—значение» в контейнер `Bundle`, связанный с `Intent`. В первом аргументе метода содержится ключ `String`, представляющий назначение дополнения, а во втором — дополнительные данные. Дополнения могут быть значениями примитивных типов, массивами примитивных типов, объектами `Bundle` и не только — за полным списком перегруженных версий `putExtra` обращайтесь к документации класса `Intent`.

Дополнение в строках 228–229 задает тему сообщения в виде строкового ресурса `R.string.shareSubject`. Для активности, *не использующей* тему (например, при публикации в социальной сети), это дополнение игнорируется. Дополнение в строках 230–231 представляет пересылаемый текст — отформатированную строку, в которой значение `urlString` подставляется в строковый ресурс `R.string.shareMessage`. Строка 232 назначает интенту тип MIME `text/plain` — такие данные могут обрабатываться любой активностью, способной отправлять простые текстовые сообщения.

Выбор интента

Чтобы вывести окно выбора интента, показанное на рис. 8.7, *а*, мы передаем объект `Intent` и строку заголовка статическому методу `createChooser` класса

Intent (строки 235–236). Заголовок окна выбора задается вторым аргументом (R.string.shareSearch). Важно задать этот заголовок, чтобы напомнить пользователю о выборе соответствующей активности. Вы не можете управлять ни приложениями, установленными на телефоне пользователя, ни фильтрами интентов, которые могут запускать эти приложения, поэтому в окне выбора могут появиться несовместимые активности. Метод createChooser возвращает объект Intent, который передается startActivity для отображения окна выбора.

8.5.10. Метод deleteSearch

Метод deleteSearch (листинг 8.10) вызывается тогда, когда пользователь выполняет длинное нажатие на теге и выбирает команду Delete. Перед удалением запроса приложение выводит окно AlertDialog для подтверждения операции. В строке 243 (листинг 8.10) в заголовке диалогового окна назначается отформатированная строка, в которой значение tag заменяет форматный спецификатор в строковом ресурсе R.string.confirm_Message. В строке 246 настраивается негативная кнопка для закрытия диалогового окна. В строках 249–264 настраивается позитивная кнопка для удаления запроса. В строке 252 тег удаляется из коллекции tags, а строки 255–258 используют объект SharedPreferences.Editor для удаления запроса из объекта SharedPreferences приложения. Затем строка 261 оповещает ArrayAdapter об изменении используемых данных, чтобы компонент RecyclerView мог обновить свой список.

Листинг 8.10. Метод deleteSearch класса MainActivity

```
239    // Удаление запроса после подтверждения операции пользователем
240    private void deleteSearch(final String tag) {
241       // Создание нового объекта AlertDialog и назначение сообщения
242       AlertDialog.Builder confirmBuilder = new AlertDialog.Builder(this);
243       confirmBuilder.setMessage(getString(R.string.confirm_message, tag));
244
245       // Настройка негативной кнопки (CANCEL)
246       confirmBuilder.setNegativeButton(getString(R.string.cancel), null);
247
248       // Настройка позитивной кнопки (DELETE)
249       confirmBuilder.setPositiveButton(getString(R.string.delete),
250          new DialogInterface.OnClickListener() {
251             public void onClick(DialogInterface dialog, int id) {
252                tags.remove(tag); // Удаление тега из tags
253
254                // Получение SharedPreferences.Editor для удаления запроса
255                SharedPreferences.Editor preferencesEditor =
256                   savedSearches.edit();
257                preferencesEditor.remove(tag); // Удаление запроса
258                preferencesEditor.apply(); // Сохранение изменений
259
260                // Повторное связывание для вывода обновленного списка
```

```
261                adapter.notifyDataSetChanged();
262             }
263          }
264       );
265
266       confirmBuilder.create().show(); // Отображение AlertDialog
267    }
268 }
```

8.6. Класс SearchesAdapter

В этом разделе представлен субкласс `RecyclerView.Adapter`, который связывает элементы из коллекции `List<String>` с именем `tags`, связывающей элементы `tags` с компонентом `RecyclerView`.

8.6.1. Команды package, import, переменные экземпляров и конструктор

В листинге 8.11 приведено начало определения класса `SearchesAdapter`. Класс расширяет обобщенный класс `RecyclerView.Adapter`, указывая в качестве аргумента-типа вложенный класс `SearchesAdapter.ViewHolder` (см. раздел 8.6.2). Переменные экземпляров в строках 17–18 содержат ссылки на слушателей событий (определяемых в классе `MainActivity`), регистрируемых для каждого элемента `RecyclerView`. Переменная экземпляра в строке 21 содержит ссылку на коллекцию `List<String>` класса `MainActivity`, в которой хранятся имена тегов.

Листинг 8.11. Команды package, import, переменные экземпляров и конструктор

```
 1 // SearchesAdapter.java
 2 // Субкласс RecyclerView.Adapter для связывания данных с элементами RecyclerView
 3 package com.deitel.twittersearches;
 4
 5 import android.support.v7.widget.RecyclerView;
 6 import android.view.LayoutInflater;
 7 import android.view.View;
 8 import android.view.ViewGroup;
 9 import android.widget.TextView;
10
11 import java.util.List;
12
13 public class SearchesAdapter
14    extends RecyclerView.Adapter<SearchesAdapter.ViewHolder> {
15
16    // Слушатели MainActivity, регистрируемые для каждого элемента списка
17    private final View.OnClickListener clickListener;
18    private final View.OnLongClickListener longClickListener;
19
```

```
20    // List<String> для хранения данных элементов RecyclerView
21    private final List<String> tags; // Поисковые запросы
22
23    // Конструктор
24    public SearchesAdapter(List<String> tags,
25       View.OnClickListener clickListener,
26       View.OnLongClickListener longClickListener) {
27       this.tags = tags;
28       this.clickListener = clickListener;
29       this.longClickListener = longClickListener;
30    }
31
```

8.6.2. Вложенный класс ViewHolder

Каждый элемент `RecyclerView` должен быть «упакован» в отдельный объект `RecyclerView.ViewHolder`. Для нашего приложения определяется субкласс `RecyclerView.ViewHolder` с именем `ViewHolder` (листинг 8.12). Конструктор `ViewHolder` (строки 39–48) получает объект `View` и слушателей событий `OnClick` и `OnLongClick` этого объекта. Объект `View` представляет элемент списка `RecyclerView`, передаваемый конструктору суперкласса (строка 42). Строка 43 сохраняет ссылку на компонент `TextView` элемента. Строка 46 регистрирует слушателя `OnClickListener` для `TextView`, который выводит результаты поиска для тега, содержащегося в `TextView`. Строка 47 регистрирует слушателя `OnLongClickListener` компонента `TextView`, который открывает диалоговое окно с командами пересылки, изменения и удаления тега из `TextView`. Конструктор вызывается тогда, когда `RecyclerView.Adapter` создает новый элемент списка (раздел 8.6.3).

Листинг 8.12. Вложенный класс ViewHolder

```
32    // Вложенный субкласс RecyclerView.ViewHolder используется для
33    // реализации паттерна View-Holder в контексте RecyclerView--
34    // логики повторного использования представлений
35    public static class ViewHolder extends RecyclerView.ViewHolder  {
36       public final TextView textView;
37
38       // Настройка объекта ViewHolder элемента RecyclerView
39       public ViewHolder(View itemView,
40          View.OnClickListener clickListener,
41          View.OnLongClickListener longClickListener) {
42          super(itemView);
43          textView = (TextView) itemView.findViewById(R.id.textView);
44
45          // Связывание слушателей с itemView
46          itemView.setOnClickListener(clickListener);
47          itemView.setOnLongClickListener(longClickListener);
48       }
49    }
50
```

8.6.3. Переопределенные методы RecyclerView.Adapter

В листинге 8.13 определяются переопределенные методы `RecyclerView.Adapter`: `onCreateViewHolder` (строки 52–61), `onBindViewHolder` (строки 64–67) и `getItemCount` (строки 70–73).

Листинг 8.13. Переопределенные методы onCreateViewHolder, onBindViewHolder и getItemCount

```
51    // Создает новый элемент списка и его объект ViewHolder
52    @Override
53    public ViewHolder onCreateViewHolder(ViewGroup parent,
54       int viewType) {
55       // Заполнение макета list_item
56       View view = LayoutInflater.from(parent.getContext()).inflate(
57          R.layout.list_item, parent, false);
58
59       // Создание ViewHolder для текущего элемента
60       return (new ViewHolder(view, clickListener, longClickListener));
61    }
62
63    // Назначение текста элемента списка для вывода тега запроса
64    @Override
65       public void onBindViewHolder(ViewHolder holder, int position) {
66       holder.textView.setText(tags.get(position));
67    }
68
69    // Возвращение количества элементов, связываемых через адаптер
70    @Override
71    public int getItemCount() {
72       return tags.size();
73    }
74 }
```

Переопределение метода onCreateViewHolder

Компонент `RecyclerView` вызывает метод `onCreateViewHolder` своего объекта `RecyclerView.Adapter` (строки 52–61) для заполнения макета каждого элемента `RecyclerView` (строки 56–57) и упаковки его в объект субкласса `RecyclerView.ViewHolder` с именем `ViewHolder` (строка 60). Новый объект `ViewHolder` возвращается `RecyclerView` для отображения.

Переопределение метода onBindViewHolder

Компонент `RecyclerView` вызывает метод `onBindViewHolder` своего объекта `RecyclerView.Adapter` (строки 64–67) для назначения данных, отображаемых для конкретного элемента `RecyclerView`. Метод получает:

- объект субкласса `RecyclerView.ViewHolder` с представлениями `View`, в которых будут отображаться данные (в данном случае один компонент `TextView`);
- значение `int`, представляющее позицию элемента в `RecyclerView`.

Строка 66 заполняет текст `TextView` строкой из `tags`, находящейся в заданной позиции.

Переопределение метода getItemCount

`RecyclerView` вызывает метод `getItemCount` своего объекта `RecyclerView.Adapter` (строки 70–73) для получения общего количества элементов, которые должны отображаться компонентом `RecyclerView`, — в данном случае количества элементов в `tags` (строка 72).

8.7. Класс ItemDivider

Объект `RecyclerView.ItemDecoration` рисует декоративные элементы (например, разделители строк) в компоненте `RecyclerView`. Класс `ItemDivider` (являющийся субклассом `RecyclerView.ItemDecoration`) (листинг 8.14) рисует разделительные линии между элементами списка. Строки 17–18 конструктора получают предопределенный ресурс Android `android.R.attr.listDivider`, который используется по умолчанию в `ListView`.

Листинг 8.14

```
1  // ItemDivider.java
2  // Класс определяет разделители, отображаемые между элементами
3  // RecyclerView; см. пример реализации bit.ly/DividerItemDecoration
4  package com.deitel.twittersearches;
5
6  import android.content.Context;
7  import android.graphics.Canvas;
8  import android.graphics.drawable.Drawable;
9  import android.support.v7.widget.RecyclerView;
10 import android.view.View;
11
12  class ItemDivider extends RecyclerView.ItemDecoration {
13    private final Drawable divider;
14
15    // Конструктор загружает встроенный разделитель элементов списка
16    public ItemDivider(Context context) {
17       int[] attrs = {android.R.attr.listDivider};
18       divider = context.obtainStyledAttributes(attrs).getDrawable(0);
19    }
20
21    // Рисование разделителей элементов списка в RecyclerView
22    @Override
23    public void onDrawOver(Canvas c, RecyclerView parent,
24       RecyclerView.State state) {
25       super.onDrawOver(c, parent, state);
26
27       // Вычисление координат x для всех разделителей
28       int left = parent.getPaddingLeft();
```

```
29       int right = parent.getWidth() - parent.getPaddingRight();
30
31       // Для каждого элемента, кроме последнего, нарисовать линию
32       for (int i = 0; i < parent.getChildCount() - 1; ++i) {
33          View item = parent.getChildAt(i); // Получить i-й элемент списка
34
35          // Вычисление координат у текущего разделителя
36          int top = item.getBottom() + ((RecyclerView.LayoutParams)
37             item.getLayoutParams()).bottomMargin;
38          int bottom = top + divider.getIntrinsicHeight();
39
40          // Рисование разделителя с вычисленными границами
41          divider.setBounds(left, top, right, bottom);
42          divider.draw(c);
43       }
44    }
45 }
```

Переопределение метода onDrawOver

В процессе прокрутки содержимого `RecyclerView` содержимое списка постоянно перерисовывается, а элементы отображаются в новых позициях экрана. Частью процесса перерисовки является вызов метода `onDrawOver` объекта `RecyclerView.ItemDecoration` компонентом `RecyclerView` (строки 22–44) для прорисовки декоративных элементов. Метод получает следующие аргументы:

- объект `Canvas` для рисования декоративных элементов на `RecyclerView`;
- объект `RecyclerView`, на котором рисуется содержимое `Canvas`;
- `RecyclerView.State` — объект с информацией, передаваемой между разными компонентами `RecyclerView`. В этом приложении значение просто передается методу `onDrawOver` суперкласса (строка 25).

Строки 28–29 вычисляют левую и правую координаты *x*, определяющие границы выводимого объекта `Drawable`. Левая координата *x* определяется вызовом метода `getPaddingLeft` класса `RecyclerView`, который возвращает величину отступа между левым краем `RecyclerView` и его содержимым. Правая координата *x* определяется вызовом метода `getWidth` компонента `RecyclerView` и вычитанием результата вызова метода `getPaddingRight` класса `RecyclerView`, который возвращает величину отступа между правым краем `RecyclerView` и его содержимым.

Строки 32–43 рисуют разделители на объекте `Canvas` компонента `RecyclerView`; для этого перебираются все элементы, кроме последнего, и под каждым из них выводится разделитель. Строка 33 получает и сохраняет текущий элемент `RecyclerView`. Строки 36–37 вычисляют верхнюю координату у одного разделителя, прибавляя к нижней координате *y* величину нижнего отступа. Строка 38 вычисляет нижнюю координату *y* разделителя как сумму верхней координаты *y* и высоты разделителя (возвращаемой методом `getIntrinsicHeight` объекта `Drawable`). Строка 41 задает границы разделителя, а строка 42 рисует его на `Canvas`.

8.8. Fabric: новая платформа мобильной разработки

В главе 7 мы использовали REST-совместимые веб-сервисы для получения прогноза погоды. «Твиттер» предоставляет обширный набор веб-сервисов для интеграции функциональности «Твиттера» в приложения. Для использования этих веб-сервисов необходима учетная запись разработчика «Твиттера» и специальная аутентификация. Впрочем, использование веб-сервисов «Твиттера» не является основной темой этой главы, поэтому приложение выполняет поисковые запросы так, как если бы они были введены непосредственно на сайте «Твиттера». Сайт возвращает результаты прямо браузеру устройства для отображения.

Напрямую работать с веб-сервисами «Твиттера» средствами, описанными в главе 7, достаточно сложно. Понимая это, «Твиттер» предоставляет *Fabric* — мощную платформу мобильной разработки для Android и iOS. Fabric инкапсулирует подробности работы с веб-сервисами «Твиттера» в библиотеках, которые встраиваются в проект и упрощают включение функциональности «Твиттера» в приложения. Также возможно включение механизма управления мобильной идентификацией (Digits), средств монетизации за счет показа рекламы (MoPub) и отправки отчетов о сбоях приложений (Crashlytics).

Чтобы использовать Fabric, зарегистрируйтесь по адресу

https://get.fabric.io/

и установите плагин для Android Studio. Щелкните на значке плагина на панели инструментов Android Studio и пройдите серию шагов по добавлению библиотек Fabric в проект. На сайте также доступна обширная документация и учебники по Fabric.

8.9. Резюме

В этой главе мы создали приложение `Twitter Searches`. Для хранения и управления парами «ключ—значение», представляющими поисковые запросы «Твиттера», используется файл `SharedPreferences`.

Вы познакомились с `RecyclerView` (из пакета `android.support.v7.widget`) — гибким компонентом с широкими возможностями настройки, который позволяет управлять выводом списка с возможностью прокрутки. Компонент `RecyclerView` поддерживает менеджеры макетов; для вертикального размещения элементов `RecyclerView` этого приложения использовался класс `LinearLayoutManager` — субкласс `RecyclerView.LayoutManager`.

Для повторного использования представлений, выходящих за границы экрана, снова использовался паттерн View-Holder. Вы узнали, что компонент `RecyclerView` формализует паттерн View-Holder и делает его обязательным. Мы создали субкласс `RecyclerView.Adapter` для связывания элементов списка `RecyclerView` с данными. Также был создан субкласс `RecyclerView.ViewHolder` для хранения ссылок на представления всех элементов списка, обеспечивающих возможность их повторного использования. Чтобы между элементами `RecyclerView` выводились разделительные линии, мы определили субкласс `RecyclerView.ItemDecoration`.

В приложении были использованы два неявных интента, для которых не указываются конкретные компоненты, выполняющие их обработку. Один интент запускает браузер устройства для отображения результатов поиска в «Твиттере», а другой выводит окно для выбора приложений, поддерживающих пересылку текста.

Наконец, для отображения списка команд, из которых пользователь может выбрать только одну, был использован объект `AlertDialog`. Метод `setItems` класса `AlertDialog.Builder` использовался для определения ресурса строкового массива с именами команд и назначения обработчика события, который должен вызываться при выборе варианта.

В главе 9 будет построено приложение `Address Book`, работающее с базой данных и предоставляющее простой и быстрый доступ к списку контактов, с возможностью добавления, удаления и редактирования контактов. Вы научитесь динамически переключать фрагменты в графическом интерфейсе; кроме того, в этом приложении снова будут определены раздельные макеты, оптимизирующие использование экранного пространства на телефонах и планшетах.

9 Приложение Address Book

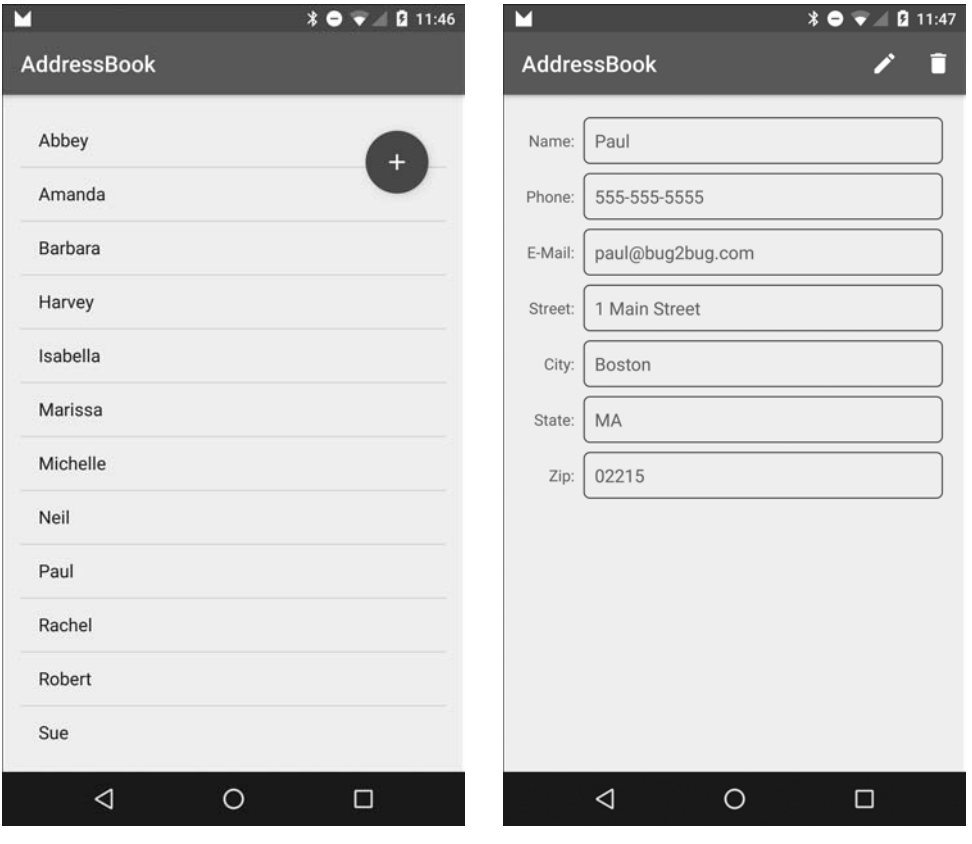
FragmentTransaction и стек возврата, SQLite, SQLiteDatabase, SQLiteOpenHelper, ContentProvider, ContentResolver, Loader, LoaderManager, Cursor и стили GUI

В этой главе…

- Использование транзакций и стека возврата для динамического присоединения и отсоединения фрагментов
- Использование RecyclerView для вывода информации из базы данных
- Создание и открытие баз данных SQLite с помощью класса SQLiteOpenHelper
- Использование классов ContentProvider и SQLiteDatabase для работы с информацией в базе данных SQLite
- Использование класса ContentResolver для вызова методов ContentProvider при выполнении операций с базой данных
- Использование классов LoaderManager и Loader для асинхронных обращений к базе данных за пределами потока графического интерфейса
- Работа с результатами запроса базы данных с использованием класса Cursor
- Определение стилей с часто используемыми значениями и атрибутами GUI, применяемыми к разным компонентам графического интерфейса

9.1. Введение

Приложение `Address Book` (рис. 9.1) обеспечивает удобный доступ к информации контактов, хранящейся в базе данных SQLite на устройстве. Пользователь прокручивает список контактов, отсортированный в алфавитном порядке. Чтобы просмотреть подробные сведения о контакте, пользователь касается его имени. Также приложение позволяет добавлять новые контакты, редактировать и изменять уже существующие контакты.



а *б*

Рис. 9.1. Список контактов и подробная информация о выбранном контакте: *а* — список контактов; *б* — подробная информация о выделенном контакте

Приложение предоставляет отдельный макет для планшетных устройств (рис. 9.2), в котором список контактов всегда занимает треть экрана, а остальные две трети используются либо для вывода информации выбранного контакта, либо для добавления/редактирования контакта.

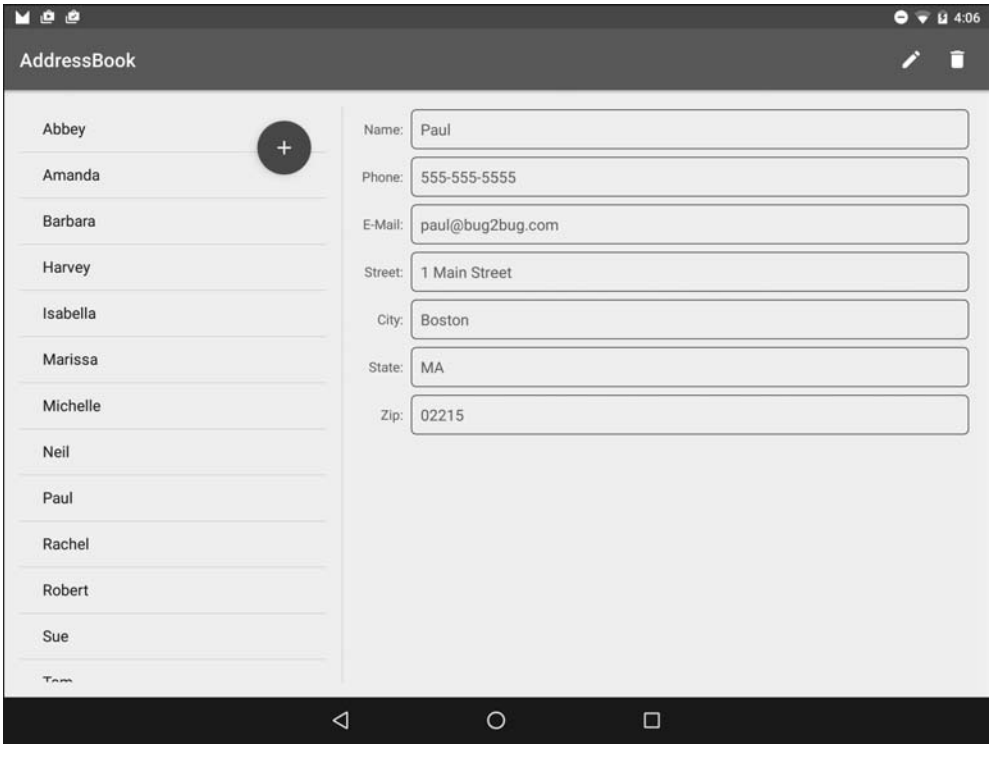


Рис. 9.2. Приложение Address Book на планшете в альбомной ориентации: значки на панели приложения отображаются вместе с текстом

Мы начнем с тестирования приложения. Далее будет приведен обзор технологий, использованных при его создании. После этого будет создан графический интерфейс приложения и файлы ресурсов. Глава завершается подробным анализом исходного кода приложения, в котором особое внимание уделяется новым возможностям.

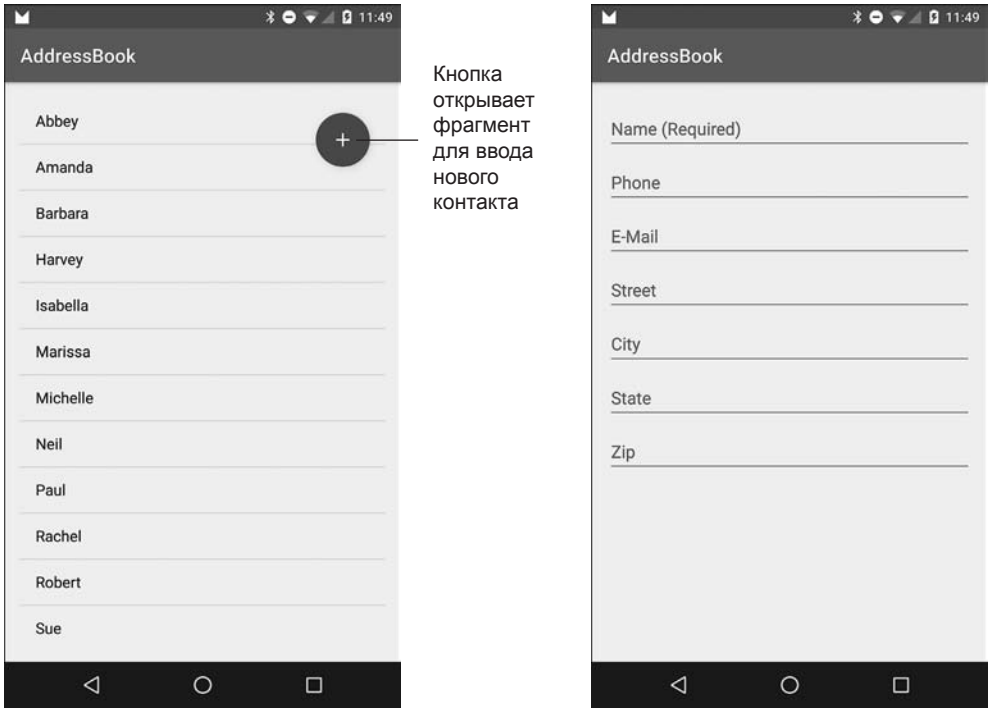
9.2. Тестирование приложения Address Book

Открытие и выполнение приложения

Запустите Android Studio и откройте приложение `Address Book` из папки `AddressBook` в примерах книги. Запустите приложение в AVD или на устройстве. Среда разработки строит проект и запускает приложение.

9.2.1. Добавление контакта

После первого запуска приложения список контактов пуст. Коснитесь кнопки `FloatingActionButton` +; открывается экран для добавления новой записи в список контактов (рис. 9.3). Приложение требует, чтобы каждому контакту было присвоено имя, поэтому кнопка сохранения (💾) появляется только в том случае, если поле `Name` содержит текст. После ввода информации о контакте коснитесь кнопки 💾 для сохранения контакта в базе данных и возврата на главный экран приложения. Если вы передумали добавлять контакт в базу данных, коснитесь кнопки `Back` устройства для возврата к главному экрану. При желании добавьте дополнительные контакты. На планшетах после добавления новых контактов справа от списка будет отображаться подробная информация (рис. 9.3). Обратите внимание: на планшетах список контактов отображается всегда.



а *б*

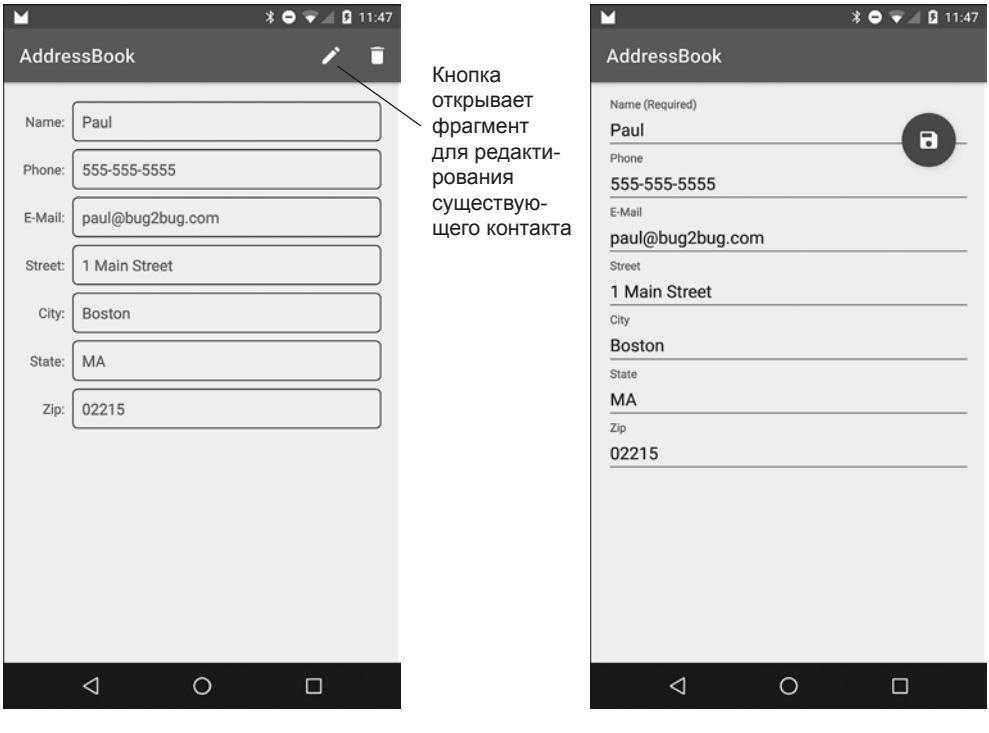
Рис. 9.3. Добавление контакта в базу данных: *а* — кнопка для создания нового контакта; *б* — фрагмент для добавления контакта

9.2.2. Просмотр контакта

Чтобы просмотреть дополнительные сведения о контакте на телефоне или AVD телефона, коснитесь его имени (как показано на рис. 9.1). Как упоминалось выше, на планшетах подробная информация автоматически выводится справа от списка (см. рис. 9.3).

9.2.3. Изменение контакта

Во время просмотра сведений о контакте коснитесь кнопки на панели действий. Появится экран с компонентами `EditText`, уже заполненными данными контакта (рис. 9.4). Внесите необходимые изменения и коснитесь кнопки `FloatingActionButton` для сохранения обновленной информации в базе данных и возврата на главный экран приложения. Если вы передумали изменять контакт, просто коснитесь кнопки `Back` (), чтобы вернуться к предыдущему экрану. На планшете после редактирования подробная обновленная информация выводится справа от списка.

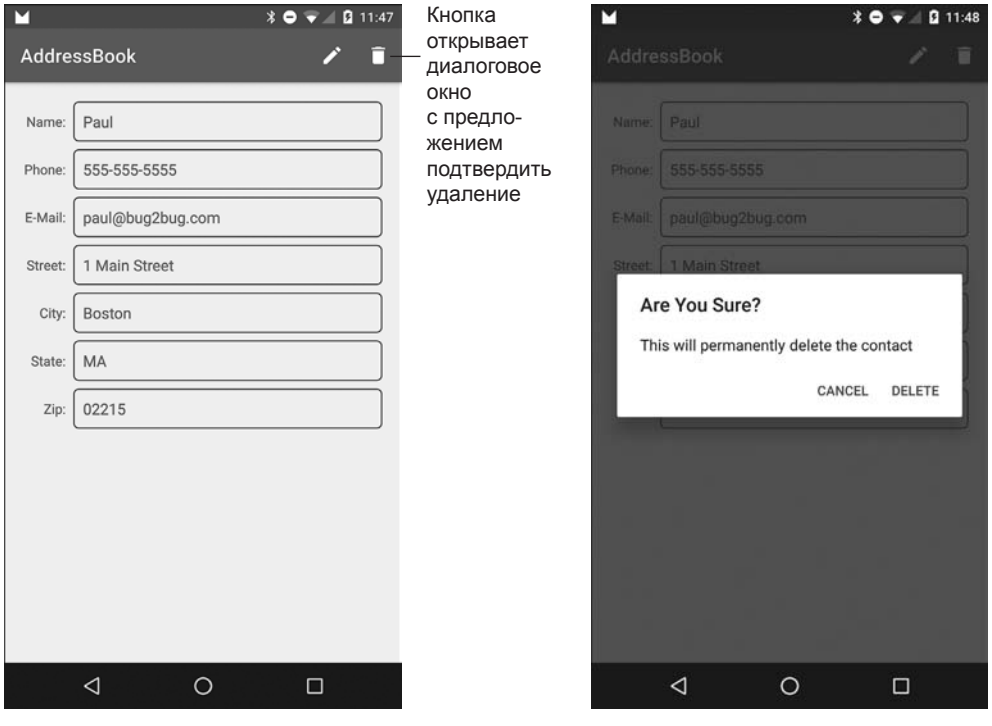


а *б*

Рис. 9.4. Редактирование данных контакта: *а* — подробная информация контакта; *б* — фрагмент для редактирования контакта

9.2.4. Удаление контакта

В процессе просмотра подробных сведений о контакте коснитесь кнопки на панели действий. Открывается диалоговое окно для подтверждения удаления. Если нажать кнопку DELETE, контакт будет удален из базы данных, а в приложении откроется обновленный список контактов. Кнопка CANCEL сохраняет контакт.



а *б*

Рис. 9.5. Удаление контакта из базы данных: *а* — подробная информация контакта; *б* — диалоговое окно для подтверждения удаления

9.3. Обзор применяемых технологий

В этом разделе вашему вниманию будут представлены новые технологии, применяемые в процессе создания приложения Address Book.

9.3.1. Отображение фрагментов с использованием FragmentTransaction

В предыдущих приложениях, в которых использовались фрагменты, мы определяли фрагменты в макетах активностей или, для `DialogFragment`, создавали их вызовом метода `show`. Приложение `Flag Quiz` показывает, как организовать размещение фрагментов в разных активностях на телефоне или отображение нескольких фрагментов в одной активности на планшете.

В этом приложении для управления всеми фрагментами приложения будет использоваться одна активность. На устройствах с размером экрана, соответствующим экрану телефона, фрагменты отображаются по одному, тогда как на планшетах всегда отображается фрагмент со списком контактов, а фрагменты просмотра, добавления и редактирования контактов отображаются по мере необходимости. Для динамического отображения фрагментов используются объекты `FragmentManager` и `FragmentTransaction`. Кроме того, мы используем *стек возврата* фрагментов Android — структуру данных, хранящую фрагменты в порядке LIFO (Last-In-First-Out) — для автоматической поддержки кнопки `Back` (◀) Android и для того, чтобы пользователи могли возвращаться к предыдущим фрагментам. За дополнительной информацией о фрагментах и `FragmentTransaction` обращайтесь по адресу

http://developer.android.com/guide/components/fragments.html

9.3.2. Передача данных между фрагментом и управляющей активностью

Обмен данными между фрагментами и управляющей активностью (или другими фрагментами активности) рекомендуется осуществлять через управляющую активность — тем самым расширяются возможности повторного использования фрагментов, которые не обращаются друг к другу напрямую. Обычно каждый фрагмент определяет интерфейс *методов обратного вызова*, реализуемый в управляющей активности. Мы используем этот прием для того, чтобы активность `MainActivity` приложения получала оповещения, когда пользователь:

- выбирает контакт;
- касается кнопки `FloatingActionBar` (+) в списке контактов;
- касается кнопок операций ✎ или 🗑;
- касается кнопки 💾 для завершении редактирования существующего или добавления нового контакта.

9.3.3. Работа с базой данных SQLite

Информация о контактах приложения хранится в базе данных SQLite. Система управления базами данных SQLite (*www.sqlite.org*) — одна из самых популярных СУБД во всем мире. Для упрощения создания базы данных используется субкласс `SQLiteOpenHelper` (пакет `android.database.sqlite`), а для работы с содержимым базы данных — объект `SQLiteDatabase` (пакет `android.database.sqlite`). Запросы к базе данных выполняются на языке SQL (Structured Query Language), а для управления результатами запроса к базе данных используется класс `Cursor` (пакет `android.database`). За дополнительной информацией об использовании SQLite в Android обращайтесь по адресу

http://developer.android.com/guide/topics/data/data-storage.html#db

9.3.4. Классы ContentProvider и ContentResolver

Класс `ContentProvider` (пакет `android.provider`) открывает доступ к данным приложения для использования в этом или других приложениях. Android предоставляет множество встроенных разновидностей `ContentProvider`. Например, ваше приложение может взаимодействовать с данными из приложений Android `Contacts` и `Calendar`. Также существуют варианты `ContentProvider` для различных телефонных функций, графики и видеоматериалов, и пользовательского словаря (используемого средствами предиктивного ввода текста в Android).

Кроме предоставления доступа к данным другим приложениям, классы `ContentProvider` также позволяют выдавать рекомендации для пользователей при проведении поиска на устройстве и используются для поддержки операций копирования/вставки между приложениями.

В этом приложении класс `ContentProvider` используется для выполнения асинхронных операций с базой данных за пределами потока GUI — это необходимо при работе с классами `Loader` и `LoaderManager` (раздел 9.3.5). Мы создадим субкласс `ContentProvider`, который определяет, как:

- выдать запрос к базе данных для получения заданного контакта или всех контактов;
- добавить новый контакт в базу данных;
- обновить информацию существующего контакта в базе данных;
- удалить существующий контакт из базы данных.

Класс `ContentProvider` использует субкласс `SQLiteOpenHelper` для создания базы данных и получения объектов `SQLiteDatabase` для выполнения упомянутых операций. При внесении изменений в базу данных `ContentProvider` оповещает

слушателей об этих изменениях, чтобы приложение могло изменить данные в графическом интерфейсе.

Uri

Класс `ContentProvider` определяет объекты `Uri`, идентифицирующие выполняемые операции. Например, в этом приложении метод `query` класса `ContentProvider` используется для выполнения двух разных операций — одна возвращает объект `Cursor` для одного контакта, а другая возвращает объект `Cursor` для имен всех контактов в базе данных.

ContentResolver

Для обращения к функциям получения информации, вставки, обновления и удаления `ContentProvider` мы используем соответствующие методы встроенного в `Activity` класса `ContentResolver` (пакет `android.content`). Классы `ContentProvider` и `ContentResolver` берут на себя технические подробности обмена данными — в том числе и между приложениями, если `ContentProvider` предоставляет свои данные другим приложения. Как вы вскоре увидите, в первом аргументе методов `ContentResolver` передается объект `Uri`, определяющий `ContentProvider`. Каждый метод `ContentResolver` вызывает соответствующий метод `ContentProvider`, который использует `Uri` для определения выполняемой операции.

За дополнительной информацией о классах `ContentProvider` и `ContentResolver` обращайтесь по адресу

http://developer.android.com/guide/topics/providers/content-providers.html

9.3.5. Loader и LoaderManager — асинхронные операции с базами данных

Как упоминалось ранее, долгие операции или операции, блокирующие выполнение приложения до своего завершения (например, обращения к файлам или базам данных), должны выполняться за пределами потока GUI. Это помогает приложению быстрее реагировать на действия пользователя и предотвращает появление диалоговых окон ANR (Activity Not Responding). Классы `Loader` и `LoaderManager` помогают организовать выполнение асинхронных операций из любой активности или фрагмента.

Loader

Класс `Loader` (пакет `android.content`) выполняет асинхронные обращения к данным. При взаимодействиях с `ContentProvider` для загрузки и обработки

данных обычно используется `CursorLoader` — субкласс `AsyncTaskLoader`, использующий `AsyncTask` для работы с данными в отдельном потоке. Класс `Loader` также позволяет:

- отслеживать изменения в соответствующем источнике данных и предоставлять доступ к обновленным данным соответствующей активности или фрагменту;
- заново связываться с объектом `Cursor` последнего экземпляра `Loader` вместо выполнения нового запроса при изменении конфигурации.

LoaderManager и LoaderManager.LoaderCallbacks

Объекты `Loader` активностей и фрагментов создаются и находятся под управлением класса `LoaderManager` (пакет `android.app`), связывающего жизненный цикл каждого объекта `Loader` с жизненным циклом его активности или фрагмента. Кроме того, `LoaderManager` вызывает методы интерфейса `LoaderManager.LoaderCallbacks` для оповещения активности или фрагмента о том, что объект `Loader`:

- должен быть создан;
- завершил загрузку своих данных; или
- произошел сброс состояния, и его данные недоступны.

Мы будем использовать `Loader` и `LoaderManager` в нескольких субклассах фрагментов этого приложения.

За дополнительной информацией о классах `Loader` и `LoaderManager` обращайтесь по адресу

http://developer.android.com/guide/components/loaders.html

9.3.6. Определение и применение стилей к компонентам GUI

Пары «атрибут—значение» для часто используемых компонентов GUI можно определить в виде ресурсов стилей XML (раздел 9.4.5). Затем эти стили применяются ко всем компонентам, которые разделяют соответствующие значения (раздел 9.4.9), для чего используется атрибут `style`. Все последующие изменения, внесенные в стиль, будут автоматически применены ко всем компонентам GUI, использующим этот стиль. Мы используем стили для оформления компонентов `TextView`, в которых отображается информация контакта. За дополнительной информацией о стилях обращайтесь по адресу

http://developer.android.com/guide/topics/ui/themes.html

9.3.7. Определение фона для компонентов TextView

По умолчанию у компонентов `TextView` отсутствует граница. Чтобы задать границу, укажите значение `Drawable` для атрибута `android:background` компонента `TextView`. В качестве значения `Drawable` может использоваться изображение, хотя для этого приложения можно определить новый тип `Drawable`, используя XML-представление фигуры (раздел 9.4.6). Файл ресурсов для такого объекта `Drawable` определяется в одной или нескольких папках `drawable` приложения. За дополнительной информацией о ресурсах `Drawable` обращайтесь по адресу

http://developer.android.com/guide/topics/resources/drawable-resource.html

9.4. Создание графического интерфейса пользователя и файлов ресурсов

В этом разделе будут созданы дополнительные файлы с исходным кодом Java, файлы ресурсов и файлы разметки GUI.

9.4.1. Создание проекта

Создайте новый проект на базе шаблона `Blank Activity`. Установите в параметрах проекта флажок `Use a Fragment`. На шаге `New Project` диалогового окна `Create New Project` укажите следующие значения:

- `Application name`: `Address Book`;
- `Company Domain`: `deitel.com` (или ваше доменное имя).

Добавьте в проект значок приложения так, как было описано ранее. Настройте поддержку Java SE 7 в соответствии с описанием в разделе 4.4.3. Выполните действия, описанные в разделе 8.4.3, чтобы добавить библиотеку `RecyclerView` в проект. В файле `colors.xml` замените значение `colorAccent` на `#FF4081`.

9.4.2. Создание классов приложения

При создании проекта Android Studio определяет классы `MainActivity` и `MainActivityFragment` за вас. В этом приложении класс `MainActivityFragment` следует переименовать в `ContactsFragment`. Это делается так:

1. Откройте класс `MainActivityFragment` в редакторе.
2. Щелкните правой кнопкой мыши на имени класса и выберите команду `Refactor` ▸ `Rename...`. IDE выделяет имя класса для редактирования.

3. Введите имя `ContactsFragment` и нажмите Enter. IDE изменяет имя класса и его конструктора, а также изменяет имя файла класса.

Пакет com.deitel.addressbook

Приложение состоит из семи дополнительных классов, которые необходимо добавить в проект (`File` ▸ `New` ▸ `Java Class`). Дополнительные классы в пакете `com.deitel.addressbook`:

- `ContactsAdapter` — субкласс `RecyclerView.Adapter`, поставляющий данные компоненту `RecyclerView` класса `ContactsFragment`;
- `AddEditFragment` — субкласс `Fragment`, предоставляющий графический интерфейс для добавления нового или редактирования существующего фрагмента;
- `DetailFragment` — субкласс `Fragment`, отображающий информацию одного контакта и предоставляющий команды меню для редактирования и удаления этого контакта;
- `ItemDivider` — субкласс `RecyclerView.ItemDecoration`, используемый компонентом `RecyclerView` класса `ContactsFragment` для рисования горизонтальной линии между элементами.

Этот класс идентичен классу из раздела 8.7, поэтому вы можете просто скопировать этот класс из проекта Twitter Searches и вставить его в узел `app>java>com.deitel.addressbook` в окне `Project`.

Пакет com.deitel.addressbook.data

Этот класс также определяет вложенный пакет с именем `com.deitel.addressbook.data`, содержащий классы для работы с базой данных приложения. Чтобы создать пакет, выполните следующие действия.

1. В окне `Project` щелкните правой кнопкой мыши на пакете `com.deitel.addressbook` и выберите команду `New` ▸ `Package`.
2. Введите имя `data`, чтобы создать пакет с именем `com.deitel.addressbook.data`.

Затем добавьте следующие классы в пакет `com.deitel.addressbook.data`.

- Класс `DatabaseDescription` содержит описание таблицы `contacts` базы данных.
- Класс `AddressBookDatabaseHelper` (субкласс `SQLiteOpenHelper`) создает базу данных и используется для работы с ней.
- Класс `AddressBookContentProvider` (субкласс `ContentProvider`) определяет, как приложение будет работать с базой данных. Класс создается командой `New` ▸ `Other` ▸ `Content Provider`. В поле `URI authorities` укажите имя `com.deitel.addressbook.data` и снимите флажок `Exported`, затем нажмите кнопку `Finish`.

Снятие флажка `Exported` означает, что класс `ContentProvider` предназначен для использования только в этом приложении. IDE определяет субкласс `ContentProvider` и переопределяет его обязательные методы. Кроме того, IDE объявляет `ContentProvider` в `AndroidManifest.xml` — в элементе `<provider>`, вложенном в элемент `<application>`. Это необходимо для регистрации `ContentProvider` в операционной системе Android — не только в данном приложении, но и в других (при экспортировании `ContentProvider`).

Все эти классы кратко описаны в разделе 9.5, а подробные описания приводятся в разделах 9.6–9.13.

9.4.3. Добавление значков

Добавьте в проект значки сохранения (), добавления (+), редактирования () и удаления () из `Vector Asset Studio` (как это было сделано в разделе 4.4.9) — они будут использоваться как значки `FloatingActionButton`. После добавления векторных значков перейдите в папку `res/drawable`, откройте файл XML каждого значка и замените значение атрибута `android:fillColor` элемента `<path>` на

```
"@android:color/white"
```

Так значок будет лучше виден на фоне акцентного цвета приложения, который назначается `FloatingActionButton` темой приложения.

9.4.4. strings.xml

В табл. 9.1 перечислены имена строковых ресурсов приложения и соответствующие значения. Сделайте двойной щелчок на файле `strings.xml` в папке `res/values`, чтобы открыть редактор для создания строковых ресурсов.

Таблица 9.1. Строковые ресурсы, используемые в приложении Address Book

Имя ресурса	Значение
menuitem_edit	Edit
menuitem_delete	Delete
hint_name_required	Name (Required)
hint_email	E-Mail
hint_phone	Phone
hint_street	Street

Имя ресурса	Значение
hint_city	City
hint_state	State
hint_zip	Zip
label_name	Name:
label_email	E-Mail:
label_phone	Phone:
label_street	Street:
label_city	City:
label_state	State:
label_zip	Zip:
confirm_title	Are You Sure?
confirm_message	This will permanently delete the contact
button_cancel	Cancel
button_delete	Delete
contact_added	Contact added successfully
contact_not_added	Contact was not added due to an error
contact_updated	Contact updated
contact_not_updated	Contact was not updated due to an error
invalid_query_uri	Invalid query Uri:
invalid_insert_uri	Invalid insert Uri:
invalid_update_uri	Invalid update Uri:
invalid_delete_uri	Invalid delete Uri:
insert_failed	Insert failed: s

9.4.5. styles.xml

В этом разделе определяются стили компонентов `TextView` фрагмента `DetailFragment`, используемые для вывода информации контакта (раздел 9.4.9). Как и все остальные ресурсы, ресурсы стилей хранятся в папке `res/values`. При создании проекта IDE создает файл `styles.xml` с предопределенными стилями. Для

каждого созданного стиля указывается имя, используемое для его применения к компонентам GUI, и один или несколько элементов, определяющих применяемые значения свойств. Чтобы создать новые стили, откройте файл `styles.xml` в папке `res/values` и добавьте разметку из листинга 9.1 перед закрывающим тегом `</resources>`. Сохраните и закройте файл `styles.xml`.

Листинг 9.1. Новые стили для форматирования компонентов TextView фрагмента DetailFragment

```
1    <style name="ContactLabelTextView">
2       <item name="android:layout_width">wrap_content</item>
3       <item name="android:layout_height">wrap_content</item>
4       <item name="android:layout_gravity">right|center_vertical</item>
5    </style>
6
7    <style name="ContactTextView">
8       <item name="android:layout_width">wrap_content</item>
9       <item name="android:layout_height">wrap_content</item>
10      <item name="android:layout_gravity">fill_horizontal</item>
11      <item name="android:textSize">16sp</item>
12      <item name="android:background">@drawable/textview_border</item>
13   </style>
```

В строках 1–5 определяется новый стиль с именем `ContactLabelTextView`, определяющий значения свойств макета `layout_width`, `layout_height` и `layout_gravity`. Этот стиль будет назначен компонентам `TextView` фрагмента `DetailFragment`, расположенным слева от каждого поля в информации контакта. Каждый новый стиль состоит из элемента `style`, содержащего элементы `item`. Атрибут `name` элемента `style` используется для применения стиля. Атрибут `name` элемента `item` определяется свойство, а его значение задается этому свойству при применении стиля. В строках 7–13 определяется стиль с именем `ContactTextView`, который будет применен к компонентам `TextView` фрагмента `DetailFragment`. В строке 12 свойству `android:background` задается ресурс `drawable`, определяемый в разделе 9.4.6.

9.4.6. Файл textview_border.xml

Стиль `ContactTextView`, созданный в предыдущем разделе, определяет внешний вид компонентов `TextView`, используемых для отображения подробной информации о контактах. Мы задали объект `Drawable` (графическое изображение) с именем `@drawable/textview_border` как значение атрибута `android:background` компонента `TextView`. В этом разделе мы определим этот объект `Drawable` в папке `res/drawable`. Чтобы определить объект `Drawable`, выполните следующие действия.

1. Щелкните правой кнопкой мыши на папке `res/drawable-mdpi` и выберите команду `New` ▸ `Drawable resource file`.

2. Введите в поле File name значение textview_border.xml и нажмите OK.
3. Замените содержимое файла разметкой XML из листинга 9.2.

Листинг 9.2. XML-представление объекта Drawable, используемого для создания рамок компонентов TextView

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <shape xmlns:android="http://schemas.android.com/apk/res/android"
3    android:shape="rectangle">
4    <corners android:radius="5dp"/>
5    <stroke android:width="1dp" android:color="#555"/>
6    <padding android:top="10dp" android:left="10dp" android:bottom="10dp"
7       android:right="10dp"/>
8 </shape>
```

Атрибут android:shape элемента shape (строка 3) может иметь значение "rectangle" (использованное в этом примере), "oval", "line" или "ring". Элемент corners (строка 4) определяет радиус закруглений углов прямоугольного контура. Элемент stroke (строка 5) определяет толщину и цвет контура прямоугольника. Элемент padding (строки 6–7) определяет интервалы вокруг содержимого элемента, к которому применяется объект Drawable. Отступы со всех четырех сторон должны задаваться отдельно. Полную информацию об определении фигур можно просмотреть по адресу

http://developer.android.com/guide/topics/resources/drawable-resource.html#Shape

9.4.7. Макет MainActivity

По умолчанию макет MainActivity содержит кнопку FloatingActionButton и включает файл макета content_main.xml. В нашем приложении кнопки FloatingActionButton будут предоставляться фрагментами приложения. Откройте файл activity_main.xml из папки res/layout и удалите предопределенную кнопку FloatingActionButton. Также задайте свойству id компонента CoordinatorLayout значение coordinatorLayout — оно будет использоваться при отображении уведомлений SnackBar. Удалите код настройки FloatingActionButton из метода onCreate класса MainActivity.

Макет для телефона: activity_main.xml в res/layout

Для MainActivity будут определены два макета content_main.xml — для устройств с размером экрана, соответствующим размеру экрана телефона, и для планшетных устройств. Откройте файл content_main.xml в папке res/layout и замените его содержимое разметкой XML из листинга 9.2. MainActivity динамически отображает фрагменты приложения в компоненте FrameLayout с именем fragmentContainer. Этот компонент занимает все свободное место в макете

MainActivity с отступами 16dp с каждой из сторон. Свойство app:layout_behavior (строка 20) используется компонентом CoordinatorLayout из activity_main.xml для управления взаимодействиями между представлениями. Задание свойства гарантирует, что содержимое FrameLayout будет располагаться под объектом Toolbar, определенным в activity_main.xml.

Листинг 9.3. Файл content_main.xml для телефона

```
 9 <FrameLayout
10    android:id="@+id/fragmentContainer"
11    xmlns:android="http://schemas.android.com/apk/res/android"
12    xmlns:app="http://schemas.android.com/apk/res-auto"
13    xmlns:tools="http://schemas.android.com/tools"
14    android:layout_width="match_parent"
15    android:layout_height="match_parent"
16    android:paddingBottom="@dimen/activity_vertical_margin"
17    android:paddingLeft="@dimen/activity_horizontal_margin"
18    android:paddingRight="@dimen/activity_horizontal_margin"
19    android:paddingTop="@dimen/activity_vertical_margin"
20    app:layout_behavior="@string/appbar_scrolling_view_behavior"
21    tools:context=".MainActivity"/>
```

Макет для планшета: content_main.xml для больших устройств

Создайте файл content_main.xml (см. раздел 4.5.4). Этот макет будет состоять из горизонтального компонента LinearLayout, содержащего фрагмент ContactsFragment и пустой компонент FrameLayout. (листинг 9.4). Создайте ресурс divider_margin (16dp), используемый в строках 24 и 32. Класс LinearLayout содержит несколько свойств, которые ранее не упоминались.

- divider (строка 9) — свойство задает ресурс Drawable, используемый для разделения элементов LinearLayout. В данном случае мы используем предопределенный ресурс темы Android ?android:listDivider. Запись ?android: обозначает, что компонент LinearLayout должен использовать разделитель, заданный в текущей теме.
- showDividers (строка 15) — свойство используется в сочетании с divider для определения позиции разделителей — в данном случае значение middle указывает, что разделители должны выводиться только между элементами LinearLayout. Также можно вывести разделитель перед первым элементом в макете (beginning) и после последнего элемента (end); эти значения могут объединяться оператором |.
- weightSum (строка 16) — свойство обеспечивает распределение горизонтального пространства между ContactsFragment и FrameLayout. Задавая свойству weightSum компонента LinearLayout значение 3, а свойствам layout_weight компонентов ContactsFragment и FrameLayout — значения 1 и 2 соответственно,

мы указываем, что компонент `ContactsFragment` должен занимать 1/3 ширины `LinearLayout`, а компонент `FrameLayout` — оставшиеся 2/3.

Листинг 9.4. Файл content_main.xml для планшета

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:baselineAligned="false"
   android:divider="?android:listDivider"
   android:orientation="horizontal"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:showDividers="middle"
   android:weightSum="3"
   app:layout_behavior="@string/appbar_scrolling_view_behavior">

   <fragment
      android:id="@+id/contactsFragment"
      android:name="com.deitel.addressbook.ContactsFragment"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_marginEnd="@dimen/divider_margin"
      android:layout_weight="1"
      tools:layout="@layout/fragment_contacts"/>

   <FrameLayout
      android:id="@+id/rightPaneContainer"
      android:layout_width="0dp"
      android:layout_height="match_parent"
      android:layout_marginStart="@dimen/divider_margin"
      android:layout_weight="2"/>
</LinearLayout>
```

9.4.8. Макет ContactsFragment

Помимо переименования класса `MainActivityFragment` в `ContactsFragment`, соответствующий файл макета также был переименован в `fragment_contacts.xml`. Затем мы удалили сгенерированный компонент `TextView`, заменили компонент по умолчанию `RelativeLayout` компонентом `FrameLayout` и удалили свойства `padding` макета. После этого был добавлен компонент `RecyclerView` с именем `recyclerView` и кнопка `FloatingActionButton` с именем `addButton`. Итоговая разметка XML макета приведена в листинге 9.5. Убедитесь в том, что вы задали свойства `RecyclerView` и `FloatingActionButton` так, как показано в листинге.

Листинг 9.5. Макет fragment_contacts.xml

```
<FrameLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <android.support.v7.widget.RecyclerView
      android:id="@+id/recyclerView"
      android:layout_width="match_parent"
      android:layout_height="match_parent"/>

   <android.support.design.widget.FloatingActionButton
      android:id="@+id/addButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="top|end"
      android:layout_margin="@dimen/fab_margin"
      android:src="@drawable/ic_add_24dp"/>
</FrameLayout>
```

9.4.9. Макет DetailFragment

Когда пользователь касается контакта в `MainActivity`, приложение отображает фрагмент `DetailFragment` (рис. 9.6). Макет этого фрагмента (`fragment_details.xml`)

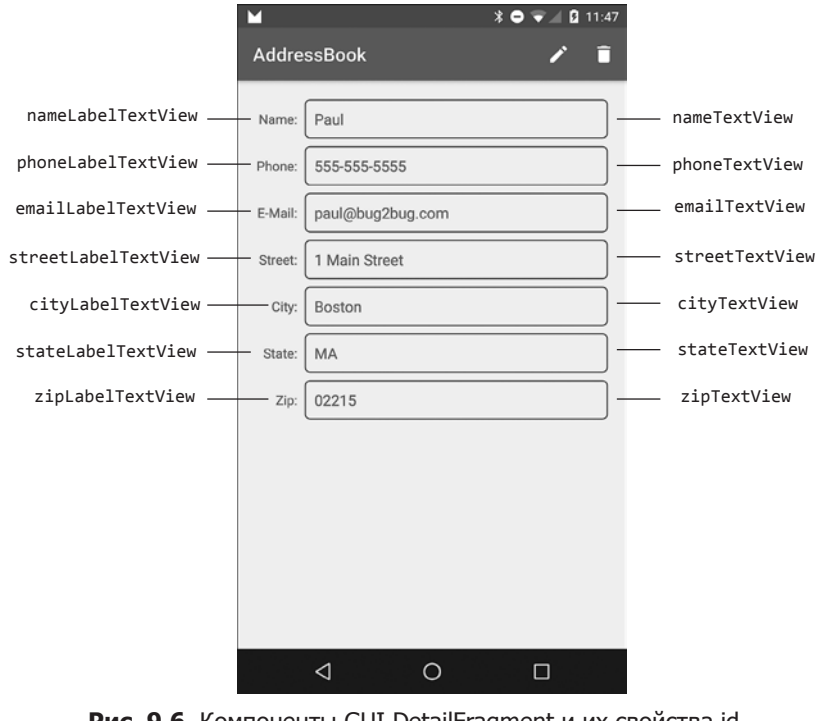


Рис. 9.6. Компоненты GUI DetailFragment и их свойства id

состоит из компонента `ScrollView` с вертикальной панелью `GridLayout`, состоящей из двух столбцов `TextView`. Компонент `ScrollView` может содержать другие представления с возможностью прокрутки информации, не помещающейся на экране. Мы используем `ScrollView`, чтобы пользователь мог прокрутить информацию о контакте, если на устройстве не хватает места для отображения всех компонентов `TextView` (см. рис. 9.6). Создайте для этого фрагмента новый файл ресурса макета `fragment_details.xml` и выберите корневым элементом (`Root Element`) компонент `ScrollView`. После создания файла добавьте в `ScrollView` компонент `GridLayout`.

Настройка компонента GridLayout

Для компонента `GridLayout` задайте свойству `layout:width` значение `match_parent`, свойству `layout:height` — значение `wrap_content`, свойству `columnCount` — значение 2 и свойству `useDefaultMargins` — значение `true`. Значение `layout:height` позволяет родительскому компоненту `ScrollView` определить фактическую высоту `GridLayout` и решить, нужно ли предоставить средства прокрутки. Добавьте компоненты `TextView` на панель `GridLayout`, как показано на рис. 9.6.

Настройки компонентов TextView в левом столбце

Задайте свойство `id` каждого компонента `TextView` в левом столбце в соответствии с рис. 9.6. Кроме того, задайте следующие свойства:

- `layout:row` — значение от 0 до 6 (в зависимости от строки);
- `layout:column` — значение 0;
- `text` — соответствующий строковый ресурс из файла `strings.xml`;
- `style` — значение `@style/ContactLabelTextView` (для определения стилевых ресурсов используется синтаксис `@style/`*`имяСтиля`*).

Настройки компонентов TextView в правом столбце

Задайте свойство `id` каждого компонента `TextView` в правом столбце в соответствии с рис. 9.6. Кроме того, задайте следующие свойства:

- `layout:row` — значение от 0 до 6 (в зависимости от строки);
- `layout:column` — значение 1;
- `style` — значение `@style/ContactTextView`.

9.4.10. Макет AddEditFragment

Когда пользователь касается кнопки + в `ContactsFragment` или кнопки ✎ на панели приложения в `DetailFragment`, активность `MainActivity` отображает фрагмент `AddEditFragment` (рис. 9.7) с макетом `fragment_add_edit.xml`,

содержащим корневой компонент `FrameLayout` с компонентом `ScrollView` и кнопкой `FloatingActionButton`. Компонент `ScrollView` содержит вертикальный компонент `LinearLayout` с семью компонентами `TextInputLayout`.

Настройка ScrollView

Задайте свойствам `layout:width` и `layout:height` компонента `ScrollView` значение `match_parent`.

Настройка LinearLayout

У компонента `LinearLayout` свойству `layout:width` задается значение `match_parent`, свойству `layout:height` — значение `wrap_content` и свойству `orientation` — значение `vertical`. Затем в него добавляются семь компонентов `TextInputLayout`; свойства `id` этих компонентов изображены на рис. 9.7, свойству `layout:width` задано значение `match_parent`, а свойству `layout:height` — значение `wrap_content`.

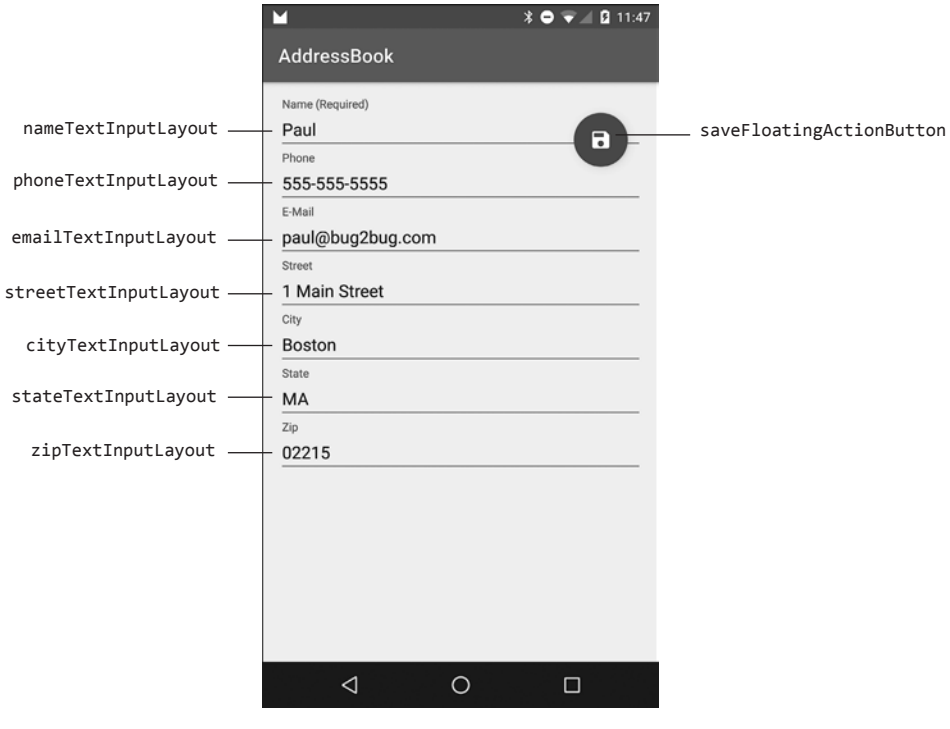


Рис. 9.7. Компоненты GUI AddEditFragment со значениями свойств id. Корневым компонентом GUI является компонент ScrollView, содержащий вертикальную панель GridLayout

Настройка EditText

Мы поместили по компоненту `EditText` в каждый компонент `TextInputLayout` и задали свойствам `hint` этих компонентов соответствующие строковые ресурсы из `strings.xml`. Также были заданы свойства `inputType` и `imeOptions` каждого компонента `EditText`. Для устройств, отображающих экранную клавиатуру, свойство `inputType` определяет тип клавиатуры, которая должна отображаться для соответствующего компонента `EditText`. Это позволяет настроить клавиатуру для конкретного типа данных, которые должны вводиться в этом компоненте `EditText`. Чтобы на экранных клавиатурах полей `nameTextInputLayout`, `phoneTextInputLayout`, `emailTextInputLayout`, `streetTextInputLayout`, `cityTextInputLayout` и `stateTextInputLayout` отображалась кнопка , мы задали свойству `imeOptions` значение `actionNext`. Когда один из этих компонентов получает фокус, кнопка передает фокус следующему компоненту `EditText` в макете. Если фокус принадлежит `zipTextInputLayout`, экранную клавиатуру можно скрыть кнопкой — для этого компонента `EditText` задайте свойству `imeOptions` значение `actionDone`.

Задайте значения свойств `inputType` компонентов `EditText` так, чтобы для них отображались соответствующие клавиатуры:

- `EditText` в `nameTextInputLayout` : `textPersonName` и `textCapWords` — для ввода имен, каждое слово начинается с большой буквы;
- `EditText` в `phoneTextInputLayout`: `phone` — для ввода телефонных номеров;
- `EditText` в `emailTextInputLayout`: `textEmailAddress` — для ввода адресов электронной почты;
- `EditText` в `streetTextInputLayout`: `textPostalAddress` и `textCapWords` — для ввода адреса, каждое слово начинается с большой буквы;
- `EditText` в `cityTextInputLayout`: `textPostalAddress` и `textCapWords`;
- `EditText` в `stateTextInputLayout`: `textPostalAddress` и `textCapCharacters` — названия штатов записываются большими буквами;
- `EditText` в `zipTextInputLayout`: `number` — для ввода чисел.

9.4.11. Меню DetailFragment

При создании проекта среда разработки сгенерировала ресурс меню `menu_main.xml`. Активности `MainActivity` в этом приложении меню не нужно, поэтому мы можем удалить методы `onCreateOptionsMenu` и `onOptionsItemSelected` класса `MainActivity` и переименовать этот ресурс меню для использования во фрагменте `DetailFragment`, отображающем команды меню на панели приложения

для изменения и удаления существующих контактов. Переименуйте файл `menu_main.xml` в `fragment_details_menu.xml`, после чего замените команду меню `Settings` содержимым листинга 9.6. Свойству `android:icon` каждой команды меню назначается ресурс `Drawable`, добавленный в разделе 9.4.3.

Листинг 9.6. Файл ресурса меню fragment_details_menu.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

   <item
      android:id="@+id/action_edit"
      android:icon="@drawable/ic_mode_edit_24dp"
      android:orderInCategory="1"
      android:title="@string/menuitem_edit"
      app:showAsAction="always"/>

   <item
      android:id="@+id/action_delete"
      android:icon="@drawable/ic_delete_24dp"
      android:orderInCategory="2"
      android:title="@string/menuitem_delete"
      app:showAsAction="always"/>
</menu>
```

9.5. Обзор классов этой главы

Это приложение состоит из девяти классов, объединенных в два пакета. Из-за большого объема кода мы приводим краткую сводку классов и их предназначения.

Пакет com.deitel.addressbook.data

Пакет содержит три класса, относящихся к работе с базой данных SQLite.

- `DatabaseDescription` (раздел 9.6) — класс содержит открытые статические поля, используемые классами `ContentProvider` и `ContentResolver`. Вложенный класс `Contact` определяет статические поля для имени таблицы базы данных, `Uri` для обращения к таблице через `ContentProvider`, имен столбцов таблицы, а также содержит статический метод для создания объекта `Uri`, ссылающегося на конкретный контакт в базе данных.
- `AddressBookDatabaseHelper` (раздел 9.7) — субкласс `SQLiteOpenHelper`, который создает базу данных и дает возможность `AddressBookContentProvider` обращаться к ней.
- `AddressBookContentProvider` (раздел 9.8) — субкласс `ContentProvider`, определяющий операции получения данных, вставки, обновления и удаления с базой данных.

Пакет com.deitel.addressbook

Пакет содержит классы, определяющие главную активность, фрагменты и адаптер приложения, используемые для отображения информации из базы данных в `RecyclerView`.

- `MainActivity` (раздел 9.9) — класс управляет фрагментами приложения и реализует их методы интерфейса обратного вызова для реакции на выбор контакта, добавление нового, обновление или удаление существующего контакта.
- `ContactsFragment` (раздел 9.10) — класс управляет списком `RecyclerView` и кнопкой `FloatingActionButton` для добавления контактов. На телефоне это первый фрагмент, отображаемый `MainActivity`. На планшете `MainActivity` всегда отображает этот фрагмент. Вложенный интерфейс `ContactsFragment` определяет методы обратного вызова, реализуемые `MainActivity`, чтобы активность могла реагировать на выбор или добавление контакта.
- `ContactsAdapter` (раздел 9.11) — субкласс `RecyclerView.Adapter`, используемый компонентом `RecyclerView` фрагмента `ContactsFragment` для связывания отсортированного списка имен контактов с `RecyclerView`. Класс `RecyclerView.Adapter` был представлен в разделах 8.3.4 и 8.6, поэтому в этом классе рассматриваются только операции, относящиеся к работе с базой данных.
- `AddEditFragment` (раздел 9.12) — класс управляет компонентами `TextInputLayout` и кнопкой `FloatingActionButton` для добавления нового или редактирования существующего контакта. Вложенный интерфейс `AddEditFragment` определяет метод обратного вызова, реализуемый `MainActivity`, чтобы активность могла реагировать на сохранение нового или обновленного контакта.
- `DetailFragment` (раздел 9.13) — класс управляет компонентами `TextView`, в которых отображается информация о выбранном контакте, и командами на панели приложения, которые позволяют пользователю отредактировать или удалить текущий контакт. Вложенный интерфейс `DetailFragment` определяет методы обратного вызова, реализуемые `MainActivity`, чтобы активность могла реагировать на удаление контакта или прикосновение к команде на панели приложения для редактирования контакта.
- `ItemDivider` — класс определяет разделитель, отображаемый между элементами компонента `RecyclerView` фрагмента `ContactsFragment`. Класс в этой главе не описывается, потому что он не отличается от аналогичного класса из раздела 8.7.

9.6. Класс DatabaseDescription

Класс `DatabaseDescription` содержит статические поля, используемые классами `ContentProvider` и `ContentResolver`, а также вложенный класс `Contact` с описанием единственной таблицы базы данных и ее столбцов.

9.6.1. Статические поля

Класс `DatabaseDescription` определяет два статических поля (листинг 9.7; строки 12–17), которые совместно определяют *авторитетное имя* (authority), передаваемое `ContentResolver` для получения `ContentProvider`. Обычно авторитетное имя соответствует имени пакета субкласса `ContentProvider`. Каждый идентификатор URI, используемый для обращения к конкретному объекту `ContentProvider`, начинается с префикса `"content://"`, за которым следует авторитетное имя — базовый идентификатор URI объекта `ContentProvider`. В строке 17 для получения базового идентификатора URI используется метод `parse` класса `Uri`.

Листинг 9.7. Объявление класса DatabaseDescription и статические поля

```
1 // DatabaseDescription.java
2 // Класс описывает имя таблицы и имена столбцов базы данных, а также
3 // содержит другую информацию, необходимую для ContentProvider
4 package com.deitel.addressbook.data;
5
6 import android.content.ContentUris;
7 import android.net.Uri;
8 import android.provider.BaseColumns;
9
10 public class DatabaseDescription {
11    // Имя ContentProvider: обычно совпадает с именем пакета
12    public static final String AUTHORITY =
13       "com.deitel.addressbook.data";
14
15    // Базовый URI для взаимодействия с ContentProvider
16    private static final Uri BASE_CONTENT_URI =
17       Uri.parse("content://" + AUTHORITY);
18
```

9.6.2. Вложенный класс Contact

Вложенный класс `Contact` (листинг 9.8) определяет имя таблицы базы данных (строка 21), объект `Uri` для обращения к таблице через `ContentProvider` (строки 24–25) и имена столбцов таблицы (строки 28–34). Имя таблицы и имена столбцов будут использоваться классом `AddressBookDatabaseHelper` (раздел 9.7) при создании базы данных. Метод `buildContactUri` создает объект `Uri` для конкретного контакта в таблице базы данных (строки 37–39). Класс `ContentUris` (пакет `android.content`) содержит статические вспомогательные методы для выполнения операций с URI `"content://"`. Метод с `withAppendedId` присоединяет косую черту (`/`) и идентификатор записи к объекту `Uri` в первом аргументе. Для каждой таблицы базы данных обычно создается класс, сходный с классом `Contact`.

Листинг 9.8. Вложенный класс Contact

```
19    // Вложенный класс, определяющий содержимое таблицы contacts
20    public static final class Contact implements BaseColumns {
21       public static final String TABLE_NAME = "contacts"; // Имя таблицы
22
23       // Объект Uri для таблицы contacts
24       public static final Uri CONTENT_URI =
25          BASE_CONTENT_URI.buildUpon().appendPath(TABLE_NAME).build();
26
27       // Имена столбцов таблицы
28       public static final String COLUMN_NAME = "name";
29       public static final String COLUMN_PHONE = "phone";
30       public static final String COLUMN_EMAIL = "email";
31       public static final String COLUMN_STREET = "street";
32       public static final String COLUMN_CITY = "city";
33       public static final String COLUMN_STATE = "state";
34       public static final String COLUMN_ZIP = "zip";
35
36       // Создание Uri для конкретного контакта
37       public static Uri buildContactUri(long id) {
38          return ContentUris.withAppendedId(CONTENT_URI, id);
39       }
40    }
41 }
```

В таблице базы данных каждой строке обычно присваивается первичный ключ, однозначно идентифицирующий строку. При работе с `ListView` и `Cursor` этому столбцу должно быть присвоено имя `"_id"` — Android также использует его для столбца `ID` в таблицах баз данных SQLite. Для `RecyclerView` это имя не является обязательным, но мы используем его из-за определенного сходства между `ListView` и `RecyclerView` и еще потому, что мы используем `Cursor` и базу данных SQLite. Вместо того чтобы определять эту константу прямо в классе `Contact`, мы реализуем интерфейс `BaseColumns` (пакет `android.provider`; строка 20), определяющий константу `_ID` со значением `"_id"`.

9.7. Класс AddressBookDatabaseHelper

Класс `AddressBookDatabaseHelper` (листинг 9.9) расширяет абстрактный класс `SQLiteOpenHelper`, упрощающий создание баз данных и управление изменениями их версий.

Листинг 9.9. Класс AddressBookDatabaseHelper (субкласс SQLiteOpenHelper) определяет базу данных приложения

```
1 // AddressBookDatabaseHelper.java
2 // Субкласс SQLiteOpenHelper, определяющий базу данных приложения
3 package com.deitel.addressbook.data;
```

```
 4
 5 import android.content.Context;
 6 import android.database.sqlite.SQLiteDatabase;
 7 import android.database.sqlite.SQLiteOpenHelper;
 8
 9 import com.deitel.addressbook.data.DatabaseDescription.Contact;
10
11 class AddressBookDatabaseHelper extends SQLiteOpenHelper {
12    private static final String DATABASE_NAME = "AddressBook.db";
13    private static final int DATABASE_VERSION = 1;
14
15    // Конструктор
16    public AddressBookDatabaseHelper(Context context) {
17       super(context, DATABASE_NAME, null, DATABASE_VERSION);
18    }
19
20    // Создание таблицы contacts при создании базы данных
21    @Override
22       public void onCreate(SQLiteDatabase db) {
23       // Команда SQL для создания таблицы contacts
24       final String CREATE_CONTACTS_TABLE =
25          "CREATE TABLE " + Contact.TABLE_NAME + "(" +
26          Contact._ID + " integer primary key, " +
27          Contact.COLUMN_NAME + " TEXT, " +
28          Contact.COLUMN_PHONE + " TEXT, " +
29          Contact.COLUMN_EMAIL + " TEXT, " +
30          Contact.COLUMN_STREET + " TEXT, " +
31          Contact.COLUMN_CITY + " TEXT, " +
32          Contact.COLUMN_STATE + " TEXT, " +
33          Contact.COLUMN_ZIP + " TEXT);";
34       db.execSQL(CREATE_CONTACTS_TABLE); // Создание таблицы contacts
35    }
36
37    // Обычно определяет способ обновления при изменении схемы базы данных
38    @Override
39    public void onUpgrade(SQLiteDatabase db, int oldVersion,
40       int newVersion) { }
41 }
```

Конструктор

Конструктор (строки 16–18) просто вызывает конструктор суперкласса, получающий четыре аргумента:

- объект `Context`, в котором создается или открывается база данных;
- имя базы данных — может быть равно `null`, если база данных существует только в памяти;
- объект `CursorFactory` — `null` означает, что вы хотите использовать объект `CursorFactory` по умолчанию (типично для большинства приложений);
- номер версии базы данных (начиная с 1).

Переопределенные методы

Мы должны переопределить абстрактные методы этого класса `onCreate` и `onUpgrade`. Если база данных еще не существует, метод `onCreate` класса `DatabaseOpenHelper` будет вызван для ее создания. Если передаваемая версия больше версии, хранящейся на устройстве, метод `onUpgrade` класса `DatabaseOpenHelper` будет вызываться для преобразования базы данных к новой версии (например, с добавлением новых таблиц или новых столбцов в существующие таблицы).

Метод `onCreate` (строки 22–35) задает таблицу, которая должна создаваться командой SQL `CREATE TABLE`. Команда определяется в формате `String` (строки 24–33) и строится из констант класса `Contact` (раздел 9.6.2). В данном случае таблица `contacts` содержит целочисленное поле первичного ключа (`Contact._ID`) и текстовые поля для остальных столбцов. Строка 34 использует метод `execSQL` класса `SQLiteDatabase` для выполнения команды `CREATE TABLE`.

Так как в нашем приложении обновлять базу данных не нужно, метод `onUpgrade` просто переопределяется с пустым телом. Класс `SQLiteOpenHelper` также предоставляет метод `onDowngrade`, предназначенный для понижения версии базы данных (если текущая версия больше версии, указанной при вызове конструктора класса `SQLiteOpenHelper`). Понижение версии может использоваться для возврата к предыдущей схеме базы данных с меньшим количеством столбцов в таблице или меньшим количеством таблиц в базе — например, для исправления ошибки в приложении.

9.8. Класс AddressBookContentProvider

Класс `AddressBookContentProvider` (субкласс `ContentProvider`) определяет, как должны выполняться операции `query`, `insert`, `update` и `delete` с базой данных.

ЗАЩИТА ОТ ОШИБОК 9.1

Обращения к ContentProvider могут осуществляться из разных потоков одного процесса и разных процессов, поэтому важно заметить, что ContentProvider по умолчанию не обеспечивает никакой синхронизации. Однако SQLite синхронизирует доступ к базе данных, поэтому в этом приложении не нужно предоставлять собственные механизмы синхронизации.

9.8.1. Поля AddressBookContentProvider

Класс `AddressBookContentProvider` (листинг 9.10) определяет несколько полей.

- `dbHelper` (строка 17) — ссылка на объект `AddressBookDatabaseHelper`, который создает базу данных и разрешает объекту `ContentProvider` выполнять с базой данных операции чтения и записи.
- `uriMatcher` (строки 20–21) — статическая переменная, содержащая объект класса `UriMatcher` (пакет `android.content`). `ContentProvider` использует `UriMatcher` для определения того, какие операции должны выполняться в методах `query`, `insert`, `update` и `delete`.
- `UriMatcher` возвращает целочисленные константы `ONE_CONTACT` и `CONTACTS` (строки 24–25) — `ContentProvider` использует их в командах `switch` в методах `query`, `insert`, `update` и `delete`.

Листинг 9.10. Поля AddressBookContentProvider

```
1 // AddressBookContentProvider.java
2 // Субкласс ContentProvider для работы с базой данных приложения
3 package com.deitel.addressbook.data;
4
5 import android.content.ContentProvider;
6 import android.content.ContentValues;
7 import android.content.UriMatcher;
8 import android.database.Cursor;
9 import android.database.SQLException;
10 import android.database.sqlite.SQLiteQueryBuilder;
11 import android.net.Uri;
12
13 import com.deitel.addressbook.data.DatabaseDescription.Contact;
14
15 public class AddressBookContentProvider extends ContentProvider {
16    // Используется для обращения к базе данных
17    private AddressBookDatabaseHelper dbHelper;
18
19    // UriMatcher помогает ContentProvider определить выполняемую операцию
20    private static final UriMatcher uriMatcher =
21       new UriMatcher(UriMatcher.NO_MATCH);
22
23    // Константы, используемые для определения выполняемой операции
24    private static final int ONE_CONTACT = 1; // Один контакт
25    private static final int CONTACTS = 2; // Таблица контактов
26
27    // Статический блок для настройки UriMatcher объекта ContentProvider
28    static {
29       // Uri для контакта с заданным идентификатором
30       uriMatcher.addURI(DatabaseDescription.AUTHORITY,
31          Contact.TABLE_NAME + "/#", ONE_CONTACT);
32
33       // Uri для таблицы
34       uriMatcher.addURI(DatabaseDescription.AUTHORITY,
35          Contact.TABLE_NAME, CONTACTS);
36    }
37
```

Строки 28–36 определяют статический блок, добавляющий объекты `Uri` в статический объект `UriMatcher`, — этот блок выполняется один раз, когда класс `AddressBookContentProvider` загружается в память.

Метод `addUri` класса `UriMatcher` получает три аргумента.

- Строка, представляющая авторитетное имя `ContentProvider` (`DatabaseDescription.AUTHORITY` в этом приложении).
- Строка, представляющая путь — каждый URI, используемый для обращения к `ContentProvider`, состоит из префикса `"content://"`, за которым следует авторитетное имя и путь, по которому `ContentProvider` определяет выполняемую задачу.
- Код `int`, возвращаемый `UriMatcher`, когда идентификатор URI, переданный `ContentProvider`, соответствует URI, хранящемуся в `UriMatcher`.

В строках 30–31 добавляется URI в форме

content://com.deitel.addressbook.data/contacts/#,

где # — метасимвол, обозначающий последовательность числовых символов — в данном случае уникальное значение первичного ключа для одного контакта в таблице `contacts`. Также существует метасимвол *, обозначающий любое количество символов. Когда URI соответствует этому формату, `UriMatcher` возвращает константу `ONE_CONTACT`.

В строках 34–35 добавляется URI в форме

content://com.deitel.addressbook.data/contacts,

представляющий всю таблицу `contacts`. Когда URI соответствует этому формату, `UriMatcher` возвращает константу `CONTACTS`. Во время рассмотрения остального кода класса `AddressBookContentProvider` вы увидите, как используются `UriMatcher` и константы `ONE_CONTACT` и `CONTACTS`.

9.8.2. Переопределенные методы onCreate и getType

Как вы увидите, объект `ContentResolver` используется для вызова методов `ContentProvider`. Получая запрос от `ContentResolver`, Android автоматически создает соответствующий объект `ContentProvider` — или использует существующий, если он был создан ранее. При создании объекта `ContentProvider` Android вызывает его метод `onCreate` для настройки `ContentProvider` (листинг 9.11, строки 39–44). Строка 42 создает объект `AddressBookDatabaseHelper`, при помощи которого провайдер обращается к базе данных. При первом обращении к провайдеру для записи в базу данных будет вызван метод `onCreate`

объекта `AddressBookDatabaseHelper` для создания базы данных (листинг 9.10, строки 22–35).

Листинг 9.11. Переопределенные методы onCreate и getType класса ContentProvider

```
38    // Вызывается при создании AddressBookContentProvider
39    @Override
40       public boolean onCreate() {
41       // Создание объекта AddressBookDatabaseHelper
42       dbHelper = new AddressBookDatabaseHelper(getContext());
43       return true; // Объект ContentProvider создан успешно
44    }
45
46    // Обязательный метод: здесь не используется, возвращаем null
47    @Override
48    public String getType(Uri uri) {
49       return null;
50    }
51
```

Метод `getType` (листинг 9.11, строки 47–50) — обязательный метод `ContentProvider`, который в этом приложении просто возвращает `null`. Метод обычно используется при создании и запуске интентов для URI с конкретными типами MIME. На основании типов MIME Android может выбирать активности для обработки интентов.

9.8.3. Переопределенный метод query

Переопределенный метод `query` класса `ContentProvider` (листинг 9.12) получает данные из источника данных провайдера — в данном случае базы данных. Он возвращает объект `Cursor`, используемый для работы с результатами. Метод `query` получает пять аргументов.

- `uri` — объект `Uri`, представляющий загружаемые данные.
- `projection` — массив `String`, представляющий столбцы, которые должен вернуть запрос. Если аргумент равен `null`, то в результат включаются все столбцы.
- `selection` — строка с критериями выборки: условие SQL `WHERE`, заданное без ключевого слова `WHERE`. Если этот аргумент равен `null`, то все строки будут включены в результат.
- `selectionArgs` — массив `String` с аргументами, заменяющими все заполнители аргументов (`?`) в строке `selection`.
- `sortOrder` — строка, представляющая порядок сортировки: условие SQL `ORDER BY`, заданное без ключевых слов `ORDER BY`. Если этот аргумент равен

null, то порядок сортировки определяется провайдером — таким образом, без явного указания порядка сортировки последовательность возвращаемых результатов не гарантирована.

SQLiteQueryBuilder

Строка 58 создает объект SQLiteQueryBuilder (пакет android.database.sqlite) для построения запросов SQL, передаваемых базе данных SQLite. В строке 59 метод setTables указывает, что запрос будет выбирать данные из таблицы contacts базы данных. Строковый аргумент этого метода может использоваться для выполнения операций соединений таблиц; для этого таблицы перечисляются через запятую или используется синтаксис условия SQL JOIN.

Листинг 9.12. Переопределенный метод query

```
52    // Получение информации из базы данных
53    @Override
54    public Cursor query(Uri uri, String[] projection,
55       String selection, String[] selectionArgs, String sortOrder) {
56
57       // Создание SQLiteQueryBuilder для запроса к таблице contacts
58       SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
59       queryBuilder.setTables(Contact.TABLE_NAME);
60
61       switch (uriMatcher.match(uri)) {
62          case ONE_CONTACT: // Выбрать контакт с заданным идентификатором
63             queryBuilder.appendWhere(
64                Contact._ID + "=" + uri.getLastPathSegment());
65             break;
66          case CONTACTS: // Выбрать все контакты
67             break;
68          default:
69             throw new UnsupportedOperationException(
70                getContext().getString(R.string.invalid_query_uri) + uri);
71       }
72
73       // Выполнить запрос для получения одного или всех контактов
74       Cursor cursor = queryBuilder.query(dbHelper.getReadableDatabase(),
75          projection, selection, selectionArgs, null, null, sortOrder);
76
77       // Настройка отслеживания изменений в контенте
78       cursor.setNotificationUri(getContext().getContentResolver(), uri);
79       return cursor;
80    }
81
```

Использование объекта UriMatcher для определения выполняемой операции

В этом приложении используются два запроса:

- выборка конкретного контакта из базы данных для просмотра или редактирования его данных и

- выборка всех контактов из базы данных для отображения имен в списке `RecyclerView` класса `ContactsFragment`.

В строках 61–71 метод `match` класса `UriMatcher` используется для определения выполняемой операции запроса. Этот метод возвращает одну из констант, зарегистрированных с `UriMatcher` (раздел 9.8.1). Если возвращается константа `ONE_CONTACT`, выбирается только контакт с идентификатором, заданным в `Uri`. В этом случае в строках 63–64 метод `appendWhere` класса `SQLiteQueryBuilder` используется для добавления условия `WHERE`, содержащего идентификатор контакта. Метод `getLastPathSegment` класса `Uri` возвращает последний сегмент URI — например, идентификатор контакта 5 в следующем URI:

content://com.deitel.addressbook.data/contacts/5

Если возвращается константа `CONTACTS`, то конструкция `switch` завершается, не добавляя ничего к запросу — в этом случае будут выбраны все контакты, поскольку условие `WHERE` отсутствует. Для всех URI, не соответствующих схеме, в строках 69–70 инициируется исключение `UnsupportedOperationException`, указывающее на недействительность URI.

Запрос к базе данных

В строках 74–75 метод `query` класса `SQLiteQueryBuilder` используется для выполнения запроса к базе данных и получения объекта `Cursor`, представляющего результаты. Аргументы метода похожи на аргументы, передаваемые методу `query` класса `ContentProvider`.

- `SQLiteDatabase` — база данных, к которой обращен запрос. Метод `getReadableDatabase` класса `AddressBookDatabaseHelper` возвращает объект `SQLiteDatabase`, доступный только для чтения.
- `projection` — массив `String`, представляющий столбцы, которые должен вернуть запрос. Если аргумент равен `null`, то в результат включаются все столбцы.
- `selection` — строка с критериями выборки: условие SQL `WHERE`, заданное *без* ключевого слова `WHERE`. Если этот аргумент равен `null`, то все записи будут включены в результат.
- `selectionArgs` — массив `String` с аргументами, заменяющими все заполнители аргументов (`?`) в строке `selection`.
- `groupBy` — строка, представляющая критерий группировки: условие SQL `GROUP BY`, заданное *без* ключевых слов `GROUP BY`. Если этот аргумент равен `null`, то группировка не выполняется.
- `having` — при использовании `groupBy` этот аргумент содержит строку, определяющую, какие группы должны быть включены в результат: условие SQL

HAVING, заданное *без* ключевого слова HAVING. Если этот аргумент равен null, то в результат включаются все группы, заданные аргументом groupBy.

- sortOrder — строка, представляющая порядок сортировки: условие SQL ORDER BY, заданное *без* ключевых слов ORDER BY. Если этот аргумент равен null, то порядок сортировки определяется провайдером — таким образом, без явного указания порядка сортировки последовательность возвращаемых результатов не гарантирована.

Отслеживание изменений в данных

В строке 78 вызывается метод setNotificationUri класса Cursor, который сообщает, что объект Cursor должен обновляться при изменении данных, на которые он ссылается. В первом аргументе передается объект ContentResolver, обращающийся к ContentProvider, а во втором — объект Uri, используемый для обращения. Строка 79 возвращает объект Cursor с результатами запроса.

9.8.4. Переопределенный метод insert

Переопределенный метод insert класса ContentProvider (листинг 9.13) добавляет в таблицу contacts новую запись. Метод insert получает два аргумента.

- uri — объект Uri, представляющий таблицу, в которую будут вставлены данные.
- values — объект ContentValues с парами «ключ—значение». Имена столбцов являются ключами, а данные, сохраняемые в столбцах, — значениями.

Строки 87–108 проверяют, относится ли URI к таблице contacts — в противном случае URI недействителен для операции insert, и в строках 106–107 инициируется исключение UnsupportedOperationException. Если URI подходит, строки 90–91 вставляют новый контакт в базу данных. Сначала мы используем getWritableDatabase класса AddressBookDatabaseHelper для получения объекта SQLiteDatabaseObject для изменения данных в базе.

Листинг 9.13. Переопределенный метод insert

```
82    // Вставка нового контакта в базу данных
83    @Override
84       public Uri insert(Uri uri, ContentValues values) {
85       Uri newContactUri = null;
86
87       switch (uriMatcher.match(uri)) {
88          case CONTACTS:
89             // При успехе возвращается идентификатор записи нового контакта
90             long rowId = dbHelper.getWritableDatabase().insert(
91                Contact.TABLE_NAME, null, values);
```

```
92
93              // Если контакт был вставлен, создать подходящий Uri;
94              // в противном случае выдать исключение
95              if (rowId > 0) { // SQLite row IDs start at 1
96                 newContactUri = Contact.buildContactUri(rowId);
97
98                 // Оповестить наблюдателей об изменениях в базе данных
99                 getContext().getContentResolver().notifyChange(uri, null);
100             }
101             else
102                throw new SQLException(
103                   getContext().getString(R.string.insert_failed) + uri);
104             break;
105          default:
106             throw new UnsupportedOperationException(
107                getContext().getString(R.string.invalid_insert_uri) + uri);
108       }
109
110       return newContactUri;
111    }
112
```

Метод `insert` класса `SQLiteDatabase` (строки 90–91) вставляет значения из объекта `ContentValues`, содержащегося в третьем аргументе, в таблицу, заданную первым аргументом, — в данном случае это таблица `contacts`. Второй параметр этого метода `nullColumnHack` не используется в приложении, поскольку SQLite не поддерживает вставку совершенно пустых записей — это было бы эквивалентно передаче пустого объекта `ContentValues`. Вместо того чтобы запрещать передачу методу пустого объекта `ContentValues`, параметр `nullColumnHack` определяет столбец, способный принимать значения `null`.

Метод `insert` возвращает уникальный идентификатор нового контакта, если вставка завершилась успешно, или –1 в случае неудачи. Строка 95 проверяет, что `rowID` больше 0 (в SQLite записи индексируются с 1). Если проверка дает положительный результат, то строка 96 создает `Uri` для представления нового контакта, а строка 99 оповещает `ContentResolver` об изменении базы данных, чтобы клиентский код `ContentResolver` мог отреагировать на изменения. Если `rowID` не больше 0, то попытка выполнения операции завершается неудачей и в строках 102–103 инициируется исключение `SQLException`.

9.8.5. Переопределенный метод update

Переопределенный метод `update` класса `ContentProvider` (листинг 9.14) обновляет существующую запись.

Метод `update` получает четыре аргумента.

- `uri` — объект `Uri`, представляющий таблицу, в которой обновляются данные.

- `values` — объект `ContentValues` с именами обновляемых столбцов и их значениями.
- `selection` — строка с критериями выборки: условие SQL `WHERE`, заданное *без* ключевого слова `WHERE`. Если этот аргумент равен `null`, то все записи будут включены в результат.
- `selectionArgs` — массив `String` с аргументами, заменяющими все заполнители аргументов (`?`) в строке `selection`.

Листинг 9.14. Переопределенный метод update

```
113    // Обновление существующего контакта в базе данных
114    @Override
115    public int update(Uri uri, ContentValues values,
116       String selection, String[] selectionArgs) {
117       int numberOfRowsUpdated; // 1, если обновление успешно; 0 при неудаче
118
119       switch (uriMatcher.match(uri)) {
120          case ONE_CONTACT:
121             // Получение идентификатора контакта из Uri
122             String id = uri.getLastPathSegment();
123
124             // Обновление контакта
125             numberOfRowsUpdated = dbHelper.getWritableDatabase().update(
126                Contact.TABLE_NAME, values, Contact._ID + "=" + id,
127                selectionArgs);
128             break;
129          default:
130             throw new UnsupportedOperationException(
131                getContext().getString(R.string.invalid_update_uri) + uri);
132       }
133
134       // Если были внесены изменения, оповестить наблюдателей
135       if (numberOfRowsUpdated != 0) {
136          getContext().getContentResolver().notifyChange(uri, null);
137       }
138
139       return numberOfRowsUpdated;
140    }
141
```

В этом приложении обновление производится только для одного контакта, поэтому строки 119–132 проверяют URI на соответствие `ONE_CONTACT`. Строка 122 выделяет из аргумента `Uri` последний сегмент, которым является уникальный идентификатор контакта. Строки 125–127 получают объект `SQLiteDatabase`, доступный для записи, а затем вызывают его метод `update` для обновления контакта значениями из аргумента `ContentValues`. Аргументы метода `update`:

- строка с именем обновляемой таблицы;
- объект `ContentValues` с именами обновляемых столбцов и их новыми значениями;

- строка, представляющая условие SQL `WHERE`, определяющее обновляемые записи;
- массив `String` с аргументами, которые подставляются на место заполнителей `?` в условии `WHERE`.

Если операция проходит успешно, метод `update` возвращает целое число — количество измененных записей; в противном случае `update` возвращает 0. Строка 136 оповещает `ContentResolver` об изменении базы данных, чтобы клиентский код `ContentResolver` мог отреагировать на изменения. Строка 139 возвращает количество измененных записей.

9.8.6. Переопределенный метод delete

Переопределенный метод `delete` класса `ContentProvider` (листинг 9.15) удаляет существующую запись.

Метод `delete` получает три аргумента:

- `uri` — объект `Uri`, представляющий удаляемую запись (или записи);
- `selection` — строка с условием SQL `WHERE`, определяющим удаляемые записи;
- `selectionArgs` — массив `String` с аргументами, заменяющими все заполнители аргументов (`?`) в строке `selection`.

Листинг 9.15. Переопределенный метод delete

```
142    // Удаление существующего контакта из базы данных
143    @Override
144       public int delete(Uri uri, String selection, String[] selectionArgs) {
145       int numberOfRowsDeleted;
146
147       switch (uriMatcher.match(uri)) {
148          case ONE_CONTACT:
149             // Получение из URI идентификатора контакта
150             String id = uri.getLastPathSegment();
151
152             // Удаление контакта
153             numberOfRowsDeleted = dbHelper.getWritableDatabase().delete(
154                Contact.TABLE_NAME, Contact._ID + "=" + id, selectionArgs);
155             break;
156          default:
157             throw new UnsupportedOperationException(
158                getContext().getString(R.string.invalid_delete_uri) + uri);
159       }
160
161       // Оповестить наблюдателей об изменениях в базе данных
162       if (numberOfRowsDeleted != 0) {
163          getContext().getContentResolver().notifyChange(uri, null);
```

```
164        }
165
166        return numberOfRowsDeleted;
167     }
168 }
```

В этом приложении удаление производится только для одного контакта, поэтому строки 147–159 проверяют URI на соответствие `ONE_CONTACT` — любой другой URI представляет неподдерживаемую операцию. Строка 150 выделяет из аргумента `Uri` последний сегмент, которым является уникальный идентификатор контакта. Строки 153–154 получают объект `SQLiteDatabase`, доступный для записи, а затем вызывают его метод `delete` для удаления контакта. В трех аргументах метода передаются таблица базы данных, из которой удаляется запись, условие `WHERE` и (если устройство `WHERE` получает аргументы) строковый массив значений, подставляемых в условие `WHERE`. Метод возвращает количество удаленных записей. Строка 163 оповещает `ContentResolver` об изменении базы данных, чтобы клиентский код `ContentResolver` мог отреагировать на изменения. Строка 166 возвращает количество удаленных записей.

9.9. Класс MainActivity

Класс `MainActivity` управляет фрагментами приложения и координирует взаимодействия между ними. На телефонах `MainActivity` в любой момент времени отображает только один фрагмент начиная с `ContactsFragment`. На планшетах `MainActivity` всегда отображает `ContactsFragment` слева и в зависимости от контекста — либо `DetailFragment`, либо `AddEditFragment` в правых 2/3 экрана.

9.9.1. Суперкласс, реализованные интерфейсы и поля

Класс `MainActivity` (листинг 9.16) использует класс `FragmentTransaction` (импортируемый в строке 6) из библиотеки поддержки `v4` для добавления и удаления фрагментов приложения. `MainActivity` реализует следующие три интерфейса.

- `ContactsFragment.ContactsFragment` (раздел 9.10.2) содержит методы обратного вызова, при помощи которых `ContactsFragment` сообщает `MainActivity`, что пользователь выбрал контакт в списке или добавил новый контакт.
- `DetailFragment.DetailFragmentListener` (раздел 9.13.2) содержит методы обратного вызова, при помощи которых `DetailFragment` сообщает `MainActivity`, что пользователь удаляет контакт или хочет отредактировать существующий контакт.

- `AddEditFragment.AddEditFragmentListener` (раздел 9.12.2) содержит методы обратного вызова, при помощи которых `AddEditFragment` сообщает `MainActivity`, что пользователь завершил добавление нового контакта или редактирование существующего контакта.

Константа `CONTACT_URI` (строка 17) используется как ключ в паре «ключ—значение», передаваемой между активностью `MainActivity` и ее фрагментами. Переменная экземпляра `contactsFragment` (строка 19) приказывает `ContactsFragment` обновить список контактов после добавления или удаления контакта.

Листинг 9.16. Суперкласс, реализованные интерфейсы и поля MainActivity

```
 1 // MainActivity.java
 2 // Управление фрагментами приложения и обмен данными между ними
 3 package com.deitel.addressbook;
 4
 5 import android.net.Uri;
 6 import android.os.Bundle;
 7 import android.support.v4.app.FragmentTransaction;
 8 import android.support.v7.app.AppCompatActivity;
 9 import android.support.v7.widget.Toolbar;
10
11 public class MainActivity extends AppCompatActivity
12    implements ContactsFragment.ContactsFragment,
13    DetailFragment.DetailFragmentListener,
14    AddEditFragment.AddEditFragmentListener {
15
16    // Ключ для сохранения Uri контакта в переданном объекте Bundle
17    public static final String CONTACT_URI = "contact_uri";
18
19    private ContactsFragment contactsFragment; // Вывод списка контактов
20
```

9.9.2. Переопределенный метод onCreate класса MainActivity

Метод `onCreate` (листинг 9.17) заполняет графический интерфейс `MainActivity` и, если приложение выполняется на телефоне, — отображает `ContactsFragment`. Если активность восстанавливается после завершения или создается повторно после изменения конфигурации, значение `savedInstanceState` будет отлично от `null`. В этом случае строки 43–45 просто получают ссылку на существующий объект `ContactsFragment` — на телефоне он будет сохранен, а на планшете он является частью макета `MainActivity`, заполняемого в строке 25.

Листинг 9.17. Переопределенный метод onCreate класса MainActivity

```
21    // Отображает ContactsFragment при первой загрузке MainActivity
22    @Override
23    protected void onCreate(Bundle savedInstanceState) {
```

```
24       super.onCreate(savedInstanceState);
25       setContentView(R.layout.activity_main);
26       Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
27       setSupportActionBar(toolbar);
28
29       // Если макет содержит fragmentContainer, используется макет для
30       // телефона; отобразить ContactsFragment
31       if (savedInstanceState != null &&
32          findViewById(R.id.fragmentContainer) != null) {
33          // Создание ContactsFragment
34          contactsFragment = new ContactsFragment();
35
36          // Добавление фрагмента в FrameLayout
37          FragmentTransaction transaction =
38             getSupportFragmentManager().beginTransaction();
39          transaction.add(R.id.fragmentContainer, contactsFragment);
40          transaction.commit(); // Вывод ContactsFragment
41       }
42       else {
43          contactsFragment =
44             (ContactsFragment) getSupportFragmentManager().
45                findFragmentById(R.id.contactsFragment);
46       }
47    }
48
```

Если `R.id.fragmentContainer` существует в макете `MainActivity` (строка 32), значит, приложение выполняется на телефоне. В этом случае строка 34 создает `ContactsFragment`, после чего в строках 37–40 объект `FragmentTransaction` используется для добавления `ContactsFragment` в пользовательский интерфейс. В строках 37–38 вызывается метод `beginTransaction` объекта `FragmentManager` для получения объекта `FragmentTransaction`. Затем в строке 39 метод `add` класса `FragmentTransaction` указывает, что при завершении `FragmentTransaction` фрагмент `ContactsFragment` должен быть присоединен к представлению с идентификатором, передаваемым в первом аргументе. Наконец, строка 41 использует метод `commit` класса `FragmentTransaction` для завершения транзакции и отображения фрагмента `ContactsFragment`.

9.9.3. Методы ContactsFragment.ContactsFragment

В листинге 9.18 приведены реализации методов обратного вызова интерфейса `ContactsFragment.ContactsFragment` в классе `MainActivity`. Метод `onContactSelected` (строки 50–60) вызывается объектом `ContactsFragment` для оповещения `MainActivity` о том, что пользователь выбрал контакт для отображения. Если приложение работает на телефоне (строка 52), то в строке 53 вызывается метод `displayContact` (раздел 9.9.4), который заменяет `ContactsFragment` в объекте `fragmentContainer` (см. определение в разделе 9.4.7) фрагментом

DetailFragment, содержащим информацию о контакте. На планшетах в строке 56 вызывается метод popBackStack класса FragmentManager для извлечения верхнего фрагмента из стека возврата, после чего строка 58 вызывает метод displayContact, заменяющий содержимое rightPaneContainer (см. раздел 9.4.7) фрагментом DetailFragment с подробной информацией о контакте.

Листинг 9.18. Методы ContactsFragment.ContactsFragment

```
49    // Отображение DetailFragment для выбранного контакта
50    @Override
51    public void onContactSelected(Uri contactUri) {
52       if (findViewById(R.id.fragmentContainer) != null) // Телефон
53          displayContact(contactUri, R.id.fragmentContainer);
54       else { // Планшет
55          // Извлечение с вершины стека возврата
56          getSupportFragmentManager().popBackStack();
57
58          displayContact(contactUri, R.id.rightPaneContainer);
59       }
60    }
61
62    // Отображение AddEditFragment для добавления нового контакта
63    @Override
64    public void onAddContact() {
65       if (findViewById(R.id.fragmentContainer) != null) // Телефон
66          displayAddEditFragment(R.id.fragmentContainer, null);
67       else // Планшет
68          displayAddEditFragment(R.id.rightPaneContainer, null);
69    }
70
```

Метод onAddContact (строки 63–69) вызывается объектом ContactsFragment для оповещения MainActivity о том, что пользователь выбрал команду добавления нового контакта. Если макет содержит fragmentContainer, то в строке 66 вызывается метод displayAddEditFragment (раздел 9.9.5), отображающий AddEditFragment в fragmentContainer. В противном случае строка 68 отображает фрагмент в rightPaneContainer. Во втором аргументе передается объект Bundle, при помощи которого AddEditFragment определяет, что именно происходит — добавление нового или редактирование существующего контакта. Передача null означает, что добавляется новый контакт; в противном случае объект Bundle включает Uri существующего контакта.

9.9.4. Метод displayContact класса MainActivity

Метод displayContact (листинг 9.19) создает фрагмент DetailFragment для отображения выбранного контакта. Чтобы передать аргументы фрагменту, разместите их в объекте Bundle, содержащем пары «ключ—значение», — мы

используем эту возможность для передачи Uri выбранного контакта, чтобы фрагмент DetailFragment знал, какой контакт следует получить от ContentProvider. Объект Bundle создается в строке 76. Строка 77 вызывает его метод putParcelable для сохранения пары «ключ—значение» с ключом CONTACT_URI (String) и значением contactUri (Uri). Класс Uri реализует интерфейс Parcelable, что позволяет сохранить Uri в Bundle как объект Parcel. Строка 78 передает объект Bundle методу setArguments фрагмента — далее фрагмент может извлечь информацию из Bundle (см. раздел 9.13).

Листинг 9.19. Метод displayContact

```
71    // Отображение информации о контакте
72    private void displayContact(Uri contactUri, int viewID) {
73       DetailFragment detailFragment = new DetailFragment();
74
75       // Передача URI контакта в аргументе DetailFragment
76       Bundle arguments = new Bundle();
77       arguments.putParcelable(CONTACT_URI, contactUri);
78       detailFragment.setArguments(arguments);
79
80       // Использование FragmentTransaction для отображения
81       FragmentTransaction transaction =
82          getSupportFragmentManager().beginTransaction();
83       transaction.replace(viewID, detailFragment);
84       transaction.addToBackStack(null);
85       transaction.commit(); // Приводит к отображению DetailFragment
86    }
87
```

В строках 81–82 мы получаем объект FragmentTransaction, после чего строка 83 вызывает метод replace класса FragmentTransaction; тем самым мы указываем, что при завершении FragmentTransaction фрагмент DetailFragment должен заменить содержимое представления с идентификатором, переданным в первом аргументе. Строка 84 вызывает метод addToBackStack класса FragmentTransaction для включения DetailFragment в стек возврата. Это делается для того, чтобы при нажатии кнопки Back фрагмент извлекался из стека возврата, а активность MainActivity могла извлечь фрагмент из стека на программном уровне. Аргумент метода addToBackStack может использоваться для извлечения нескольких фрагментов из стека для возврата к предшествующему состоянию после добавления нескольких фрагментов в стек. По умолчанию извлекается только верхний фрагмент.

9.9.5. Метод displayAddEditFragment класса MainActivity

Метод displayAddEditFragment (листинг 9.20) получает идентификатор ресурса представления, который сообщает, куда следует прикрепить AddEditFragment,

а также объект Uri, представляющий редактируемый контакт. Если второй аргумент равен null, добавляется новый контакт; в строке 90 создается объект AddEditFragment. Если аргумент contactUri отличен от null, строка 95 помещает его в объект Bundle, используемый для назначения аргументов Fragment. Затем в строках 100–104 создается объект FragmentTransaction, содержимое представления заменяется ресурсом с заданным идентификатором, фрагмент добавляется в стек возврата, после чего транзакция закрепляется.

Листинг 9.20. Метод displayAddEditFragment

```
88    // Отображение фрагмента для добавления или изменения контакта
89    private void displayAddEditFragment(int viewID, Uri contactUri) {
90       AddEditFragment addEditFragment = new AddEditFragment();
91
92       // При изменении передается аргумент contactUri
93       if (contactUri != null) {
94          Bundle arguments = new Bundle();
95          arguments.putParcelable(CONTACT_URI, contactUri);
96          addEditFragment.setArguments(arguments);
97       }
98
99       // Использование FragmentTransaction для отображения AddEditFragment
100       FragmentTransaction transaction =
101          getSupportFragmentManager().beginTransaction();
102       transaction.replace(viewID, addEditFragment);
103       transaction.addToBackStack(null);
104       transaction.commit(); // Приводит к отображению AddEditFragment
105    }
106
```

9.9.6. Методы DetailFragment.DetailFragmentListener

В листинге 9.21 приведены реализации методов обратного вызова MainActivity из интерфейса DetailFragment.DetailFragmentListener. Метод onContactDeleted (строки 108–113) вызывается DetailFragment для оповещения MainActivity об удалении контакта пользователем. В данном случае строка 111 извлекает DetailFragment из стека возврата, чтобы информация удаленного фрагмента не отображалась. Строка 112 вызывает метод updateContactList класса ContactsFragment для обновления списка контактов.

Листинг 9.21. Методы DetailFragment.DetailFragmentListener

```
107    // Возвращение к списку контактов при удалении текущего контакта
108    @Override
109    public void onContactDeleted() {
110       // Удаление с вершины стека
111       getSupportFragmentManager().popBackStack();
112       contactsFragment.updateContactList(); // Обновление контактов
```

```
113    }
114
115    // Отображение AddEditFragment для изменения существующего контакта
116    @Override
117    public void onEditContact(Uri contactUri) {
118       if (findViewById(R.id.fragmentContainer) != null) // Телефон
119          displayAddEditFragment(R.id.fragmentContainer, contactUri);
120       else // Планшет
121          displayAddEditFragment(R.id.rightPaneContainer, contactUri);
122    }
123
```

Метод `onEditContact` (строки 116–122) вызывается `DetailFragment` для оповещения `MainActivity`, когда пользователь касается команды на панели приложения для редактирования информации контакта. `DetailFragment` передает объект `Uri`, представляющий изменяемый контакт, чтобы его данные можно было отобразить в полях `EditText` фрагмента `AddEditFragment` для редактирования. Если макет содержит `fragmentContainer`, то строка 119 вызывает `displayAddEditFragment` (раздел 9.9.5) для отображения `AddEditFragment` в `fragmentContainer`; в противном случае строка 121 отображает `AddEditFragment` в `rightPaneContainer`.

9.9.7. Метод AddEditFragment.AddEditFragmentListener

Метод `onAddEditCompleted` (листинг 9.22) из интерфейса `AddEditFragment.AddEditFragmentListener` вызывается `AddEditFragment` для оповещения `MainActivity` о том, что пользователь сохраняет новый контакт или сохраняет изменения в существующем контакте. Строка 128 извлекает `AddEditFragment` из стека возврата, а строка 129 обновляет список контактов `ContactsFragment`.

Если приложение выполняется на планшете (строка 131), то строка 133 снова извлекает элемент из стека возврата, чтобы удалить фрагмент `DetailFragment` (если он присутствует). Затем строка 136 отображает подробную информацию нового или обновленного контакта на правой панели `rightPaneContainer`.

Листинг 9.22. Метод AddEditFragment.AddEditFragmentListener

```
124    // Обновление GUI после сохранения нового или существующего контакта
125    @Override
126    public void onAddEditCompleted(Uri contactUri) {
127       // Удаление вершины стека возврата
128       getSupportFragmentManager().popBackStack();
129       contactsFragment.updateContactList(); // Обновление контактов
130
131       if (findViewById(R.id.fragmentContainer) == null) { // Планшет
132          // Удаление с вершины стека возврата
133          getSupportFragmentManager().popBackStack();
134
```

```
135          // На планшете выводится добавленный или измененный контакт
136          displayContact(contactUri, R.id.rightPaneContainer);
137       }
138    }
139 }
```

9.10. Класс ContactsFragment

Класс `ContactsFragment` выводит список контактов в `RecyclerView`, а также предоставляет плавающую кнопку `FloatingActionButton` для добавления нового контакта.

9.10.1. Суперкласс и реализуемые интерфейсы

В листинге 9.23 приведены команды `package` и `import` класса `ContactsFragment`. `ContactsFragment` использует классы `LoaderManager` и `Loader` для выдачи запроса к `AddressBookContentProvider` и получения объекта `Cursor`, используемого `ContactsAdapter` (раздел 9.11) для поставки данных `RecyclerView`. Класс `ContactsFragment` реализует интерфейс `LoaderManager.LoaderCallbacks<Cursor>` (строка 23), чтобы он мог реагировать на вызовы методов из `LoaderManager`, создать `Loader` и обработать результаты, возвращаемые `AddressBookContentProvider`.

Листинг 9.23. Суперкласс и реализуемый интерфейс ContactsFragment

```
 1 // ContactsFragment.java
 2 // Субкласс Fragment для вывода алфавитного списка имен контактов
 3 package com.deitel.addressbook;
 4
 5 import android.content.Context;
 6 import android.database.Cursor;
 7 import android.net.Uri;
 8 import android.os.Bundle;
 9 import android.support.design.widget.FloatingActionButton;
10 import android.support.v4.app.Fragment;
11 import android.support.v4.app.LoaderManager;
12 import android.support.v4.content.CursorLoader;
13 import android.support.v4.content.Loader;
14 import android.support.v7.widget.LinearLayoutManager;
15 import android.support.v7.widget.RecyclerView;
16 import android.view.LayoutInflater;
17 import android.view.View;
18 import android.view.ViewGroup;
19
20 import com.deitel.addressbook.data.DatabaseDescription.Contact;
21
22 public class ContactsFragment extends Fragment
23    implements LoaderManager.LoaderCallbacks<Cursor> {
24
```

9.10.2. ContactsFragment

В листинге 9.24 объявляется вложенный интерфейс `ContactsFragment` с методами обратного вызова, которые реализуются `MainActivity` для оповещения о выборе пользователем контакта (строка 28) и о том, что пользователь коснулся кнопки `FloatingActionButton` для добавления нового контакта (строка 31).

Листинг 9.24. Вложенный интерфейс ContactsFragment

```
25    // Метод обратного вызова, реализуемый MainActivity
26    public interface ContactsFragment {
27       // Вызывается при выборе контакта
28       void onContactSelected(Uri contactUri);
29
30       // Вызывается при нажатии кнопки добавления
31       void onAddContact();
32    }
33
```

9.10.3. Поля

В листинге 9.25 представлены поля класса `ContactsFragment`. В строке 34 объявляется константа, используемая для идентификации `Loader` при обработке результатов, возвращаемых `AddressBookContentProvider`. В данном случае используется только один объект `Loader` — если класс использует несколько объектов `Loader`, с каждым должно быть связано уникальное целое значение для идентификации объекта в методах обратного вызова `LoaderManager.LoaderCallbacks<Cursor>`. Переменная экземпляра `listener` (строка 37) ссылается на объект, реализующий интерфейс (`MainActivity`). Переменная экземпляра `contactsAdapter` (строка 39) ссылается на объект `ContactsAdapter`, связывающий данные с `RecyclerView`.

Листинг 9.25. Поля ContactsFragment

```
34    private static final int CONTACTS_LOADER = 0; // Идентификатор Loader
35
36    // Сообщает MainActivity о выборе контакта
37    private ContactsFragment listener;
38
39    private ContactsAdapter contactsAdapter; // Адаптер для recyclerView
40
```

9.10.4. Переопределенный метод onCreateView

Переопределенный метод `onCreateView` класса `Fragment` (листинг 9.26) заполняет и настраивает графический интерфейс фрагмента. Большая часть этого кода уже

встречалась вам в предыдущих главах, так что здесь мы ограничимся новыми возможностями. Строка 47 сообщает, что у фрагмента `ContactsFragment` есть команды меню, которые должны отображаться на панели приложения `Activity` (или в меню команд). Строки 56–74 настраивают компонент `RecyclerView`. В строках 60–67 создается объект `ContactsAdapter`, заполняющий `RecyclerView`. В аргументе конструктора передается реализация интерфейса `ContactsAdapter.ContactClickListener` (раздел 9.11), указывающая, что при касании контакта должен вызываться метод `onContactSelected` интерфейса `ContactsFragment` с передачей `Uri` контакта, отображаемого в `DetailFragment`.

Листинг 9.26. Переопределенный метод onCreateView

```
41    // Настройка графического интерфейса фрагмента
42    @Override
43    public View onCreateView(
44       LayoutInflater inflater, ViewGroup container,
45       Bundle savedInstanceState) {
46       super.onCreateView(inflater, container, savedInstanceState);
47       setHasOptionsMenu(true); // У фрагмента есть команды меню
48
49       // Заполнение GUI и получение ссылки на RecyclerView
50       View view = inflater.inflate(
51          R.layout.fragment_contacts, container, false);
52       RecyclerView recyclerView =
53          (RecyclerView) view.findViewById(R.id.recyclerView);
54
55       // recyclerView выводит элементы в вертикальном списке
56       recyclerView.setLayoutManager(
57          new LinearLayoutManager(getActivity().getBaseContext()));
58
59       // создание адаптера recyclerView и слушателя щелчков на элементах
60       contactsAdapter = new ContactsAdapter(
61          new ContactsAdapter.ContactClickListener() {
62             @Override
63             public void onClick(Uri contactUri) {
64                listener.onContactSelected(contactUri);
65             }
66          }
67       );
68       recyclerView.setAdapter(contactsAdapter); // Назначение адаптера
69
70       // Присоединение ItemDecorator для вывода разделителей
71       recyclerView.addItemDecoration(new ItemDivider(getContext()));
72
73       // Улучшает быстродействие, если размер макета RecyclerView не изменяется
74       recyclerView.setHasFixedSize(true);
75
76       // Получение FloatingActionButton и настройка слушателя
77       FloatingActionButton addButton =
78          (FloatingActionButton) view.findViewById(R.id.addButton);
79       addButton.setOnClickListener(
80          new View.OnClickListener() {
```

```
81              // Отображение AddEditFragment при касании FAB
82              @Override
83              public void onClick(View view) {
84                 listener.onAddContact();
85              }
86           }
87       );
88
89       return view;
90    }
91
```

9.10.5. Переопределенные методы onAttach и onDetach

Класс `ContactsFragment` переопределяет методы `onAttach` и `onDetach` жизненного цикла фрагмента (листинг 9.27) для назначения переменной экземпляра `listener`. В нашем приложении `listener` присваивается ссылка на управляющую активность (строка 96) при присоединении `ContactsFragment` и `null` (строка 103) при отсоединении `ContactsFragment`.

Листинг 9.27. Переопределенные методы onAttach и onDetach

```
 92    // Присваивание ContactsFragment при присоединении фрагмента
 93    @Override
 94    public void onAttach(Context context) {
 95       super.onAttach(context);
 96       listener = (ContactsFragment) context;
 97    }
 98
 99    // Удаление ContactsFragment при отсоединении фрагмента
100    @Override
101    public void onDetach() {
102       super.onDetach();
103       listener = null;
104    }
105
```

9.10.6. Переопределенный метод onActivityCreated

Метод жизненного цикла `onActivityCreated` (листинг 9.28) вызывается после создания управляющей активности фрагмента и завершения выполнения метода `onCreateView` фрагмента — на этой стадии графический интерфейс фрагмента является частью иерархии представлений активности. Мы используем этот метод для того, чтобы приказать `LoaderManager` инициализировать `Loader` — важно, чтобы это происходило в то время, когда иерархия представлений уже существует, потому что компонент `RecyclerView` должен существовать для отображения загруженных данных. В строке 110 метод `getLoaderManager` фрагмента используется для получения объекта `LoaderManager` фрагмента.

Затем вызывается метод `initLoader` класса `LoaderManager`, который получает три аргумента:

- целочисленный идентификатор `Loader`;
- объект `Bundle` с аргументами конструктора `Loader` или `null` при отсутствии аргументов;
- ссылка на реализацию интерфейса `LoaderManager.LoaderCallbacks<Cursor>` (представляет `ContactsAdapter`) — реализации методов `onCreateLoader`, `onLoadFinished` и `onLoaderReset` этого интерфейса приведены в разделе 9.10.8.

Если активного объекта `Loader` с заданным идентификатором не существует, метод `initLoader` асинхронно вызывает метод `onCreateLoader` для создания и запуска объекта `Loader` для этого идентификатора.

Если активный объект `Loader` существует, то метод `initLoader` немедленно вызывает метод `onLoadFinished`.

Листинг 9.28. Переопределенный метод onActivityCreated

```
106    // Инициализация Loader при создании активности этого фрагмента
107    @Override
108    public void onActivityCreated(Bundle savedInstanceState) {
109       super.onActivityCreated(savedInstanceState);
110       getLoaderManager().initLoader(CONTACTS_LOADER, null, this);
111    }
112
```

9.10.7. Метод updateContactList

Метод `updateContactList` класса `ContactsFragment` (листинг 9.29) просто оповещает `ContactsAdapter` об изменении данных. Этот метод вызывается при добавлении новых контактов, а также обновлении или удалении существующих контактов.

Листинг 9.29. Метод updateContactList

```
113    // Вызывается из MainActivity при обновлении базы данных другим фрагментом
114    public void updateContactList() {
115       contactsAdapter.notifyDataSetChanged();
116    }
117
```

9.10.8. Методы LoaderManager.LoaderCallbacks<Cursor>

В листинге 9.30 приведены реализации методов обратного вызова интерфейса `LoaderManager.LoaderCallbacks<Cursor>` классом `ContactsFragment`.

Листинг 9.30. Методы LoaderManager.LoaderCallbacks<Cursor>

```
118    // Вызывается LoaderManager для создания Loader
119    @Override
120    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
121       // Создание CursorLoader на основании аргумента id; в этом
122       // фрагменте только один объект Loader, и команда switch не нужна
123       switch (id) {
124          case CONTACTS_LOADER:
125             return new CursorLoader(getActivity(),
126                Contact.CONTENT_URI, // Uri таблицы contacts
127                null, // все столбцы
128                null, // все записи
129                null, // без аргументов
130                Contact.COLUMN_NAME + " COLLATE NOCASE ASC"); // сортировка
131          default:
132             return null;
133       }
134    }
135
136    // Вызывается LoaderManager при завершении загрузки
137    @Override
138    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
139       contactsAdapter.swapCursor(data);
140    }
141
142    // Вызывается LoaderManager при сбросе Loader
143    @Override
144    public void onLoaderReset(Loader<Cursor> loader) {
145       contactsAdapter.swapCursor(null);
146    }
147 }
```

Метод onCreateLoader

`LoaderManager` вызывает метод `onCreateLoader` (строки 119–134) для создания и возвращения нового объекта `Loader` с заданным идентификатором; этим объектом `LoaderManager` управляет в контексте жизненного цикла фрагмента или активности. Строки 123–133 определяют создаваемый объект `Loader` на основании идентификатора, переданного в первом аргументе `onCreateLoader`.

ПЕРЕДОВОЙ ОПЫТ ПРОГРАММИРОВАНИЯ 9.1

В классе ContactsFragment достаточно одного объекта Loader, поэтому команда switch не нужна. Тем не менее мы все равно включили ее, чтобы код строился по стандартной схеме.

Строки 125–130 создают и возвращают объект `CursorLoader`, который обращается к `AddressBookContentProvider` для получения списка контактов, а затем предоставляет результаты в виде объекта `Cursor`. Конструктор `CursorLoader` получает объект `Context`, управляющий жизненным циклом `Loader`, и аргументы `uri`, `projection`, `selection`, `selectionArgs` и `sortOrder`, аналогичные одноименным

аргументам метода `query` класса `ContentProvider` (раздел 9.8.3). В данном случае в аргументах `projection`, `selection` и `selectionArgs` передаются значения `null` — это означает, что контакты должны быть отсортированы по имени без учета регистра символов.

Метод onLoadFinished

Метод `onLoadFinished` (строки 137–140) вызывается `LoaderManager` после того, как объект `Loader` завершит загрузку своих данных и вы сможете перейти к обработке результатов в аргументе `Cursor`. В данном случае мы вызываем метод `swapCursor` класса `ContactsAdapter` с передачей `Cursor` в аргументе, так что `ContactsAdapter` может обновить компонент `RecyclerView` на основании нового содержимого `Cursor`.

Метод onLoaderReset

Метод `onLoaderReset` (строки 143–146) вызывается `LoaderManager` тогда, когда происходит сброс объекта `Loader`, а его данные становятся недоступными. В этот момент приложение должно немедленно разорвать связь с данными. В таком случае мы вызываем метод `swapCursor` класса `ContactsAdapter` с аргументом `null`, показывая тем самым, что данные для связывания с `RecyclerView` отсутствуют.

9.11. Класс ContactsAdapter

В разделе 8.6 рассматривалось создание объекта `RecyclerView.Adapter`, используемого для передачи данных `RecyclerView`. В этом разделе выделен только новый код, который помогает `ContactsAdapter` (листинг 9.31) заполнить список `RecyclerView` именами контактов из `Cursor`.

Листинг 9.31. Субкласс RecyclerView.Adapter, связывающий контакты с RecyclerView

```
 1 // ContactsAdapter.java
 2 // Субкласс RecyclerView.Adapter, связывающий контакты с RecyclerView
 3 package com.deitel.addressbook;
 4
 5 import android.database.Cursor;
 6 import android.net.Uri;
 7 import android.support.v7.widget.RecyclerView;
 8 import android.view.LayoutInflater;
 9 import android.view.View;
10 import android.view.ViewGroup;
11 import android.widget.TextView;
12
13 import com.deitel.addressbook.data.DatabaseDescription.Contact;
14
15 public class ContactsAdapter
```

```
extends RecyclerView.Adapter<ContactsAdapter.ViewHolder> {

// Интерфейс реализуется ContactsFragment для обработки
// прикосновения к элементу в списке RecyclerView
public interface ContactClickListener {
   void onClick(Uri contactUri);
}

// Вложенный субкласс RecyclerView.ViewHolder используется
// для реализации паттерна View-Holder в контексте RecyclerView
public class ViewHolder extends RecyclerView.ViewHolder {
   public final TextView textView;
   private long rowID;

   // Настройка объекта ViewHolder элемента RecyclerView
   public ViewHolder(View itemView) {
      super(itemView);
      textView = (TextView) itemView.findViewById(android.R.id.text1);

      // Присоединение слушателя к itemView
      itemView.setOnClickListener(
         new View.OnClickListener() {
            // Выполняется при щелчке на контакте в ViewHolder
            @Override
            public void onClick(View view) {
               clickListener.onClick(Contact.buildContactUri(rowID));
            }
         }
      );
   }

   // Идентификатор записи базы данных для контакта в ViewHolder
   public void setRowID(long rowID) {
      this.rowID = rowID;
   }
}

// Переменные экземпляров ContactsAdapter
private Cursor cursor = null;
private final ContactClickListener clickListener;

// Конструктор
public ContactsAdapter(ContactClickListener clickListener) {
   this.clickListener = clickListener;
}

// Подготовка нового элемента списка и его объекта ViewHolder
@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
   // Заполнение макета android.R.layout.simple_list_item_1
   View view = LayoutInflater.from(parent.getContext()).inflate(
      android.R.layout.simple_list_item_1, parent, false);
   return new ViewHolder(view); // ViewHolder текущего элемента
}

// Назначает текст элемента списка
```

```
72    @Override
73    public void onBindViewHolder(ViewHolder holder, int position) {
74       cursor.moveToPosition(position);
75       holder.setRowID(cursor.getLong(cursor.getColumnIndex(Contact._ID)));
76       holder.textView.setText(cursor.getString(cursor.getColumnIndex(
77          Contact.COLUMN_NAME)));
78    }
79
80    // Возвращает количество элементов, предоставляемых адаптером
81    @Override
82    public int getItemCount() {
83       return (cursor != null) ? cursor.getCount() : 0;
84    }
85
86    // Текущий объект Cursor адаптера заменяется новым
87       public void swapCursor(Cursor cursor) {
88       this.cursor = cursor;
89       notifyDataSetChanged();
90    }
91 }
```

Вложенный интерфейс ContactClickListener

Строки 20–22 определяют вложенный интерфейс `ContactClickListener`, реализуемый классом `ContactsFragment` для уведомлений о том, что пользователь коснулся контакта в списке `RecyclerView`.

Каждому элементу `RecyclerView` назначается слушатель, который вызывает метод `onClick` интерфейса `ContactClickListener` и передает URI контакта, выбранного пользователем. Затем `ContactsFragment` оповещает `MainActivity` о том, что контакт был выбран, и `MainActivity` может вывести контакт в `DetailFragment`.

Вложенный класс ViewHolder

Класс `ViewHolder` (строки 26–51) хранит ссылку на компонент `TextView` элемента `RecyclerView` и значение `rowID` для соответствующего контакта в базе данных. Значение `rowID` необходимо из-за того, что контакты сортируются перед отображением, поэтому позиция каждого компонента в `RecyclerView` с большой вероятностью будет отличаться от его идентификатора записи в базе данных. Конструктор `ViewHolder` сохраняет ссылку на компонент `TextView` элемента `RecyclerView` и назначает его реализацию `View.OnClickListener`, которая передает URI контакта объекту `ContactClickListener` адаптера.

Переопределенный метод onCreateViewHolder класса RecyclerView.Adapter

Метод `onCreateViewHolder` (строки 63–69) заполняет графический интерфейс объекта `ViewHolder`. В нашем приложении используется предопределенный макет `android.R.layout.simple_list_item_1`, который определяет макет с одним компонентом `TextView` с именем `text1`.

Переопределенный метод onBindViewHolder класса RecyclerView.Adapter

Метод `onBindViewHolder` (строки 72–78) использует метод `moveToPosition` класса `Cursor` для перехода к контакту, соответствующему позиции текущего элемента `RecyclerView`. Строка 75 задает значение `rowID` для `ViewHolder`. Чтобы получить это значение, мы используем метод `getColumnIndex` класса `Cursor` для получения номера столбца `Contact._ID`. Полученное число передается методу `getLong` класса `Cursor` для получения идентификатора записи контакта. Строки 76–77 назначают текст компонента `textView` объекта `ViewHolder` с использованием аналогичного процесса — на этот раз с получением номера столбца `Contact.COLUMN_NAME` и последующим вызовом метода `getString` класса `Cursor` для получения имени контакта.

Переопределенный метод getItemCount класса RecyclerView.Adapter

Метод `getItemCount` (строки 81–84) возвращает общее количество строк в `Cursor` или 0, если равен `null`.

Метод swapCursor

Метод `swapCursor` (строки 87–90) заменяет текущий объект `Cursor` адаптера и уведомляет адаптер о том, что его данные изменились. Этот метод вызывается из методов `onLoadFinished` и `onLoaderReset` класса `ContactsFragment`.

9.12. Класс AddEditFragment

Класс `AddEditFragment` предоставляет интерфейс для добавления новых или редактирования существующих контактов. Многие концепции, использованные в коде, уже встречались в этой или предыдущих главах, поэтому описание будет ограничено новыми возможностями.

9.12.1. Суперкласс и реализуемый интерфейс

В листинге 9.32 приведены команды `package` и `import`, а также начало определения класса `AddEditFragment`. Класс расширяет `Fragment` и реализует интерфейс `LoaderManager.LoaderCallbacks<Cursor>` для реакции на события `LoaderManager`.

Листинг 9.32. Команды package и import класса AddEditFragment

```
1 // AddEditFragment.java
2 // Фрагмент для добавления нового или изменения существующего контакта
3 package com.deitel.addressbook;
4
```

```
 5 import android.content.ContentValues;
 6 import android.content.Context;
 7 import android.database.Cursor;
 8 import android.net.Uri;
 9 import android.os.Bundle;
10 import android.support.design.widget.CoordinatorLayout;
11 import android.support.design.widget.FloatingActionButton;
12 import android.support.design.widget.Snackbar;
13 import android.support.design.widget.TextInputLayout;
14 import android.support.v4.app.Fragment;
15 import android.support.v4.app.LoaderManager;
16 import android.support.v4.content.CursorLoader;
17 import android.support.v4.content.Loader;
18 import android.text.Editable;
19 import android.text.TextWatcher;
20 import android.view.LayoutInflater;
21 import android.view.View;
22 import android.view.ViewGroup;
23 import android.view.inputmethod.InputMethodManager;
24
25 import com.deitel.addressbook.data.DatabaseDescription.Contact;
26
27 public class AddEditFragment extends Fragment
28    implements LoaderManager.LoaderCallbacks<Cursor> {
29
```

9.12.2. Интерфейс AddEditFragmentListener

В листинге 9.33 объявляется вложенный интерфейс `AddEditFragmentListener`, содержащий метод обратного вызова `onAddEditCompleted`, реализуемый `MainActivity` для оповещения о сохранении пользователем нового или измененного существующего контакта.

Листинг 9.33. Вложенный интерфейс AddEditFragmentListener

```
30    // Определяет метод обратного вызова, реализованный MainActivity
31    public interface AddEditFragmentListener {
32       // Вызывается при сохранении контакта
33       void onAddEditCompleted(Uri contactUri);
34    }
35
```

9.12.3. Поля

В листинге 9.34 перечислены поля класса.

- Константа `CONTACT_LOADER` (строка 37) идентифицирует объект `Loader`, который обращается с запросом к `AddressBookContentProvider` для получения одного контакта для редактирования.

- Переменная экземпляра `listener` (строка 39) содержит ссылку на объект `AddEditFragmentListener` (`MainActivity`), который должен оповещаться о сохранении нового или обновленного контакта.
- Переменная экземпляра `contactUri` (строка 40) представляет редактируемый контакт.
- Переменная экземпляра `addingNewContact` (строка 41) определяет тип операции: добавление нового контакта (`true`) или редактирование существующего контакта (`false`).
- Переменные экземпляров в строках 44–53 содержат ссылки на компоненты `TextInputLayout`, `FloatingActionButton` и `CoordinatorLayout` фрагмента.

Листинг 9.34. Поля AddEditFragment

```
36    // Константа для идентификации Loader
37    private static final int CONTACT_LOADER = 0;
38
39    private AddEditFragmentListener listener; // MainActivity
40    private Uri contactUri; // Uri выбранного контакта
41    private boolean addingNewContact = true; // Добавление (true) или изменение
42
43    // Компоненты EditText для информации контакта
44    private TextInputLayout nameTextInputLayout;
45    private TextInputLayout phoneTextInputLayout;
46    private TextInputLayout emailTextInputLayout;
47    private TextInputLayout streetTextInputLayout;
48    private TextInputLayout cityTextInputLayout;
49    private TextInputLayout stateTextInputLayout;
50    private TextInputLayout zipTextInputLayout;
51    private FloatingActionButton saveContactFAB;
52
53    private CoordinatorLayout coordinatorLayout; // Для SnackBar
54
```

9.12.4. Переопределенные методы onAttach, onDetach и onCreateView

В листинге 9.34 приведены переопределенные методы `onAttach`, `onDetach` и `onCreateView`. Методы `onAttach` и `onDetach` присваивают переменной экземпляра `listener` ссылку на управляющую активность при присоединении `AddEditFragment` или `null` при отсоединении `AddEditFragment`.

Листинг 9.35. Переопределенные методы onAttach, onDetach и onCreateView

```
55    // Назначение AddEditFragmentListener при присоединении фрагмента
56    @Override
57    public void onAttach(Context context) {
58       super.onAttach(context);
```

```
59        listener = (AddEditFragmentListener) context;
60     }
61
62     // Удаление AddEditFragmentListener при отсоединении фрагмента
63     @Override
64     public void onDetach() {
65        super.onDetach();
66        listener = null;
67     }
68
69     // Вызывается при создании представлений фрагмента
70     @Override
71     public View onCreateView(
72        LayoutInflater inflater, ViewGroup container,
73        Bundle savedInstanceState) {
74        super.onCreateView(inflater, container, savedInstanceState);
75        setHasOptionsMenu(true); // У фрагмента есть команды меню
76
77        // Заполнение GUI и получение ссылок на компоненты EditText
78        View view =
79           inflater.inflate(R.layout.fragment_add_edit, container, false);
80        nameTextInputLayout =
81           (TextInputLayout) view.findViewById(R.id.nameTextInputLayout);
82        nameTextInputLayout.getEditText().addTextChangedListener(
83           nameChangedListener);
84        phoneTextInputLayout =
85           (TextInputLayout) view.findViewById(R.id.phoneTextInputLayout);
86        emailTextInputLayout =
87           (TextInputLayout) view.findViewById(R.id.emailTextInputLayout);
88        streetTextInputLayout =
89           (TextInputLayout) view.findViewById(R.id.streetTextInputLayout);
90        cityTextInputLayout =
91           (TextInputLayout) view.findViewById(R.id.cityTextInputLayout);
92        stateTextInputLayout =
93           (TextInputLayout) view.findViewById(R.id.stateTextInputLayout);
94        zipTextInputLayout =
95           (TextInputLayout) view.findViewById(R.id.zipTextInputLayout);
96
97        // Назначение слушателя событий FloatingActionButton
98        saveContactFAB = (FloatingActionButton) view.findViewById(
99           R.id.saveFloatingActionButton);
100       saveContactFAB.setOnClickListener(saveContactButtonClicked);
101       updateSaveButtonFAB();
102
103       // Используется для отображения SnackBar с короткими сообщениями
104       coordinatorLayout = (CoordinatorLayout) getActivity().findViewById(
105          R.id.coordinatorLayout);
106
107       Bundle arguments = getArguments(); // null при создании контакта
108
109       if (arguments != null) {
110          addingNewContact = false;
111          contactUri = arguments.getParcelable(MainActivity.CONTACT_URI);
112       }
113
```

```
114       // При изменении существующего контакта создать Loader
115       if (contactUri != null)
116          getLoaderManager().initLoader(CONTACT_LOADER, null, this);
117
118       return view;
119    }
120
```

Метод `onCreateView` заполняет графический интерфейс, получает ссылки на компоненты `TextInputLayout` и настраивает `FloatingActionButton`. Затем метод `getArguments` фрагмента используется для получения объекта `Bundle` с аргументами (строка 107). При запуске `AddEditFragment` из `MainActivity` вместо `Bundle` передается `null`, потому что пользователь добавляет информацию нового контакта. В этом случае метод `getArguments` вернет `null`. Если он вернет `Bundle` (строка 109), значит, пользователь редактирует существующий контакт. Строка 111 читает `Uri` контакта из `Bundle` вызовом метода `getParcelable`. Если значение `contactUri` отлично от `null`, строка 116 использует объект `LoaderManager` фрагмента для инициализации объекта `Loader`, который будет использоваться `AddEditFragment` для получения данных редактируемого контакта.

9.12.5. nameChangedListener и метод updateSaveButtonFAB

В листинге 9.36 приведен объект класса `TextWatcher` с именем `nameChangedListener` и метод `updatedSaveButtonFAB`. Слушатель вызывает метод `updatedSaveButtonFAB`, когда пользователь редактирует текст в поле `EditText`, связанном с `nameTextInputLayout`. Имя в данном приложении не может быть пустым, поэтому метод `updatedSaveButtonFAB` отображает `FloatingActionButton` только в том случае, если компонент `EditText`, связанный с `nameTextInputLayout`, содержит текст.

Листинг 9.36. nameChangedListener и метод updateSaveButtonFAB

```
121    // Обнаруживает изменения в тексте поля EditTex, связанного
122    // с nameTextInputLayout, для отображения или скрытия saveButtonFAB
123    private final TextWatcher nameChangedListener = new TextWatcher() {
124       @Override
125       public void beforeTextChanged(CharSequence s, int start, int count,
126          int after) {}
127
128       // Вызывается при изменении текста в nameTextInputLayout
129       @Override
130       public void onTextChanged(CharSequence s, int start, int before,
131          int count) {
132          updateSaveButtonFAB();
133       }
134
135       @Override
```

```
136       public void afterTextChanged(Editable s) { }
137    };
138
139    // Кнопка saveButtonFAB видна, если имя не пусто
140    private void updateSaveButtonFAB() {
141       String input =
142          nameTextInputLayout.getEditText().getText().toString();
143
144       // Если для контакта указано имя, показать FloatingActionButton
145       if (input.trim().length() != 0)
146          saveContactFAB.show();
147       else
148          saveContactFAB.hide();
149    }
150
```

9.12.6. saveContactButtonClicked и метод saveContact

Когда пользователь касается кнопки `FloatingActionButton` этого фрагмента, выполняется слушатель `saveContactButtonClicked` (листинг 9.37, строки 152–162). Метод `onClick` скрывает клавиатуру (строки 157–159), а затем вызывает метод `saveContact`.

Листинг 9.37. saveContactButtonClicked и метод saveContact

```
151    // Реагирует на событие, генерируемое при сохранении контакта
152    private final View.OnClickListener saveContactButtonClicked =
153       new View.OnClickListener() {
154          @Override
155          public void onClick(View v) {
156             // Скрыть виртуальную клавиатуру
157             ((InputMethodManager) getActivity().getSystemService(
158                Context.INPUT_METHOD_SERVICE)).hideSoftInputFromWindow(
159                getView().getWindowToken(), 0);
160             saveContact(); // Сохранение контакта в базе данных
161          }
162       };
163
164    // Сохранение информации контакта в базе данных
165    private void saveContact() {
166       // Создание объекта ContentValues с парами "ключ–значение"
167       ContentValues contentValues = new ContentValues();
168       contentValues.put(Contact.COLUMN_NAME,
169          nameTextInputLayout.getEditText().getText().toString());
170       contentValues.put(Contact.COLUMN_PHONE,
171          phoneTextInputLayout.getEditText().getText().toString());
172       contentValues.put(Contact.COLUMN_EMAIL,
173          emailTextInputLayout.getEditText().getText().toString());
174       contentValues.put(Contact.COLUMN_STREET,
175          streetTextInputLayout.getEditText().getText().toString());
176       contentValues.put(Contact.COLUMN_CITY,
177          cityTextInputLayout.getEditText().getText().toString());
```

```
178       contentValues.put(Contact.COLUMN_STATE,
179          stateTextInputLayout.getEditText().getText().toString());
180       contentValues.put(Contact.COLUMN_ZIP,
181          zipTextInputLayout.getEditText().getText().toString());
182
183       if (addingNewContact) {
184          // Использовать объект ContentResolver активности для вызова
185          // insert для объекта AddressBookContentProvider
186          Uri newContactUri = getActivity().getContentResolver().insert(
187             Contact.CONTENT_URI, contentValues);
188
189          if (newContactUri != null) {
190             Snackbar.make(coordinatorLayout,
191                R.string.contact_added, Snackbar.LENGTH_LONG).show();
192             listener.onAddEditCompleted(newContactUri);
193          }
194          else {
195             Snackbar.make(coordinatorLayout,
196                R.string.contact_not_added, Snackbar.LENGTH_LONG).show();
197          }
198       }
199       else {
200          // Использовать объект ContentResolver активности для вызова
201          // update для объекта AddressBookContentProvider
202          int updatedRows = getActivity().getContentResolver().update(
203             contactUri, contentValues, null, null);
204
205          if (updatedRows > 0) {
206             listener.onAddEditCompleted(contactUri);
207             Snackbar.make(coordinatorLayout,
208                R.string.contact_updated, Snackbar.LENGTH_LONG).show();
209          }
210          else {
211             Snackbar.make(coordinatorLayout,
212                R.string.contact_not_updated, Snackbar.LENGTH_LONG).show();
213          }
214       }
215    }
216
```

Метод `saveContact` (строки 165–215) создает объект `ContentValues` (строка 167) и добавляет в него пары «ключ—значение», представляющие имена столбцов и значения, которые должны вставляться или обновляться в базе данных (строки 168–181). Если пользователь добавляет новый контакт (строки 183–198), то в строках 186–187 метод `ContentResolver` используется для вызова `insert` для `AddressBookContentProvider` и включения нового контакта в базу данных. Если вставка проходит успешно, то возвращаемый объект `Uri` не равен `null`, и строки 190–192 выводят уведомление `SnackBar` о добавлении контакта, после чего оповещают `AddEditFragmentListener` с передачей добавленного контакта. Напомним, что при выполнении приложения на планшете это приводит к отображению данных контакта в `DetailFragment` рядом с `ContactsFragment`. Если

попытка вставки завершилась неудачей, то в строках 195–195 отображается соответствующее уведомление `SnackBar`.

Если пользователь редактирует существующий контакт (строки 199–214), то в строках 202–203 метод `update` класса `ContentResolver` используется для вызова `update` для `AddressBookContentProvider` и сохранения данных измененного контакта. Если обновление прошло успешно, возвращаемое целое число (количество обновленных записей) больше 0; в этом случае строки 206–208 оповещают `AddEditFragmentListener` с передачей измененного контакта, после чего выводится сообщение. Если попытка обновления завершилась неудачей, то в строках 211–212 отображается соответствующее уведомление `SnackBar`.

9.12.7. Методы LoaderManager.LoaderCallbacks<Cursor>

В листинге 9.38 представлены реализации методов интерфейса `LoaderManager.LoaderCallbacks<Cursor>` из `AddEditFragment`. Эти методы используются в классе `AddEditFragment` только в том случае, когда пользователь редактирует существующий контакт. Метод `onCreateLoader` (строки 219–233) создает `CursorLoader` для конкретного редактируемого контакта. Метод `onLoadFinished` (строки 236–267) проверяет, отличен ли объект `Cursor` от `null`, и если отличен — вызывает метод `moveToFirst`. Если метод возвращает `true`, значит, контакт, соответствующий `contactUri`, был найден в базе данных; тогда строки 241–263 получают информацию контакта из `Cursor` и отображают ее в графическом интерфейсе. Метод `onLoaderReset` в `AddEditFragment` не используется, поэтому он ничего не делает.

Листинг 9.38. Методы LoaderManager.LoaderCallbacks<Cursor>

```
217    // Вызывается LoaderManager для создания Loader
218    @Override
219    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
220       // Создание CursorLoader на основании аргумента id; в этом
221       // фрагменте только один объект Loader, и команда switch не нужна
222       switch (id) {
223          case CONTACT_LOADER:
224             return new CursorLoader(getActivity(),
225                contactUri, // Uri отображаемого контакта
226                null, // Все столбцы
227                null, // Все записи
228                null, // Без аргументов
229                null); // Порядок сортировки
230          default:
231             return null;
232       }
233    }
234
235    // Вызывается LoaderManager при завершении загрузки
236    @Override
```

```
237    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
238       // Если контакт существует в базе данных, вывести его информацию
239       if (data != null && data.moveToFirst() ) {
240          // Получение индекса столбца для каждого элемента данных
241          int nameIndex = data.getColumnIndex(Contact.COLUMN_NAME);
242          int phoneIndex = data.getColumnIndex(Contact.COLUMN_PHONE);
243          int emailIndex = data.getColumnIndex(Contact.COLUMN_EMAIL);
244          int streetIndex = data.getColumnIndex(Contact.COLUMN_STREET);
245          int cityIndex = data.getColumnIndex(Contact.COLUMN_CITY);
246          int stateIndex = data.getColumnIndex(Contact.COLUMN_STATE);
247          int zipIndex = data.getColumnIndex(Contact.COLUMN_ZIP);
248
249          // Заполнение компонентов EditText полученными данными
250          nameTextInputLayout.getEditText().setText(
251             data.getString(nameIndex));
252          phoneTextInputLayout.getEditText().setText(
253             data.getString(phoneIndex));
254          emailTextInputLayout.getEditText().setText(
255             data.getString(emailIndex));
256          streetTextInputLayout.getEditText().setText(
257             data.getString(streetIndex));
258          cityTextInputLayout.getEditText().setText(
259             data.getString(cityIndex));
260          stateTextInputLayout.getEditText().setText(
261             data.getString(stateIndex));
262          zipTextInputLayout.getEditText().setText(
263             data.getString(zipIndex));
264
265          updateSaveButtonFAB();
266       }
267    }
268
269    // Вызывается LoaderManager при сбросе Loader
270    @Override
271    public void onLoaderReset(Loader<Cursor> loader) { }
272 }
```

9.13. Класс DetailFragment

Класс `DetailFragment` выводит информацию одного контакта и предоставляет команды меню на панели приложения, при помощи которых пользователь может изменить или удалить данные контакта.

9.13.1. Суперкласс и реализуемый интерфейс

В листинге 9.39 приведены команды `package`, `import` и начало объявления класса `DetailFragment`. Класс расширяет `Fragment` и реализует интерфейс `LoaderManager.LoaderCallbacks<Cursor>` для реакции на события `LoaderManager`.

Листинг 9.39. Команды package и import класса DetailFragment

```
 1 // DetailFragment.java
 2 // Субкласс Fragment для вывода подробной информации о контакте
 3 package com.deitel.addressbook;
 4
 5 import android.app.AlertDialog;
 6 import android.app.Dialog;
 7 import android.content.Context;
 8 import android.content.DialogInterface;
 9 import android.database.Cursor;
10 import android.net.Uri;
11 import android.os.Bundle;
12 import android.support.v4.app.DialogFragment;
13 import android.support.v4.app.Fragment;
14 import android.support.v4.app.LoaderManager;
15 import android.support.v4.content.CursorLoader;
16 import android.support.v4.content.Loader;
17 import android.view.LayoutInflater;
18 import android.view.Menu;
19 import android.view.MenuInflater;
20 import android.view.MenuItem;
21 import android.view.View;
22 import android.view.ViewGroup;
23 import android.widget.TextView;
24
25 import com.deitel.addressbook.data.DatabaseDescription.Contact;
26
27 public class DetailFragment extends Fragment
28    implements LoaderManager.LoaderCallbacks<Cursor> {
29
```

9.13.2. Интерфейс DetailFragmentListener

В листинге 9.40 объявляется вложенный интерфейс `DetailFragmentListener`, содержащий методы обратного вызова, реализуемые `MainActivity` для оповещения об удалении контакта (строка 32) и касании команды меню для редактирования контакта (строка 35).

Листинг 9.40. Вложенный интерфейс DetailFragmentListener

```
30    // Методы обратного вызова, реализованные MainActivity
31    public interface DetailFragmentListener {
32       void onContactDeleted(); // Вызывается при удалении контакта
33
34       // Передает URI редактируемого контакта DetailFragmentListener
35       void onEditContact(Uri contactUri);
36    }
37
```

9.13.3. Переменные экземпляров DetailFragment

В листинге 9.41 приведены переменные экземпляров класса.

- Константа `CONTACT_LOADER` (строка 38) идентифицирует объект `Loader`, который обращается с запросом к `AddressBookContentProvider` для получения одного контакта для отображения.
- Переменная экземпляра `listener` (строка 40) содержит ссылку на объект `DetailFragmentListener` (`MainActivity`), который должен оповещаться об удалении контакта или начале его редактирования.
- Переменная экземпляра `contactUri` (строка 41) представляет отображаемый контакт.
- Переменные экземпляров в строках 43–49 содержат ссылки на компоненты `TextView` фрагмента.

Листинг 9.41. Поля DetailFragment

```
38    private static final int CONTACT_LOADER = 0; // Идентифицирует Loader
39
40    private DetailFragmentListener listener; // MainActivity
41    private Uri contactUri; // Uri выбранного контакта
42
43    private TextView nameTextView; // Имя контакта
44    private TextView phoneTextView; // Телефон
45    private TextView emailTextView; // Электронная почта
46    private TextView streetTextView; // Улица
47    private TextView cityTextView; // Город
48    private TextView stateTextView; // Штат
49    private TextView zipTextView; // Почтовый индекс
50
```

9.13.4. Переопределенные методы onAttach, onDetach и onCreateView

В листинге 9.42 приведены переопределенные методы `onAttach`, `onDetach` и `onCreateView` фрагмента. Методы `onAttach` и `onDetach` присваивают переменной экземпляра ссылку на управляющую активность при присоединении `DetailFragment` и `null` при отсоединении `DetailFragment`. Метод `onCreateView` (строки 66–95) получает объект `Uri` выбранного контакта (строки 74–77). Строки 80–90 заполняют графический интерфейс и получают ссылки на компоненты `TextView`. В строке 93 объект `LoaderManager` фрагмента используется для инициализации объекта `Loader`, который будет получать данные отображаемого контакта.

Листинг 9.42. Переопределенные методы onAttach, onDetach и onCreateView

```
51    // Назначение DetailFragmentListener при присоединении фрагмента
52    @Override
53    public void onAttach(Context context) {
54       super.onAttach(context);
55       listener = (DetailFragmentListener) context;
56    }
57
58    // Удаление DetailFragmentListener при отсоединении фрагмента
59    @Override
60    public void onDetach() {
61       super.onDetach();
62       listener = null;
63    }
64
65    // Вызывается при создании представлений фрагмента
66    @Override
67    public View onCreateView(
68       LayoutInflater inflater, ViewGroup container,
69       Bundle savedInstanceState) {
70       super.onCreateView(inflater, container, savedInstanceState);
71       setHasOptionsMenu(true); // У фрагмента есть команды меню
72
73       // Получение объекта Bundle с аргументами и извлечение URI
74       Bundle arguments = getArguments();
75
76       if (arguments != null)
77          contactUri = arguments.getParcelable(MainActivity.CONTACT_URI);
78
79       // Заполнение макета DetailFragment
80       View view =
81          inflater.inflate(R.layout.fragment_detail, container, false);
82
83       // Получение компонентов EditTexts
84       nameTextView = (TextView) view.findViewById(R.id.nameTextView);
85       phoneTextView = (TextView) view.findViewById(R.id.phoneTextView);
86       emailTextView = (TextView) view.findViewById(R.id.emailTextView);
87       streetTextView = (TextView) view.findViewById(R.id.streetTextView);
88       cityTextView = (TextView) view.findViewById(R.id.cityTextView);
89       stateTextView = (TextView) view.findViewById(R.id.stateTextView);
90       zipTextView = (TextView) view.findViewById(R.id.zipTextView);
91
92       // Загрузка контакта
93       getLoaderManager().initLoader(CONTACT_LOADER, null, this);
94       return view;
95    }
96
```

9.13.5. Переопределенные методы onCreateOptionsMenu и onOptionsItemSelected

Меню `DetailsFragment` предоставляет команды для редактирования текущего контакта и его удаления. Метод `onCreateOptionsMenu` (листинг 9.43,

строки 98–102) заполняет меню по ресурсному файлу `fragment_details_menu.xml`. Метод `onOptionsItemSelected` (строки 105–117) использует идентификатор ресурса выбранного элемента `MenuItem` для определения того, какой элемент был выбран. Если пользователь выбрал команду редактирования (), строка 109 вызывает метод `onEditContact` интерфейса `DetailsFragmentListener` с `contactUri`. Если пользователь выбрал команду удаления (), то строка 112 вызывает метод `deleteContact` (листинг 9.43).

Листинг 9.43. Переопределенные методы onCreateOptionsMenu и onOptionsItemSelected

```
97    // Отображение команд меню фрагмента
98    @Override
99    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
100       super.onCreateOptionsMenu(menu, inflater);
101       inflater.inflate(R.menu.fragment_details_menu, menu);
102    }
103
104    // Обработка выбора команд меню
105    @Override
106    public boolean onOptionsItemSelected(MenuItem item) {
107       switch (item.getItemId()) {
108          case R.id.action_edit:
109             listener.onEditContact(contactUri); // Передача Uri слушателю
110             return true;
111          case R.id.action_delete:
112             deleteContact();
113             return true;
114       }
115
116       return super.onOptionsItemSelected(item);
117    }
118
```

9.13.6. Метод deleteContact и фрагмент confirmDelete

Метод `deleteContact` (листинг 9.44, строки 120–123) отображает фрагмент `DialogFragment` (строки 126–157) с предложением подтвердить удаление текущего контакта. Если удаление подтверждается, в строках 147–148 вызывается метод `delete` класса `ContentResolver` для вызова метода `delete` класса `AddressBookContentProvider` и удаления контакта из базы данных. Метод `delete` получает URI удаляемых данных, строку с описанием условия `WHERE`, определяющего удаляемые данные, и строковый массив аргументов, подставляемых в условие `WHERE`. В данном случае последние два аргумента равны `null`, потому что идентификатор записи удаляемого контакта встроен в URI — он извлекается из `Uri` методом `delete` класса `AddressBookContentProvider`. Строка 149 вызывает метод `onContactDeleted` слушателя, чтобы активность `MainActivity` могла удалить `DetailFragment` с экрана.

Листинг 9.44. Методы deleteContact и confirmDelete класса DialogFragment

```
119    // Удаление контакта
120    private void deleteContact() {
121       // FragmentManager используется для отображения confirmDelete
122       confirmDelete.show(getFragmentManager(), "confirm delete");
123    }
124
125    // DialogFragment для подтверждения удаления контакта
126    private final DialogFragment confirmDelete =
127       new DialogFragment() {
128          // Создание объекта AlertDialog и его возвращение
129          @Override
130          public Dialog onCreateDialog(Bundle bundle) {
131             // Создание объекта AlertDialog.Builder
132             AlertDialog.Builder builder =
133                new AlertDialog.Builder(getActivity());
134
135             builder.setTitle(R.string.confirm_title);
136             builder.setMessage(R.string.confirm_message);
137
138             // Кнопка OK просто закрывает диалоговое окно
139             builder.setPositiveButton(R.string.button_delete,
140                new DialogInterface.OnClickListener() {
141                   @Override
142                   public void onClick(
143                      DialogInterface dialog, int button) {
144
145                      // объект ContentResolver используется
146                      // для вызова delete в AddressBookContentProvider
147                      getActivity().getContentResolver().delete(
148                         contactUri, null, null);
149                      listener.onContactDeleted(); // Оповещение слушателя
150                   }
151                }
152             );
153
154             builder.setNegativeButton(R.string.button_cancel, null);
155             return builder.create(); // Вернуть AlertDialog
156          }
157       };
158
```

9.13.7. Методы LoaderManager.LoaderCallback<Cursor>

В листинге 9.45 приведены реализации методов интерфейса `LoaderManager.LoaderCallbacks<Cursor>` класса `DetailFragment`. Метод `onCreateLoader` (строки 160–181) создает объект `CursorLoader` для конкретного отображаемого контакта. Метод `onLoadFinished` (строки 184–206) проверяет, отличен ли объект `Cursor` от `null`, и если отличен — вызывает метод `moveToFirst`. Если метод возвращает `true`, значит, контакт, соответствующий `contactUri`, был найден в базе данных;

тогда строки 189–204 получают информацию контакта из `Cursor` и отображают ее в графическом интерфейсе. Метод `onLoaderReset` в `DetailFragment` не используется, поэтому он ничего не делает.

Листинг 9.45. Методы LoaderManager.LoaderCallback<Cursor>

```
159    // Вызывается LoaderManager для создания Loader
160    @Override
161    public Loader<Cursor> onCreateLoader(int id, Bundle args) {
162       // Создание CursorLoader на основании аргумента id; в этом
163       // фрагменте только один объект Loader, и команда switch не нужна
164       CursorLoader cursorLoader;
165
166       switch (id) {
167          case CONTACT_LOADER:
168             cursorLoader = new CursorLoader(getActivity(),
169                contactUri, // Uri отображаемого контакта
170                null, // Все столбцы
171                null, // Все записи
172                null, // Без аргументов
173                null); // Порядок сортировки
174             break;
175          default:
176             cursorLoader = null;
177             break;
178       }
179
180       return cursorLoader;
181    }
182
183    // Вызывается LoaderManager при завершении загрузки
184    @Override
185    public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
186       // Если контакт существует в базе данных, вывести его информацию
187       if (data != null && data.moveToFirst()) {
188          // Получение индекса столбца для каждого элемента данных
189          int nameIndex = data.getColumnIndex(Contact.COLUMN_NAME);
190          int phoneIndex = data.getColumnIndex(Contact.COLUMN_PHONE);
191          int emailIndex = data.getColumnIndex(Contact.COLUMN_EMAIL);
192          int streetIndex = data.getColumnIndex(Contact.COLUMN_STREET);
193          int cityIndex = data.getColumnIndex(Contact.COLUMN_CITY);
194          int stateIndex = data.getColumnIndex(Contact.COLUMN_STATE);
195          int zipIndex = data.getColumnIndex(Contact.COLUMN_ZIP);
196
197          // Заполнение TextView полученными данными
198          nameTextView.setText(data.getString(nameIndex));
199          phoneTextView.setText(data.getString(phoneIndex));
200          emailTextView.setText(data.getString(emailIndex));
201          streetTextView.setText(data.getString(streetIndex));
202          cityTextView.setText(data.getString(cityIndex));
203          stateTextView.setText(data.getString(stateIndex));
204          zipTextView.setText(data.getString(zipIndex));
205       }
```

```
206    }
207
208    // Вызывается LoaderManager при сбросе Loader
209    @Override
210    public void onLoaderReset(Loader<Cursor> loader) { }
211 }
```

9.14. Резюме

В этой главе мы создали приложение `Address Book`, с помощью которого пользователи могут добавлять, просматривать, изменять либо удалять информацию о контактах, которая хранится в базе данных SQLite.

Для управления всеми фрагментами приложения используется одна активность. На устройствах с размером экрана, типичным для телефонов, фрагменты отображаются по одному. На планшетах активность отображала фрагмент, содержащий список контактов, а фрагменты для просмотра, добавления и редактирования контактов заменялись по мере необходимости. Объекты `FragmentManager` и `FragmentTransaction` использовались для динамического отображения фрагментов. Стек возврата был использован для автоматической поддержки кнопки `Back` системы Android. Для организации передачи данных между фрагментами и управляющей активностью в каждом субклассе `Fragment` был определен вложенный интерфейс методов обратного вызова, которые реализуются управляющей активностью.

Мы использовали субкласс `SQLiteOpenHelper` для упрощения создания базы данных и получения объекта `SQLiteDatabase`, с помощью которого выполняются операции с содержимым базы данных. Для управления запросами к базам данных использовался класс `Cursor` (пакет `android.database`).

Для асинхронного обращения к базе данных за пределами потока GUI был определен субкласс `ContentProvider`, который указывает, как должны выполняться операции выборки, вставки, обновления и удаления данных. При внесении изменений в базу данных SQLite объект `ContentProvider` оповещает слушателей о том, что в GUI можно обновить данные. Объект `ContentProvider` определял объекты `Uri` для идентификации выполняемых задач.

Для использования средств выборки, вставки, обновления и удаления вызывались соответствующие методы встроенного объекта `ContentResolver` активности. Вы узнали, что классы `ContentProvider` и `ContentResolver` организуют передачу данных за вас. Методы `ContentResolver` получает в первом аргументе объект `Uri`, который определяет объект `ContentProvider`. Каждый метод `ContentResolver` вызывает соответствующий метод `ContentProvider`, который в свою очередь использует `Uri` для определения выполняемой операции.

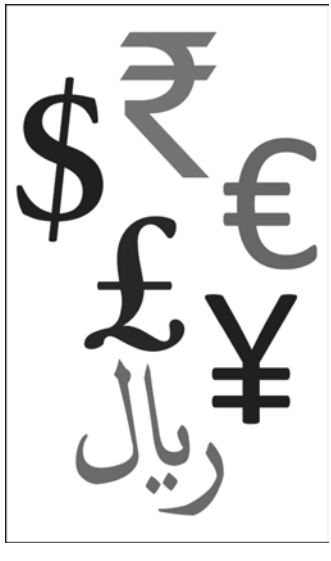
Как упоминалось ранее, долгие операции или операции, блокирующие выполнение приложения до своего завершения (например, обращения к файлам или базам данных), должны выполняться за пределами потока GUI. Для организации асинхронной работы с данными мы использовали класс `CursorLoader`. Вы узнали, что для создания объектов `Loader` и управления ими используется объект `LoaderManager` фрагмента или активности, связывающий жизненный цикл каждого объекта `Loader` с жизненным циклом активности или фрагмента. Мы реализовали интерфейс `LoaderManager.LoaderCallbacks` для реакции на события `Loader`, указывающие, когда объект `Loader` должен создаваться, завершать загрузку своих данных или сбрасываться в случае недоступности данных.

Основные пары «атрибут—значение» компонентов GUI были определены как ресурс стиля, после чего стиль был применен к компонентам `TextView` с информацией о контакте. Также мы определили границу для компонентов `TextView`, определив объект `Drawable` в качестве фона `TextView`. Таким объектом `Drawable` могло быть растровое изображение, но в этом приложении использовалась геометрическая фигура в файле ресурсов.

В главе 10 рассматривается коммерческая сторона программирования Android-приложений. Вы узнаете, как подготовить приложение для отправки в Google Play, включая подготовку значков; как протестировать приложения на устройствах и опубликовать их в Google Play. Мы рассмотрим характеристики успешных приложений и рекомендации по дизайну приложений, которые следует соблюдать. В этой главе приводятся рекомендации по выбору цены и маркетингу приложений. Также будут упомянуты преимущества предоставления бесплатной версии для увеличения сбыта других продуктов — например, версии приложения с расширенной функциональностью или привилегированного контента. Вы научитесь использовать Google Play для получения данных о продажах, проведения платежей и других целей.

10 Google Play и коммерческие аспекты разработки

В этой главе…



- Добавление значка и метки в манифест приложения
- Подготовка приложений для публикации
- Выбор цены, преимущества платных и бесплатных приложений
- Заработок на рекламе в приложениях
- Продажа виртуальных товаров с использованием внутренних платежей
- Регистрация в Google Play
- Создание учетной записи продавца
- Отправка приложений в Google Play
- Запуск Google Play из приложения
- Другие магазины приложений Android
- Другие популярные мобильные платформы, на которые можно портировать приложения для расширения круга пользователей
- Продвижение приложений на рынке

10.1. Введение

В главах 2–9 мы разработали несколько работоспособных приложений Android. После того как вы освоите разработку и тестирование ваших собственных приложений (как в эмуляторе, так и на устройствах Android), следующим шагом должна стать отправка в Google Play (и/или другие магазины приложений) для распространения их по всему миру. В этой главе вы узнаете:

- как зарегистрироваться в Google Play и создать учетную запись Google Payments, чтобы вы могли продавать свои приложения;
- как подготовить приложения к публикации;
- как загрузить их в Google Play.

В отдельных случаях мы будем давать ссылки на документацию Android, вместо того чтобы описывать конкретные действия, потому что они с большой вероятностью изменятся. Мы также расскажем о других магазинах приложений, в которых вы можете распространять свои приложения Android. Также будет рассмотрен вопрос о преимуществах платного и бесплатного распространения и перечислены такие механизмы извлечения прибыли, как реклама в приложениях и продажа виртуальных товаров. Мы представим некоторые ресурсы для продвижения приложений на рынок и упомянем о других платформах, на которые можно портировать приложения Android для расширения круга пользователей.

10.2. Подготовка приложений к публикации

Компания Google предоставляет обширную документацию, которая поможет разработчику освоиться в процессе публикации приложений. В документе «Preparing for Release» (*http://developer.android.com/tools/publishing/preparing.html*) перечислены ключевые моменты, которые следует учесть перед публикацией приложения в Google Play. В их числе:

- получение *криптографического ключа* для включения цифровой подписи в приложение;
- создание значка приложения;
- включение в приложение лицензионного соглашения End User License Agreement (этот шаг не обязателен);
- контроль версий приложения (например, 1.0, 1.1, 2.0, 2.3, 3.0);
- компиляция приложения;
- тестирование приложения на устройствах Android.

  http://developer.android.com/tools/testing/what_to_test.html

Также перед публикацией приложения стоит ознакомиться с контрольным списком Core App Quality (*http://developer.android.com/distribute/essentials/quality/core.html*), содержащим рекомендации по качеству приложений; с документом Tablet App Quality (*http://developer.android.com/distribute/essentials/quality/tablets.html*), содержащим рекомендации конкретно для планшетных приложений; списком Launch Checklist для публикации приложений в магазине Google Play (*http://developer.android.com/distribute/tools/launch-checklist.html*) и списком Localization Checklist для приложений, которые должны продаваться в других странах (*http://developer.android.com/distribute/tools/localization-checklist.html*). В оставшейся части этого раздела рассматриваются некоторые аспекты, которые следует учесть перед публикацией приложения.

10.2.1. Тестирование приложения

Прежде чем отправлять свое приложение в Google Play, тщательно протестируйте его на разных устройствах. Даже если приложение идеально работает на эмуляторе, при запуске его на конкретных устройствах Android могут возникнуть проблемы. Сервис Google Cloud Test Lab

https://developers.google.com/cloud-test-lab

позволяет протестировать приложение на широком диапазоне устройств.

10.2.2. Лицензионное соглашение

У вас имеется возможность включить в приложение лицензионное соглашение EULA (End User License Agreement). Это соглашение определяет условия, на которых вы предоставляете лицензию на свое приложение пользователю. Обычно в нем указываются правила пользования, ограничения на повторное распространение и декомпиляцию, ответственность за продукт, информация о соблюдении соответствующего законодательства и т. д. Возможно, вам стоит проконсультироваться у адвоката при подготовке EULA для вашего приложения. Пример EULA можно просмотреть по адресу

http://www.rocketlawyer.com/document/end-user-license-agreement.rl

10.2.3. Значки и метки

Спроектируйте значок для своего приложения и предоставьте текстовую метку (имя), которая будет отображаться в Google Play и на устройстве пользователя. В качестве значка можно использовать логотип компании, графику из приложения или специально созданное изображение. В документации материального

дизайна Google перечислены все аспекты, которые следует учесть при разработке значка приложения:

https://www.google.com/design/spec/style/icons.html

Значки продуктов должны иметь размеры 48×48 dp с границей 1 dp. Android масштабирует их до необходимого размера для разных размеров и вариантов плотности экрана. По этой причине в документации рекомендуется создать значок 192×192 dp с границей 4 dp — большое изображение после уменьшения смотрится лучше, чем маленькое изображение после увеличения.

Вам также понадобится значок высокого разрешения, который будет использоваться в Google Play. Он должен иметь следующие параметры:

- 512×512 пикселов;
- 32-разрядный формат PNG;
- максимальный размер 1 Мбайт.

Значок определяет первое впечатление потенциального пользователя, поэтому очень важно создать качественные значки. Рассмотрите возможность приглашения опытного дизайнера, который поможет вам создать привлекательный, профессиональный значок. В табл. 10.1 перечислены некоторые фирмы, предлагающие бесплатные значки профессионального уровня и коммерческие услуги по разработке значков по заказу клиентов. Когда значок будет готов, включите его в проект при помощи `Asset Studio` (как было сделано в разделе 4.4.9); в результате вы получите значки разных размеров, построенные на основе исходного значка.

Таблица 10.1. Фирмы, предоставляющие услуги по дизайну значков

Компания	URL-адрес	Услуги
glyphlab	http://www.glyphlab.com/icon_design/	Разработка значков по заказу, галерея значков для бесплатной загрузки
Iconiza	http://www.iconiza.com	Разработка значков по фиксированной цене, продажа готовых значков
The Iconfactory	http://iconfactory.com/home	Продажа готовых значков и разработка по заказу
Rosetta®	http://icondesign.rosetta.com/	Разработка значков по заказу
The Noun Project	https://thenounproject.com/	Тысячи значков, разработанных разными дизайнерами
Elance®	http://www.elance.com	Поиск дизайнеров, занимающихся разработкой значков

10.2.4. Контроль версии приложения

Разработчик должен включить в приложение *имя версии* (которое видят пользователи) и *код версии* (целый номер, используемый Google Play) и продумать стратегию нумерации обновлений. Например, первой версии приложения можно присвоить имя 1.0, незначительным обновлениям — имена 1.1 и 1.2, а следующему основному обновлению — 2.0. Код версии представляет собой целое число, которое обычно начинается с 1 и увеличивается на 1 для каждой новой версии приложения. Дополнительную информацию можно найти в разделе «Versioning Your Applications» по адресу

http://developer.android.com/tools/publishing/versioning.html

10.2.5. Лицензирование для управления доступом к платным приложениям

Сервис лицензирования Google Play позволяет создавать лицензионные политики, управляющие доступом к платным приложениям. Например, лицензионная политика позволяет ограничить количество устройств, на которых может быть установлено приложение. За дополнительной информацией о сервисе лицензирования обращайтесь по адресу

http://developer.android.com/google/play/licensing/index.html

10.2.6. Маскировка кода

Приложения, отправляемые в Google Play, следует «маскировать», чтобы помешать декомпиляции кода и обеспечить дополнительную защиту приложений. Бесплатная утилита `ProGuard` (которая запускается при построении окончательных версий) сокращает размер файла `.apk` (файл приложения Android, содержащий пакет для установки), оптимизирует и маскирует код «посредством удаления неиспользуемого кода и замены имен классов, полей и методов семантически несодержательными именами»[1]. За информацией о настройке и использовании программы `ProGuard` обращайтесь по адресу

http://developer.android.com/tools/help/proguard.html

10.2.7. Получение закрытого ключа для цифровой подписи

Прежде чем отправлять свое приложение на устройство, в Google Play или другой магазин, вы должны снабдить файл `.apk` *цифровой подписью*, которая

[1] *http://developer.android.com/tools/help/proguard.html.*

будет идентифицировать вас как автора приложения. Цифровой сертификат включает имя разработчика или название компании, контактную информацию и т. д. Вы можете создать цифровую подпись самостоятельно с использованием *закрытого ключа* (например, надежного пароля, используемого для шифрования сертификата); покупать сертификат у сторонней сертифицирующей организации не обязательно (хотя такой вариант тоже возможен). Android Studio автоматически снабжает приложение цифровой подписью при его выполнении на эмуляторе или на устройстве в целях отладки. Этот цифровой сертификат *не может* использоваться в Google Play. За подробными инструкциями о создании цифровых подписей обращайтесь к разделу «Signing Your Applications» по адресу

http://developer.android.com/tools/publishing/app-signing.html

10.2.8. Снимки экрана

Магазин Google Play позволяет включить в описание приложения экранные снимки и изображения, по которым потенциальный покупатель сможет получить первое впечатление о вашем приложении.

Рекламное изображение

Рекламное изображение используется Google Play для продвижения приложений на телефонах, планшетах и на сайте Google Play. О важности рекламных изображений и требований к ним:

http://android-developers.blogspot.com/2011/10/android-market-featured-image.html

Снимки экрана и программа Android Device Monitor

Вы можете отправить до восьми снимков для каждого устройства, на котором будет работать ваше приложение, — смартфон, малый планшет, большой планшет, Android TV и Android Wear. Снимки дают предварительное представление о приложении, так как пользователь не сможет протестировать приложение перед загрузкой (хотя и может вернуть приложение с возвратом денег в течение двух часов после покупки и загрузки). Постарайтесь выбрать привлекательные снимки, которые демонстрируют функциональность приложения. Требования к изображениям перечислены в табл. 10.2.

Таблица 10.2. Параметры снимков экрана

Параметр	Описание
Размер	Минимальные длина или ширина — 320 пикселов, максимальные — 3840 пикселов (максимальный размер не должен превышать минимальный более чем вдвое)
Формат	24-разрядный формат PNG или JPEG без альфа-эффектов (прозрачности)

Для сохранения снимков экрана можно воспользоваться Android Device Monitor — эта программа устанавливается с Android Studio и упрощает отладку приложений, работающих на эмуляторах и физических устройствах. Выполните следующие действия.

1. Запустите приложение в эмуляторе или на устройстве.
2. В среде Android Studio выполните команду `Tools ▸ Android ▸ Android Device Monitor`, чтобы получить доступ к Android Device Monitor.
3. На вкладке `Devices` (рис. 10.1) выберите устройство, для которого будут делаться экранные снимки.
4. Нажмите кнопку `Screen Capture`. На экране появляется окно `Device Screen Capture`.

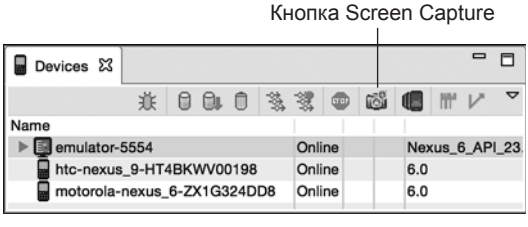


Рис. 10.1. Окно Devices

5. Убедитесь в том, что результат вас устраивает, и нажмите `Save` для сохранения полученного изображения.
6. Если вы захотите изменить содержимое экрана перед сохранением изображения, внесите изменения на устройстве (или AVD) и нажмите кнопку `Refresh` в окне `Device Screen Capture`. Вы также можете щелкнуть на кнопке `Rotate`, чтобы сохранить изображение в альбомной ориентации.

За дополнительной информацией об изображениях, которые вы можете включать в описание своего приложения, обращайтесь по адресу

https://support.google.com/googleplay/android-developer/answer/1078870

10.2.9. Информационный видеоролик

При отправке приложения в Google Play можно включить URL короткого информационного видеоролика, размещенного на сайте YouTube. Для этого необходимо иметь учетную запись YouTube и отправить свой ролик на сайт. Примеры информационных видеороликов приведены в табл. 10.3. На некоторых видеороликах показаны люди, которые держат устройство и работают

с приложением. В других роликах используются снимки экрана. В табл. 10.4 перечислены некоторые инструменты и сервисы для создания видеороликов и работы с ними (как бесплатные, так и коммерческие). Кроме того, Android Studio предоставляет функцию `Screen Record` в окне `Android Monitor`.

Таблица 10.3. Примеры информационных роликов для приложений в Google Play

Приложение	URL
Pac-Man 256	https://youtu.be/RF0GfRvm-yg
Angry Birds 2	https://youtu.be/jOUEjknadEY
Real Estate and Homes by Trulia®	https://youtu.be/BJDPKBNuqzE
Essential Anatomy	http://www.youtube.com/watch?v=U-oNyK9kl_Q

Таблица 10.4. Инструменты и сервисы для создания информационных видеороликов

Приложение	URL
Animoto	http://animoto.com
Apptamin	http://www.apptamin.com
CamStudio™	http://camstudio.org
Jing	http://www.techsmith.com/jing.html
Camtasia Studio®	http://www.techsmith.com/camtasia.html
TurboDemo™	http://www.turbodemo.com/eng/index.php

10.3. Цена приложения: платное или бесплатное?

Разработчик сам определяет цену на приложения, распространяемые через Google Play. Очень часто разработчики предлагают бесплатные приложения, которые рекламируют платные версии этих же приложений с расширенным набором функций и возможностей. При этом они получают прибыль в результате продажи продуктов и услуг, версий тех же приложений с расширенной функциональностью, продажи дополнительного контента или от рекламы, встроенной в приложение. В табл. 10.5 перечислены способы монетизации приложений. Описание способов монетизации, специфических для Google Play, приведено по адресу

http://developer.android.com/distribute/monetize/index.html

Таблица 10.5. Способы монетизации приложений

Способы монетизации приложений
Продажа приложения в Google Play
Продажа в других магазинах приложений Android
Продажа платных обновлений для приложения
Продажа виртуальных товаров (см. раздел 10.5)
Продажа приложения компании, которая займется его продвижением
Использование сервиса мобильной рекламы, встраиваемой в приложения (см. раздел 10.4)
Прямая продажа заказчикам рекламного места в приложении
Реклама версий приложения, обладающих расширенным набором функций

10.3.1. Платные приложения

Средняя цена приложения сильно зависит от категории. Например, по данным аналитического сайта AppBrain (*http://www.appbrain.com*), средняя цена игры-головоломки составляет $1,51, а средняя цена бизнес-приложения — $8,44.[1] И хотя на первый взгляд эти цены кажутся низкими, следует учитывать то, что успешные приложения могут продаваться десятками, сотнями тысяч или даже миллионами копий.

Прежде чем установить цену на приложение, ознакомьтесь с ценами конкурентов. Сколько стоят продаваемые ими приложения? Предоставляют ли эти приложения аналогичный набор функций? Будут ли ваши приложения более «продвинутыми»? Предлагает ли ваше приложение те же возможности, что и приложения конкурентов, но по более низкой цене? Хотите ли вы вернуть потраченные на разработку приложения средства и получить прибыль?

Если ваша бизнес-стратегия изменится, со временем платные приложения даже могут перейти на модель бесплатного распространения. Тем не менее в настоящее время такой переход невозможен.

Финансовые операции при продаже платных приложений на Google Play выполняются с помощью учетной записи Google Wallet (*http://google.com/wallet*), хотя пользователи некоторых мобильных операторов (AT&T, Sprint и T-Mobile) могут воспользоваться биллинговыми системами этих операторов для получения средств от клиентов, загрузивших платные приложения. Начисление на счет

[1] *http://www.appbrain.com/stats/android-market-app-categories.*

Google Wallet осуществляется ежемесячно.[1] Разработчик несет ответственность за оплату налогов с прибыли, полученной с помощью Google Play.

10.3.2. Бесплатные приложения

В настоящее время более 90% приложений для Android распространяется бесплатно, и они составляют подавляющее большинство загрузок.[2] Учитывая склонность пользователей к загрузке бесплатных приложений, предлагайте «упрощенную» версию приложения, которую пользователи смогут загрузить и опробовать бесплатно. Например, если вы написали игру, предложите версию с несколькими начальными уровнями. После прохождения бесплатных уровней на экране появится сообщение, в котором пользователю предлагается приобрести в Google Play коммерческую версию приложения с полным набором уровней. Также пользователю могут предлагаться дополнительные уровни прямо из приложения (см. раздел 10.5). Многие компании используют бесплатные приложения для продвижения своего товарного знака, а также продвижения других продуктов и услуг (табл. 10.6).

Таблица 10.6. Компании, использующие бесплатные приложения Android для продвижения своих брендов

Бесплатное приложение	Функциональность
Amazon® Mobile	Просмотр и приобретение товаров на сайте Amazon с мобильного устройства
Bank of America	Поиск банкоматов и отделений банка в вашем районе, проверка баланса и платежей
Best Buy®	Поиск и приобретение товаров на сайте Best Buy
CNN	Последние мировые новости, экстренные сообщения и видео в реальном времени
Epicurious Recipe	Тысячи кулинарных рецептов из журналов, издаваемых компанией Condé Nast, в том числе журналов Gourmet и Bon Appetit
ESPN® ScoreCenter	Настройка информационных панелей для отслеживания успехов ваших любимых спортивных команд (любителей и профессионалов)

[1] *http://support.google.com/googleplay/android-developer/answer/137997?hl=en&ref_topic=15867.*

[2] *http://www.statista.com/topics/1002/mobile-app-usage/.*

Таблица 10.6 (*окончание*)

Бесплатное приложение	Функциональность
NFL Mobile	Последние новости и обновления NFL, просмотр спортивных программ, отчеты NFL и другая информация
UPS® Mobile	Отслеживание почтовых отправлений, поиск утерянных посылок, примерная оценка стоимости почтовых отправлений и другой подобной информации
NYTimes	Чтение статей из журнала New York Times (бесплатно или за отдельную плату)
Pocket Agent™	Приложение от State Farm Insurance, с помощью которого можно связаться с агентом, заявить требование, найти местные отделения, получить информацию о банках State Farm и инвестиционных фондах и т. д.
Progressive® Insurance	Подача заявки и отправка фотографий с места происшествия, поиск локального агента, получение сведений о страховке при покупке нового автомобиля и ряда других сведений
USA Today®	Чтение статей из USA Today и получение сведений о результатах последних спортивных соревнований
Wells Fargo® Mobile	Мобильный поиск находящихся поблизости банкоматов и отделений банка, проверка баланса, проведение платежей и оплата счетов
Women's Health Workouts Lite	Различные физические упражнения из одного из популярных журналов для женщин

10.4. Монетизация приложений с помощью встроенной рекламы

Многие разработчики предлагают бесплатные приложения, доходы от которых обеспечиваются встроенной в приложение рекламой, — часто эта реклама имеет вид баннеров наподобие тех, что используются на веб-сайтах. Мобильные рекламные сети, такие как AdMob (*http://www.admob.com/*) и Google AdSense for Mobile (*http://www.google.com/adsense/start/*), собирают рекламные предложения от рекламодателей и отображают их в форме рекламы, актуальной для конкретного приложения (см. раздел 10.13). Прибыль от рекламы начисляется на основе количества просмотров. В первой сотне наиболее популярных бесплатных приложений прибыль от просмотра встроенной рекламы составляет от нескольких сотен до нескольких тысяч долларов в день. Конечно, встроенная

реклама не будет столь доходной для большинства приложений, поэтому если вы хотите вернуть средства, потраченные на разработку приложения, и получить прибыль, берите плату за приложение.

10.5. Внутренняя продажа виртуальных товаров

Реализованный в Google Play сервис внутренних продаж (*http://developer.android.com/google/play/billing/index.html*) позволяет продавать *виртуальные товары* (цифровой контент) с помощью приложений или устройств с Android 2.3 и выше (табл. 10.7). Механизм внутренних продаж работает только в приложениях, приобретенных через Google Play; если же приложение продается через другие виртуальные магазины, этот сервис *недоступен*. Чтобы воспользоваться механизмом внутренних продаж, потребуется учетная запись, дающая право публикации в Google Play (см. раздел 10.6), и учетная запись продавца Google Payments (см. раздел 10.7). Компания Google возвращает разработчику 70% выручки за покупки, сделанные в приложениях.

Таблица 10.7. Виртуальные товары

Подписки на журналы	Локализованные руководства	Аватары
Виртуальные предметы	Игровые уровни	Игровые сцены
Дополнительные функции	Рингтоны	Пиктограммы
Электронные открытки	Электронные подарки	Виртуальные деньги
Обои	Изображения	Виртуальные домашние животные
Аудиозаписи	Видеозаписи	Электронные книги

Продажа виртуальных товаров может принести бóльшую прибыль (*из расчета на одного пользователя*), чем реклама[1]. Среди приложений, особенно успешных в области продажи виртуальных товаров, стоит упомянуть Angry Birds, DragonVale, Zynga Poker, Bejeweled Blitz, NYTimes и Candy Crush Saga. Виртуальные товары особенно популярны в мобильных играх.

Чтобы включить поддержку внутренних покупок в приложение, обращайтесь к руководству по адресу

http://developer.android.com/google/play/billing/billing_integrate.html

[1] *http://www.businessinsider.com/its-morning-in-venture-capital-2012-5?utm_source=readme&utm_medium=rightrail&utm_term=&utm_content=6&utm_campaign=recirc.*

За дополнительной информацией о механизме встроенной покупки (подписка, примеры приложений, рекомендации из области безопасности, тестирование и т. д.) обращайтесь по адресу

http://developer.android.com/google/play/billing/billing_overview.html

Вы также можете пройти бесплатный обучающий курс Selling In-app Products по адресу

http://developer.android.com/training/in-app-billing/index.html

Внутренние покупки в приложениях, проданных через альтернативные магазины

Если вы намереваетесь продавать приложения в других магазинах (см. раздел 10.11), некоторые независимые провайдеры мобильных платежей предоставляют возможность встроить в приложения механизм покупки, реализованный с помощью библиотек соответствующих API (табл. 10.8), — использовать механизм внутренних покупок Google Play не удастся. Начните с создания дополнительной *заблокированной функциональности* (например, игровых уровней или аватаров). Если пользователь захочет что-либо приобрести, инструмент приобретения, встроенный в приложение, обрабатывает финансовую операцию и передает сообщение приложению, которое верифицирует платеж. Тогда приложение выполняет разблокировку дополнительной функциональности.

Таблица 10.8. Провайдеры мобильных платежей, используемых при покупках с помощью встроенной в приложения системы биллинга

Провайдер	URL	Описание
PayPal Mobile Payments Library	http://developer.paypal.com/webapps/developer/docs/classic/mobile/gs_MPL/	Пользователь щелкает на кнопке Pay with PayPal, входит в свою учетную запись PayPal, а затем щелкает на кнопке Pay
Amazon In-App Purchasing	https://developer.amazon.com/appsandservices/apis/earn/in-app-purchasing	Механизм внутренней покупки для приложений, продаваемых через магазин Amazon App Store for Android
Samsung In-App Purchase	http://developer.samsung.com/in-app-purchase	Механизм внутренней покупки для приложений, спроектированных специально для устройств Samsung
Boku	http://www.boku.com	Пользователь щелкает на кнопке Pay by Mobile, вводит номер мобильного телефона, а затем завершает операцию, отвечая на текстовое сообщение, полученное на мобильный телефон

10.6. Регистрация в Google Play

Чтобы иметь возможность публиковать приложения в магазине Google Play, следует создать учетную запись на веб-странице *http://play.google.com/apps/publish*.

За регистрацию взимается однократный взнос $25. В отличие от других мобильных платформ, Google Play *не требует одобрения загружаемых приложений* (хотя приложение проходит автоматизированную проверку на вредоносный код). Впрочем, разработчик должен соблюдать требования, изложенные в документе Google Play Developer Program Policies. Если приложение нарушает принципы этой политики, оно может быть удалено в любой момент. Повторяющиеся или серьезные нарушения положений этого документа могут привести к принудительному удалению учетной записи (табл. 10.9).

Таблица 10.9. Некоторые нарушения положений документа Google Play Content Policy for Developers (http://play.google.com/about/developer-content-policy.html#showlanguages)

Нарушение прав владельцев интеллектуальной собственности (товарные знаки, патенты и копирайты)	Нарушение неприкосновенности частной жизни пользователя
Нарушение закона	Препятствия в деятельности других сторон
Намеренный ввод пользователя в заблуждение относительно назначения приложения	Повреждение личных данных или устройства пользователя
Намеренные действия, направленные на рост расходов пользователя мобильной сети передачи данных или провайдера беспроводной сети	Азартные игры
Фальсификация или обман	Пропаганда ненависти и насилия
Поддержка контента порнографического или непристойного содержания либо любого другого контента, недопустимого для просмотра лицами до 18 лет	Реклама в виджетах и оповещениях системного уровня

10.7. Создание учетной записи Google Payments

Для продажи приложений через магазин Google Play вам понадобится учетная запись сервиса Google Payments, доступного разработчикам Google Play более чем в 150 странах[1]. После регистрации и входа в Google Play (*http://play.google.com/apps/publish/*) (раздел 10.6) щелкните на ссылке `set up a merchant account`. При создании учетной записи необходимо предоставить:

[1] *http://support.google.com/googleplay/android-developer/answer/150324?hl=en&ref_topic=15867*.

- информацию, по которой Google сможет связаться с вами;
- контактную информацию для службы поддержки, по которой с вами смогут связаться пользователи.

10.8. Отправка приложений в Google Play

Как только будут подготовлены все файлы и вы будете готовы к загрузке приложения, выполните действия, описанные в контрольном списке по адресу *http://developer.android.com/distribute/tools/launch-checklist.html*.

Подключитесь к Google Play по адресу *http://play.google.com/apps/publish* (см. раздел 10.6) и нажмите `Publish an Android App on Google Play`, чтобы начать процесс загрузки. Вам будет предложено отправить следующие компоненты.

- Пакетный файл приложения (`.apk`), содержащий файлы с кодом, ресурсы и файл манифеста.
- Не менее двух экранных снимков приложения, которые будут включены в описание в Google Play. Вы можете включить снимки для телефонов на базе Android, 7- и 10-дюймовых планшетов, Android TV и Android Wear.
- Значок приложения в высоком разрешении (512×512 пикселов), который будет размещен в Google Play.
- Графика, используемая группой Google Play Editorial для продвижения приложений и на странице вашего продукта. Изображения должны иметь размеры 1024 пикселов в ширину и 500 пикселов в высоту, в 24-разрядном формате PNG или JPEG без эффектов прозрачности.
- Рекламная графика для Google Play, которая может использоваться компанией Google, если она решит продвигать ваше приложение (не обязательно). Такие изображения должны иметь размеры 180 пикселов в ширину и 120 пикселов в высоту, в 24-разрядном формате PNG или JPEG без эффектов прозрачности.
- Видеоролик для Google Play (не обязательно). Можно приложить URL-адрес видеоролика приложения (например, ссылку на видеоролик в YouTube с демонстрацией работы приложения).

Также вам будет предложено сообщить дополнительную информацию о приложении.

1. **Язык.** По умолчанию описание приложения отображается на английском языке. Если нужно выводить сведения о приложении на дополнительных языках, выберите их из предоставленного списка (табл. 10.10).

Таблица 10.10. Языки описаний приложений в Google Play

Азербайджанский	Амхарский	Английский (Великобритания)	Английский	Арабский
Армянский	Африкаанс	Баскский	Белорусский	Бенгальский
Бирманский	Болгарский	Венгерский	Вьетнамский	Галисийский
Голландский	Греческий	Грузинский	Датский	Зулусский
Иврит	Индонезийский	Исландский	Испанский (Испания)	Испанский (Латинская Америка)
Испанский (США)	Итальянский	Индийский	Каталанский	Китайский (упрощенный)
Китайский (традиционный)	Корейский (Южная Корея)	Кхмерский	Киргизский	Лаосский
Латышский	Литовский	Македонский	Малайский	Малаялам
Маратхи	Монгольский	Немецкий	Непальский	Норвежский
Персидский	Польский	Португальский (Бразилия)	Португальский (Португалия)	Ретороманский
Румынский	Русский	Сербский	Сингальский	Словацкий
Словенский	Суахили	Тайский	Тамильский	Телугу
Турецкий	Украинский	Филиппинский	Финский	Французский
Французский (Канада)	Хинди	Хорватский	Чешский	Шведский
Эстонский	Японский			

2. **Заголовок** приложения, отображаемый на Google Play (не более 30 символов). Этот заголовок *не обязан* быть уникальным для каждого приложения Android.
3. **Краткое описание** приложения (максимум 80 символов).
4. **Описание** приложения и его свойств (максимум 4000 символов). В последнем разделе описания рекомендуется обосновать обязательность каждого разрешения и продемонстрировать, каким образом оно используется.
5. **Последние изменения.** Сводка изменений в последней версии приложения (максимум 500 символов).
6. **Промотекст.** Рекламный текст, используемый для маркетинга приложения (максимум 80 символов).
7. **Тип приложения.** Выберите тип: приложение (`Application`) или игра (`Games`).

8. **Категория.** Выберите категорию, которая соответствует вашей игре или приложению.
9. **Цена.** Чтобы продавать приложение, вам придется создать учетную запись продавца.
10. **Рейтинг.** Выберите ограничения по возрасту пользователей. За дополнительной информацией обращайтесь к разделу «Rating your application content for Google Play» по адресу *http://support.google.com/googleplay/android-developer/answer/188189*.
11. **Географические ограничения.** По умолчанию приложение будет доступно во всех странах, поддерживаемых Google Play в настоящее время и в будущем. При желании вы можете выбрать конкретные страны, в которых приложение должно быть доступно.
12. **Сайт.** В описание вашего приложения в Google Play будет включена ссылка `Visit Developer's Website`. Предоставьте прямую ссылку на страницу сайта, на которой пользователи, заинтересовавшиеся вашим приложением, смогут найти дополнительную информацию, рекламный текст, краткое описание, дополнительные снимки экранов, инструкции и т. д.
13. **Электронная почта.** В Google Play также будет включен ваш адрес электронной почты, чтобы покупатели могли обратиться к вам с вопросами, сообщениями об ошибках и т. д.
14. **Телефон.** Это поле лучше оставлять пустым, если только вы не организуете службу поддержки по телефону. Также номер службы поддержки можно разместить на сайте.
15. **Политика конфиденциальности.** Ссылка на документ с описанием политики конфиденциальности вашего приложения.

Кроме того, если в вашем приложении используются внутренние продажи или какие-либо сервисы Google, вы должны добавить их описание. За дополнительной информацией обращайтесь по адресу

http://developer.android.com/google/play/billing/billing_admin.html

10.9. Запуск Play Store из приложения

Для повышения продаж приложений предоставьте пользователю возможность запускать приложение `Play Store` (Google Play) непосредственно из приложения (обычно с помощью кнопки, отображаемой на экране приложения). В результате пользователь получит возможность загружать другие опубликованные вами приложения либо приобрести приложение с более широким набором функций, чем у загруженной «упрощенной» версии. Можно также предоставить пользователю возможность загрузки последних обновлений приложения через `Play Store`.

Существует два способа запуска приложения `Play Store`. Во-первых, можно вывести результаты поиска в Google Play приложений по таким критериям, как имя разработчика, имя пакета или строка символов. Например, если нужно стимулировать пользователей загружать другие ваши приложения, опубликованные на Google Play, включите соответствующую кнопку, отображаемую на экране приложения. Как только пользователь нажмет эту кнопку, запустится приложение `Play Store`, которое инициирует поиск приложений, включающих ваше имя или название компании. Второй вариант — отобразить страницу с информацией о соответствующем приложении на экране приложения `Play Store`. За информацией о том, как запустить `Play Store` из вашего приложения, обращайтесь к разделу «Linking Your Products» по адресу

http://developer.android.com/distribute/googleplay/promote/linking.html

10.10. Управление приложениями в Google Play

С помощью консоли разработчика Google Play Developer Console можно управлять учетной записью и приложениями, проверять пользовательский рейтинг вашего приложения (от 1 до 5 звездочек), реагировать на комментарии пользователей, отслеживать количество общих и активных установок приложения (разность между количеством установок и удалений приложения). Можно также ознакомиться со статистикой установок и загрузок копий приложения для различных версий Android, устройств и прочей информацией. В отчетах об ошибках содержится информация о сбоях и «зависаниях», полученная от пользователей. Если вы разработали обновление приложения, то с минимальными усилиями можете опубликовать новую версию. Можно также удалить приложение из Play Store, но если пользователи заблаговременно загрузили его, оно останется на пользовательских устройствах. Пользователи, которые удалили приложение, смогут установить его повторно даже после удаления (приложение остается на серверах Google до тех пор, пока не будет удалено за нарушение правил использования, изложенных в документе Terms of Service).

10.11. Другие магазины приложений Android

Помимо Google Play, приложения можно распространять с помощью других магазинов приложений (табл. 10.11) или даже с помощью вашего собственного веб-сайта при участии таких служб, как AndroidLicenser (*http://www.androidlicenser.com*). За дополнительной информацией о публикации приложений на сторонних сайтах обращайтесь по адресу

http://developer.android.com/tools/publishing/publishing_overview.html

Таблица 10.11. Другие магазины приложений Android

Магазин	URL
Amazon Appstore	https://developer.amazon.com/public/solutions/platforms/android
Opera Mobile Store	http://android.oms.apps.opera.com/en_us/
Moborobo	http://www.moborobo.com
Appitalism®	http://www.appitalism.com/index.html
GetJar	http://www.getjar.com
SlideMe	http://www.slideme.org
AndroidPit	http://www.androidpit.com

10.12. Другие популярные платформы мобильных приложений и портирование

По данным *statista.com*, в 2016 году пользователи загрузят 225 миллиардов приложений, а в 2017 году количество загрузок достигнет 270 миллиардов[1]. Портирование приложений Android на другие мобильные платформы, прежде всего iOS (устройства iPhone, iPad и iPod Touch), позволяет расширить круг потенциальных пользователей (табл. 10.12). Существуют различные инструменты, используемые для автоматизации портирования. Например, компания Microsoft предоставляет средства портирования приложения для iOS и Android в систему Windows; аналогичные средства существуют для портирования приложений Android в iOS и наоборот[2]. Также стоит отметить некоторые средства кроссплатформенной разработки (табл. 10.13).

Таблица 10.12. Популярные платформы мобильных приложений (http://www.abiresearch.com/press/android-will-account-for-58-of-smartphone-app-down)

Платформа	URL
Android	http://developer.android.com
iOS (Apple)	http://developer.apple.com/ios
Windows	https://dev.windows.com/en-us/windows-apps

[1] *http://www.statista.com/statistics/266488/forecast-of-mobile-app-downloads/.*

[2] *http://www.wired.com/2015/04/microsoft-unveils-tools-moving-android-ios-apps-onto-windows.*

Таблица 10.13. Некоторые средства разработки кросс-платформенных мобильных приложений

Платформа	URL
Appcelerator Titanium	http://www.appcelerator.com/product/
PhoneGap	http://phonegap.com/
Sencha	https://www.sencha.com/
Visual Studio	https://www.visualstudio.com/en-us/features/mobile-app-development-vs.aspx
Xamarin	https://xamarin.com/

10.13. Маркетинг приложения

После публикации приложения представьте его аудитории будущих пользователей[1]. И в этом вам поможет *вирусный маркетинг* с помощью сайтов социальных сетей, таких как Facebook, Twitter, Google+ и YouTube. Эти сайты имеют огромную аудиторию. По данным Pew Research Center, 71% взрослых пользователей Интернета используют социальные сети[2]. В табл. 10.14 представлены наиболее популярные сайты социальных сетей. Зачастую в качестве недорогих и эффективных средств маркетинга используются рассылки по электронной почте и электронные бюллетени новостей.

Таблица 10.14. Популярные сайты социальных сетей

Название	URL	Описание
Facebook	http://www.facebook.com	Социальная сеть
Instagram	https://instagram.com/	Обмен фото и видео
Twitter	http://www.twitter.com	Микроблоги, социальная сеть
Google+	http://plus.google.com	Социальная сеть
Vine	http://vine.co	Социальная сеть для обмена видео
Tumblr	http://www.tumblr.com	Блоги

[1] На тему маркетинга мобильных приложений написано много книг. Информацию о последних публикациях можно найти по адресу *http://amzn.to/1ZgpYxZ*.

[2] *http://bits.blogs.nytimes.com/2015/01/09/americans-use-more-online-social-networks/?_r=0*.

Таблица 10.14 (*окончание*)

Название	URL	Описание
Groupon	http://www.groupon.com	Выгодные предложения
Foursquare	http://www.foursquare.com	Публикация отметок геопозиционирования
Snapchat	http://www.snapchat.com	Обмен видеосообщениями
Pinterest	http://www.pinterest.com	Публикация фотографий
YouTube	http://www.youtube.com	Видеоролики
LinkedIn	http://www.linkedin.com	Социальные сети для бизнеса
Flickr	http://www.flickr.com	Публикация фотографий

Facebook

Facebook, один из самых популярных сайтов социальных сетей, имеет почти 1,5 миллиарда активных пользователей[1], из которых почти 1 миллиард активен ежедневно[2]. Это превосходный ресурс *вирусного маркетинга* (путем распространения слухов). Начните с создания официальной страницы Facebook для вашего приложения или компании. Используйте эту страницу для публикации информации о приложении, новостей, обновлений, обзоров, рекомендаций, снимков экрана, рекордных счетов в играх, мнений пользователей и ссылок Google Play для загрузки приложения. Например, мы публикуем новости и обновления о своих книгах на странице Facebook по адресу *http://www.facebook.com/DeitelFan*.

Займитесь распространением информации. Попросите ваших коллег и друзей «лайкнуть» вашу страницу на «Фейсбуке» и порекомендовать своим друзьям и знакомым сделать то же самое. По мере того как пользователи будут посещать вашу страницу, заметки о впечатлениях будут появляться на страницах новостей друзей этих пользователей, способствуя росту армии поклонников вашего приложения.

Twitter

Twitter — это сайт микроблогов и социальных сетей, количество зарегистрированных пользователей которого превышает 1 миллиард, а 316 миллионов пользователей проявляют ежемесячную активность[3]. Пользователи публикуют «твиты» — сообщения длиной до 140 символов. Twitter предоставляет доступ к твитам всем последователям (на время написания книги одна известная рок-звезда имела более 40 миллионов последователей). Многие пользователи используют «Твиттер» для получения информации о новостях, поэтому можно рассылать твиты,

[1] *http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.*

[2] *http://expandedramblings.com/index.php/by-the-numbers-17-amazing-facebook-stats/.*

[3] *http://www.statisticbrain.com/twitter-statistics/.*

содержащие сведения о приложении (в том числе анонсы новых версий, советы, факты, комментарии для пользователей и другую информацию). Также посоветуйте друзьям и коллегам разослать твиты, содержащие сведения о приложении. Воспользуйтесь *хеш-тегом* (#) для создания ссылки на сообщение. Например, в процессе рассылки твитов, содержащих информацию о нашей книге, в нашей подписке на новости Twitter @*deitel* мы используем хеш-тег *#AndroidFP3*. Другие пользователи могут воспользоваться этим хеш-тегом, чтобы добавить комментарии о книге. Это облегчит поиск твитов для сообщений, связанных с книгой.

Вирусные видеоролики

Еще один способ распространения сведений о приложении — вирусные видеоролики, размещенные на сайтах публикации видеороликов в Интернете (YouTube), сайтах социальных сетей (например, Facebook, Instagram, Twitter, Google+) или путем рассылки сообщений по электронной почте. Если вы создадите эффектный видеоролик — юмористический или даже возмутительный, — это будет способствовать быстрому росту популярности вашего приложения среди пользователей социальных сетей.

Рассылки по электронной почте

Для продвижения приложения можно воспользоваться рассылками новостей по электронной почте. Добавьте ссылки на Google Play, с помощью которых пользователи смогут загрузить приложение. Также добавьте ссылки на страницы в социальных сетях, на которых можно получить новейшую информацию о приложении.

Обзоры приложений

Свяжитесь с влиятельными блогерами и сайтами, на которых публикуются обзоры приложений (табл. 10.15), и сообщите сведения о своем приложении. Предоставьте им промокод для бесплатной загрузки приложения (см. раздел 10.3). Учтите, что подобные сайты получают большое количество запросов, поэтому будьте кратки и предельно информативны. Многие обозреватели приложений публикуют видеообзоры приложений на YouTube и других подобных сайтах (табл. 10.16).

Таблица 10.15. Сайты, публикующие обзоры приложений Android

Название сайта	URL
Appolicious™	http://www.androidapps.com
AppBrain	http://www.appbrain.com
AppZoom	http://www.appzoom.com
Appstorm	http://android.appstorm.net

Таблица 10.15 (*окончание*)

Best Android Apps Review	http://www.bestandroidappsreview.com
Android App Review Source	http://www.androidappreviewsource.com
Androinica	http://www.androinica.com
AndroidLib	http://www.androlib.com
Android and Me	http://www.androidandme.com
AndroidGuys	http://www.androidguys.com/category/reviews
Android Police	http://www.androidpolice.com
AndroidPIT	http://www.androidpit.com
Phandroid	http://www.phandroid.com

Таблица 10.16. Сайты с видеообзорами приложений Android

Название сайта	URL
State of Tech	http://stateoftech.net/
Crazy Mike's Apps	http://crazymikesapps.com
Appolicious™	http://www.appvee.com/?device_filter=android
Life of Android™	http://www.lifeofandroid.com/video/

Связи с интернет-общественностью

В индустрии связей с общественностью применяются средства массовой информации, предназначенные для облегчения передачи сообщений от компаний конечным пользователям. Специалисты по связям с общественностью включают блоги, подкасты, RSS-подписки и социальные сети в свои PR-кампании. В табл. 10.17 перечислены некоторые платные и бесплатные интернет-ресурсы, посвященные связям с общественностью, включая файлы распространения пресс-релизов, службы создания пресс-релизов и другие ресурсы.

Таблица 10.17. PR-ресурсы в Интернете

Ресурс	URL	Описание
Бесплатные		
PRWeb®	http://www.prweb.com	Сервис публикации пресс-релизов (бесплатно или на платной основе)

Ресурс	URL	Описание
ClickPress™	http://www.clickpress.com	Отправьте на этот сайт свои новости для одобрения (бесплатно или на платной основе). Одобренные новости будут доступны на сайте ClickPress, а также на сайтах новостей. За плату ClickPress будет распространять пресс-релизы по глобальным каналам финансовых новостей
PRLog	http://www.prlog.org/pub/	Бесплатная отправка и распространение пресс-релизов
Newswire	http://www.newswire.com	Бесплатная отправка и распространение пресс-релизов
openPR®	http://www.openpr.com	Бесплатная публикация пресс-релизов
Платные		
PR Leap	http://www.prleap.com	Платная служба рассылки пресс-релизов в Интернете
Marketwired	http://www.marketwired.com	Платная служба рассылки пресс-релизов, которая позволяет выбирать целевую аудиторию по различным признакам: географический, отраслевой и т. д.
Mobility PR	http://www.mobilitypr.com	Службы, поддерживающие связи с общественностью компаний, работающих в индустрии мобильных устройств
eReleases	http://www.ereleases.com	Службы рассылки пресс-релизов, которые предлагают ряд дополнительных услуг: чтение, проверка на наличие ошибок и редактирование пресс-релизов, и т. д. Ознакомьтесь с рекомендациями по написанию эффективных пресс-релизов

Мобильная реклама

Приобретение рекламы (например, в других приложениях, в Интернете, в газетах и журналах либо на радио и телевидении) — это еще один способ маркетинга приложений. Мобильные рекламные сети (табл. 10.18) специализируются на рекламе мобильных приложений, разработанных для платформы Android (и других платформ). Многие из мобильных рекламных сетей могут выбирать целевую аудиторию по местоположению, провайдеру, устройству (например, Android, iOS, Windows, BlackBerry) и другим категориям. Имейте в виду, что прибыль от большинства приложений невелика, поэтому не тратьте много денег на рекламу.

Таблица 10.18. Мобильные рекламные сети

Мобильная рекламная сеть	URL
AdMob (Google)	http://www.admob.com/
Medialets	http://www.medialets.com
Tapjoy®	http://www.tapjoy.com
Millennial Media®	http://www.millennialmedia.com/
Smaato®	http://www.smaato.com
mMedia™	http://mmedia.com
InMobi™	http://www.inmobi.com

Вы также можете использовать средства мобильной рекламы для монетизации бесплатных приложений за счет включения рекламы (баннеров, видеороликов и т. д.). Средний показатель eCPM (effective cost per 1000 impressions, эффективная стоимость за тысячу показов) для рекламы в приложениях Android зависит от сети, устройства, региона и т. д. Большая часть платежей за рекламу на платформе Android основана на рейтинге «кликабельности» (CTR, clickthrough rate), а не на количестве генерируемых показов. Средний показатель CTR для встроенной рекламы в мобильных приложениях зависит от приложения, устройства, специализации рекламной сети и т. д. Если у вашего приложения много пользователей и показатели CTR для рекламы в нем высоки, доходы от рекламы могут быть довольно значительными. Кроме того, рекламная сеть может поставлять более высокооплачиваемую рекламу, увеличивая вашу прибыль.

10.14. Резюме

В этой главе был рассмотрен процесс регистрации на Google Play и настройки учетной записи Google Wallet, необходимой для продажи приложений. Мы показали, как подготовить приложение к публикации на Google Play, включая тестирование на эмуляторе и устройствах Android, и описания различных ресурсов, необходимых для отправки приложения в Google Play. Вы узнали, как загрузить приложение на Google Play, и познакомились с альтернативными магазинами, в которых можно продавать приложения. Вашему вниманию были предложены рекомендации по формированию цен на приложения и списки ресурсов с описаниями монетизации приложений с помощью встроенной в них рекламы и продаж виртуальных товаров с помощью приложений. Также были описаны ресурсы, которые могут использоваться для маркетинга приложений, распространяемых через Google Play.