самоучитель

Денис Колисниченко

Программирование для Android

3-е издание

Android Studio 3.4
Установка SDK и эмулятора
Разработка интерфейса пользователя
Графика, аудио и анимация
Доступ к Интернету, отправка SMS
Взаимодействие с аппаратными средствами мобильного устройства
СУБД SQLite
Технологии отладки мобильных приложений
Публикация и продвижение в Google Play

bhv®

Денис Колисниченко

самоучитель

Программирование для Android

3-е издание

Санкт-Петербург
«БХВ-Петербург»
2021

УДК 004.4
ББК 32.973.26-018.2
К60

Колисниченко Д. Н.

К60 Программирование для Android. Самоучитель. — 3-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 288 с.: ил.

ISBN 978-5-9775-6587-5

Рассмотрены все основные аспекты создания приложений для платформы Android 9 (API 28): установка необходимого программного обеспечения, использование эмулятора Android, создание интерфейса пользователя, работа с графикой, воспроизведение звука и видео, методы хранения данных (в том числе СУБД SQLite), взаимодействие с аппаратными средствами мобильного устройства, отладка приложений и их публикация в магазине Google Play.

Особое внимание уделено взаимодействию с аппаратными средствами смартфона. Показано, как получить информацию об устройстве и определить его состояние, использовать его датчики (акселерометр, датчик света, датчик температуры, датчик давления), камеру, Bluetooth-адаптер. Приведены решения для различных нештатных ситуаций (отказ эмулятора, проблема с установкой программного обеспечения и т. д.), что поможет начинающему программисту.

В 3-м издании описано создание 64-разрядных приложений для Android, рассмотрены изменения в интерфейсе среды разработки и новый API.

Для программистов

УДК 004.4
ББК 32.973.26-018.2

Группа подготовки издания:

Руководитель проекта	*Павел Шалин*
Зав. редакцией	*Екатерина Сависте*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн серии	*Марины Дамбиевой*
Дизайн обложки	*Карины Соловьевой*

Подписано в печать 07.07.20.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 23,22.
Тираж 1100 экз. Заказ №11503.
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Отпечатано с готового оригинал-макета
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-6587-5

Оглавление

Введение

> Я всегда мечтал о том, чтобы моим компьютером можно было пользоваться так же легко, как телефоном; моя мечта сбылась: я уже не могу разобраться, как пользоваться моим телефоном.
>
> *Бьёрн Страуструп (Bjarne Stroustrup)*

Когда-то мобильный телефон был именно мобильным телефоном: по нему можно было поговорить вне дома без необходимости таскать за собой многометровый телефонный кабель или подключать полутораметровую антенну, как на популярных лет двадцать назад радиотелефонах Senao.

Современный мобильный телефон давно уже перестал быть просто телефоном. Скорее всего — это компьютер (и весьма мощный!) с функциями телефона. Да и называют «мобильники» теперь *смартфонами* — чтобы подчеркнуть тот факт, что в руках мы держим весьма «умное» устройство, а не просто какой-то там телефон. Смартфон отличается от мобильного телефона наличием полноценной развитой операционной системы, что облегчает жизнь как разработчикам приложений для мобильных телефонов, так и пользователям. Впрочем, в продаже можно встретить и обычные мобильные телефоны — как правило, это устройства из так называемой «бюджетной» категории.

Итак, вы приобрели современный смартфон. У вас теперь есть возможность загружать и устанавливать новые мелодии, вы можете использовать его в качестве медиапроигрывателя и органайзера, Интернет и почта тоже доступны в вашем мобильном устройстве. Кроме того, вы можете расширить его функциональность путем установки дополнительных приложений.

Стоп! Так всеми этими функциями обладают и обычные «продвинутые» телефоны, пришедшие в свое время на смену простым «говорилкам», но не носящие гордого названия «смартфон». В чем же разница? А в том, что при разработке мобильного приложения для «продвинутых» телефонов разработчику приходится как минимум учитывать требования производителя того или иного телефона, а обычно — и особенности его модели. Вспомните всевозможные сервисы загрузки на телефон приложений и прочего контента: вам надо было отправить SMS на определенный номер, но SMS не пустое, а содержащее некий код, позволяющий идентифицировать ваш телефон. Например, 1 — для Nokia, 2 — для Samsung и т. п. Как правило, после

рекламы игры или другого какого-нибудь доступного для закачки приложения размещался целый список, включающий коды не только для телефонов разных производителей, но и для разных моделей одного и того же производителя. Пользователю легко было ошибиться — случайно отправить другой код вместо требуемого. В итоге с его счета снимались деньги, но приложение не работало. А разработчикам приходилось писать несколько версий одного и того же приложения, чтобы учесть особенности всех популярных моделей мобильных телефонов.

С появлением смартфонов все стало гораздо проще. Разработчики пишут приложения не под конкретный телефон, а под определенную мобильную операционную систему. А таких операционных систем гораздо меньше, чем моделей мобильных телефонов. Чаще всего теперь встречаются последние версии iOS (для смартфонов фирмы Apple) и Android. Разработка остальных мобильных ОС — вроде Windows Phone или Symbian OS — свернута, и устройства, работающие под их управлением, сейчас редкость.

В настоящее время именно ОС Android является самой популярной мобильной операционной системой. Судите сами: если зайти в каталог, например, того же поисковика Яндекс.Маркет, где представлены смартфоны ведущих производителей (Samsung, Sony, HTC, Xiaomi и др.), то фильтр выбора операционной системы предоставляет лишь два варианта: Android и iOS (а это только Apple). Понятно, что Android-смартфонов всех производителей гораздо больше, чем актуальных моделей «айфонов».

Так что теперь разработчику достаточно написать для ОС Android всего лишь одно приложение, и оно будет работать на всех моделях смартфонов, где она установлена. Во всяком случае, почти на всех. В настоящее время самая «древняя» версия Android, устройство с которой можно купить новым, — это 5.0, ну, с натяжкой, 4.4. Да, на том же Яндекс.Маркет можно найти модели 2014 года, которые поставлялись тогда именно с этими версиями ОС. Остальные предшествующие ее версии, учитывая средний срок службы смартфона (3–4 года, хотя многие пользователи меняют их чаще), можно считать неактуальными. Впрочем, при разработке приложения в 2019 году я бы ориентировался хотя бы на версию 5.0, как на минимальную. В своем желании обеспечить работу приложения на старых версиях Android будьте осторожны — не забывайте, что оно должно работать и на более новых версиях. Будет очень неприятно, когда приложение работает хорошо на 5.0, но «глючит» на той же 8.1.

Читателям книги «Программирование для Android 5. Самоучитель»

Прежде всего, разрешите выразить вам благодарность за то, что в свое время вы купили мою книгу «Программирование для Android 5. Самоучитель». А эту книгу, посвященную новейшей, 9-й версии Android (9.0 Pie, она же Android P), вы можете рассматривать как следующее ее издание.

Что нового вы здесь найдете? За прошедшее время изменилось многое: средства разработки приложений, эмуляторы, переработан API. По сути, если сравнивать разработку Android-приложения в 2011 году и в 2019, то это два разных процесса. Описывать здесь, что и в какой главе по отношению к предыдущему изданию изменилось, я не стану — вы и сами скоро все увидите.

Как читать эту книгу?

Книга разделена на четыре части. В *первой* мы поговорим об установке необходимого программного обеспечения, напишем самое простое Android-приложение и научимся использовать эмулятор Android, позволяющий создавать приложения для Android без наличия физического устройства, что снижает стоимость разработки приложения.

Во *второй* части вы узнаете, как создать для вашего приложения интерфейс пользователя, как использовать в нем графику и звуки, рассмотрим мы также и способы хранения данных.

Третья часть книги — самая сложная, поэтому даже не пытайтесь ее читать, не прочитав предварительно вторую. Из нее вы узнаете, как создавать службы, потоки, широковещательные приемники, как использовать SQLite и подключаться к Интернету.

В *четвертой* части будут рассмотрены среда быстрой разработки приложений App Inventor, отладка приложений и подготовка вашего приложения для публикации на Google Play. В качестве бонуса вам будет рассказано о весьма популярном эмуляторе Genymotion.

Сейчас же мы познакомимся с операционной системой Android, разберемся, как называются разные ее версии, а также поговорим о выборе Android-устройства для тестирования ваших приложений.

Вкратце об Android

Если вы держите в руках эту книгу, то, скорее всего, уже знаете, что такое Android. Да, Android — это операционная система для мобильных телефонов и других мобильных устройств (уже давно популярны планшеты на базе этой системы, существуют цифровые проигрыватели и даже нетбуки с Android на борту).

Изначально операционная система разрабатывалась компанией Android Inc., но впоследствии ее купила корпорация Google. Затем Google создала Open Handset Alliance — организацию, которая занимается поддержкой и развитием этой платформы.

Разработка приложений для Android осуществляется, как правило, на языке Java, но вы можете использовать и другие языки программирования, — об этом чуть позже (в *главе 2*).

Первая версия Android появилась в сентябре 2008 года. С тех пор вышло двадцать версий системы. Самая последняя версия на момент подготовки этой книги —

девятая. И раз мы уж заговорили о версиях Android, то вы должны о них кое-что узнать, прежде чем начнете программировать для Android.

Первые семь версий (с 1.0 по 2.3) были ориентированы только на смартфоны, а версии 3.*x* Google планировала использовать для планшетов. Поэтому в 2011–2012 годах считалось, что если вы хотите разрабатывать приложение для смартфонов, то должны ориентироваться на версии 1–2.*x*, а если для планшетов — то на версии 3.*x*. Но такое разделение не способствовало бы популярности самой Android. Вспомните, одно из преимуществ Android — возможность разработки единой версии приложения, способного работать на множестве устройств, без необходимости разрабатывать отдельные приложения для устройств каждого производителя. Если бы Google и дальше продолжала вести себя в том же духе, разработчикам пришлось бы создавать как минимум две версии приложения: для смартфонов и для планшетов, что стало бы отступлением от изначально принятой концепции.

К счастью, длилось это безобразие недолго, и уже осенью 2011 года была представлена версия 4.0, которая стала универсальной и работала как на смартфонах, так и на планшетах. Впрочем, справедливости ради нужно отметить, что экраны смартфонов к тому времени несколько подросли и стали достигать размера 4–5 дюймов по диагонали. Раннее разделение Android на две ветки было связано именно с разницей в размерах экранов смартфонов и планшетов и, соответственно, с различной разметкой приложений для них.

Версия 4.*x* стала самой популярной, поскольку, появившись в 2011 году, она «продержалась» на рынке ровно три года — до октября 2014 года, когда вышла версия Android 5. Одной из самых удачных была версия 4.4 — новые устройства на ней (понятно, что это залежавшиеся на складах модели 2014–2015 годов) можно купить до сих пор, также ее часто можно встретить на всевозможных китайских автомагнитолах, которые стали популярными в последнее время. Хотя в новых продуктах производители таких устройств уже переходят на версию. 8.*x*.

В табл. В.1 приведены даты выхода основных версий Android и их кодовые названия.

Таблица В.1. *Основные версии Android*

Версия	Дата выхода	Кодовое название
1.0	21 октября 2008 года	Applebread
1.1	9 февраля 2009 года	Bender
1.5	30 апреля 2009 года	Cupcake
1.6	15 сентября 2009 года	Donut
2.0/2.1	26 октября 2009 года	Éclair
2.2	20 мая 2010 года	Froyo
2.3	7 декабря 2010 года	Gingerbread
3.0	3 февраля 2011 года	Honeycomb

Таблица В.1 (окончание)

Версия	Дата выхода	Кодовое название
3.1	10 мая 2011 года	Honeycomb
3.2	15 июля 2011 года	Honeycomb
4.0	19 октября 2011 года	Ice Cream Sandwich
4.1	27 июня 2012 года	Jelly Bean
4.2	29 октября 2012 года	Jelly Bean
4.3	25 июня 2013 года	Jelly Bean
4.4	31 октября 2013 года	KitKat
5.0	16 октября 2014 года	Lollipop
5.1.*x*	9 марта 2015 года	Lollipop
6.0	5 октября 2015 года	Marshmallow
7.0–7.1.2	22 августа 2016 года	Nougat
8.0–8.1	21 августа 2017 года	Oreo
9	8 мая 2018 года	Android P

Как уже отмечалось ранее, в настоящее время самой старой версией Android, с которой можно еще «столкнуться» на витринах магазинов, остается версия 5.*x*. Но более популярной все же является версия 6.0 — именно эту версию можно с большей вероятностью встретить на всевозможных бюджетных устройствах. Нужно понимать, что чем выше версия Android, тем более требовательная она к ресурсам. Если для версии 4.4 вполне хватало 512 Мбайт оперативной памяти, то современные версии на них «задохнутся», поэтому более новые версии Android чаще можно встретить на более дорогих устройствах (хотя бы среднего ценового диапазона). Впрочем, это не мешает китайским производителям вроде Prestigio и Fly продавать устройства с 512 Мбайт «оперативки» и Android 6.0 (или с 1 Гбайт памяти и Android 8.1). Как будут работать эти устройства, можете сами догадаться.

Если в 2015 году самой популярной (62% по данным Google Play Маркет) версией была 4.4, то сейчас ситуация несколько иная (табл. В.2).

***Таблица В.2.** Доля разных версий Android*

Версия	Начало 2018 года	Май 2019 года
9.0	еще не вышла	10,4%
8.1	0,7%	15,4%
8.0		12,9%
7.1	26,3%	7,8%
7.0		11,4%

Таблица В.2 (окончание)

Версия	Начало 2018 года	Май 2019 года
6.0	29,7%	16,9%
5.1	25,1%	11,5%
5.0		3,0%
4.4	12,8%	6,9%
4.3	5,6%	0,5%
4.2		1,5%
4.1		1,2%
4.0	0,5%	0,3%
2.3	0,4%	0,3%

В 2018 году новые версии (8 и 9) практически не использовались — и это вполне понятно. Девятая версия пока не вышла, а производители устройств еще не успели продать модели с более старыми версиями, поэтому не торопились с новинками на базе версии 8.

В целом по данным табл. В.2 прослеживается тенденция к обновлению. Если в 2018 году устройств с версией 4,4 было 12,8% (это вам не шутки!), то в 2019-м таких устройств уже 6,9%. Очевидно, что за год пользователи обновили свои смартфоны и перешли на более новые версии.

Также здесь можно увидеть, какие версии были более популярны, какие — менее. Среди версий 4.*x* самой популярной является версия 4.4, а версия 5.0, как видно из таблицы, спросом не пользуется. Весьма удачной оказалась версия 6.0 — в процентном соотношении она немного уступает более свежей версии 7.*x*. У автора этой книги один из смартфонов до сих пор успешно работает под управлением Android 6.0 (Samsung J2 Prime, модель 2016 года).

Выбор Android-устройства

Систему Android можно встретить на самых разных мобильных устройствах: мобильных телефонах (смартфонах), планшетах, цифровых проигрывателях, нетбуках. Впрочем, нетбуки с ОС Android на борту — явление достаточно редкое. Первым нетбуком с Android, добравшимся до российского рынка, был AC100 от Toshiba. На этом нетбуке установлена только одна ОС — Android. До него были и другие нетбуки — так, на некоторых моделях нетбуков от Acer Android устанавливалась как вторая система при первой Windows. Пользователи могли ознакомиться с Android, попробовать, что это такое, но, как правило, все работали в Windows. Такая мультисистемность на нетбуках от Acer объяснялась просто — компания не желала рисковать, ведь нетбуки с непривычной системой могли плохо продаваться. Компания Toshiba рискнула — выпустила нетбук с Android в качестве единствен-

ной системы. Могу сказать только одно: пока AC100 и подобные нетбуки, к сожалению, не очень распространены... Хотя сам нетбук получился очень ничего, но покупатели, в отличие от менеджеров Toshiba, рисковать не желают, поэтому предпочитают покупать устройства с проверенной ОС. Их тоже можно понять — многие покупают нетбук не просто как приятный гаджет, а для работы, каждый привык к своим любимым... Windows-приложениям. Вот этим и объясняется низкая популярность нетбуков с Android.

> ***Первый Android-нетбук***
>
> Прочитать об AC100 можно по адресу:
> **http://hi-tech.mail.ru/review/misc/Toshiba_AC100-rev.html**.

Смартфоны и планшеты часто покупаются как приятные гаджеты. В большинстве случаев пользователям все равно, какая операционная система установлена на их смартфоне или планшете. Многие сначала обращают внимание на набор функций устройства, а уже затем на установленную операционную систему. Поэтому пользователи, заинтересовавшись обширным набором возможностей, сами того не ведая, покупают устройство с Android на борту, и только потом начинается знакомство с системой, попытка установки приложений и т. д.

Первым устройством, работающим под управлением Android, стал смартфон HTC T-Mobile G1, презентация которого состоялась 23 сентября 2008 года. Этот смартфон оказался настолько удачен, что вскоре другие производители мобильных телефонов заявили о намерении выпускать устройства с этой системой.

Что влияет на стоимость устройства? В первую очередь — бренд. Устройства производства Prestigio и других «непристижных» брендов будут стоить наверняка дешевле той же Samsung. Впрочем, если раньше самым дорогим смартфоном считался iPhone (если не считать некоторых экзотических брендов вроде Vertu), то сейчас ситуация меняется — некоторые модели Samsung и Huawei стоят дороже средних моделей iPhone.

Во-вторых, на цену того или иного устройства влияет набор его характеристик как мобильного устройства: размер и качество экрана, наличие Wi-Fi, объем встроенной памяти, производительность процессора, характеристики встроенных камер (чем они лучше, тем дороже телефон).

Сама операционная система Android, по сути, на стоимость устройства не влияет. Скорее всего, наоборот — характеристики устройства влияют на установленную версию Android. Так, выбор версии операционной системы зависит в основном от предпочтений производителя и характеристик устройства. Понятно, что чем новее версия Android, тем бóльшие требования к системе предъявляются. Поэтому новые версии на совсем уж слабые устройства не установишь. Кстати, на заметку — абсолютный минимум памяти для Android (согласно спецификации **https://source.android.com/compatibility/8.0/android-8.0-cdd.pdf** — см. стр. 34 этого документа) — 32 Мбайт. Но это только для устройств с небольшой плотностью пикселов — вроде умных часов (Android Watch). Да, размер оперативной памяти напрямую зависит от размера экрана и плотности пикселов. Например, для больших экранов

с плотностью 640 dpi нужно минимум 768 Мбайт оперативной памяти. Поэтому устройства с большим размером экрана никогда не бывают дешевыми, ведь кроме большего по размеру экрана приходится подтягивать и другие характеристики — как минимум, установить больше «оперативки».

В предыдущей своей книге «Программирование для Android 5. Самоучитель» я приводил основные технические характеристики многих моделей, которые были в то время актуальными на рынке. Однако всевозможных моделей стало сейчас столь много, что описывать в книге все имеющиеся просто нет смысла. Пока книга выйдет из типографии, появится десяток новых моделей.

При выборе Android-устройства (независимо от его типа: планшет или смартфон) нужно обратить внимание на следующие основные характеристики: тип процессора, объем оперативной памяти, размер и тип экрана и тип видеоускорителя.

Объем встроенной памяти обычно значения не имеет, т. к. почти во всех моделях предусмотрена возможность установить карту памяти microSD объемом 32–128 Гбайт). Имеет значение объем встроенной памяти лишь в той ситуации, если устройство не оснащено слотом для карты microSD. Дело в том, что сейчас наметилась тенденция в угоду все меньшей толщине устройства не устанавливать в него слот для внешней карты памяти. Некоторые производители по примеру Apple также лишили пользователей возможности расширения памяти своего устройства. Что это — или стремление к «премиальности», или побуждение пользователя обновить устройство, когда памяти станет не хватать, — решайте сами. Тем не менее тенденция эта не может не беспокоить, и, приобретая такие устройства, следует иметь в виду невозможность расширения их памяти.

Процессор

Общие сведения

Первым делом нужно обратить внимание на процессор. Тут все просто — чем мощнее процессор (выше его тактовая частота и больше ядер), тем быстрее будет работать устройство и меньше раздражать вас своей нерасторопностью. Однако есть и другая прямая зависимость — чем мощнее процессор, тем дороже устройство. Но обратной зависимости нет — наивно полагать, что чем дороже устройство, тем мощнее в нем процессор. В качестве примера можно привести три устройства: Sony Xperia XZ23 Dual Black, Xiaomi Mi 9, Meizu 16th. Первое устройство — это флагман от Sony, работающий под управлением Android 9, — чтобы вы не подумали, что я рассматриваю позапрошлогоднюю модель от именитого бренда. Это устройство оснащено процессором Snapdragon 845 и еще недавно стоило почти 70 тыс. рублей. Правда, сейчас его цена снизилась до около 45 тыс. рублей. Однако устройство от Xiaomi (которое появилось на полках магазинов буквально несколько дней назад — 16 мая 2019 года) оснащено даже более мощным процессором — Snapdragon 855 — и стоит 31 360 рублей безо всяких скидок. В качестве Xiaomi, думаю, сомневаться не стоит — бренд хоть и китайский, но на его примере китайцы показали, что умеют делать качественные вещи.

Смартфон от Meizu оснащен тем же процессором, что и Sony, но стоит еще дешевле — всего 26 тыс. рублей. Учитывая, что это флагман, остальные его характеристики тоже на уровне. А дальше вступает в роль маркетинг — кто-то предпочитает Sony или Samsung, кто-то — выберет Meizu из нежелания переплачивать, а кто-то вообще пришел за iPhone, и его никак не убедишь купить Android-смартфон. Здесь, как говорится, — на вкус и цвет все фломастеры разные ;-).

При выборе процессора обратите внимание на число ядер и рабочую частоту. Причем, если раньше частота была одинаковой для всех ядер, то сейчас ситуация несколько изменилась. Например, тот же Snapdragon 845 оснащен 8-ю ядрами, 4 из которых работают на частоте 2,8 ГГц, а еще 4 — на частоте 1,7 ГГц. У Snapdragon 855 тоже 8 ядер, вот только работают они иначе: одно на частоте 2,84 ГГц, три — на частоте 2,42 ГГц, а остальные 4 имеют частоту 1,8 ГГц.

Впрочем, все современные процессоры обеспечивают должный уровень производительности. Другое дело, когда вы подбираете не флагман, а относительно бюджетное устройство. Например, в том же Xiaomi Redmi 7 установлен Snapdragon 632 с четырьмя ядрами по 1,8 ГГц, а в Xiaomi Mi Play — MediaTek P35 с восемью ядрами: 4 работают на частоте 2,3 ГГц, а 4 — на 1,8 ГГц. Поэтому второе устройство должны быть пошустрее (да и ОЗУ в нем 4 Гбайт, а не 3), но при этом оно стоит дешевле. Так можно сэкономить и получить более шустрый аппарат, если потратить несколько минут и изучить характеристики.

И еще один момент, ставший актуальным в самое последнее время. Если вы хотите иметь возможность подключаться со своего устройства к скоростному мобильному Интернету 4-го поколения (4G) по технологии LTE, процессор вашего устройства должен такую возможность обеспечивать. К счастью, все современные модели поддерживают такую возможность, и если вы не собираетесь покупать устройство 2014 года выпуска, то можно об этом забыть.

> ***Примечание***
>
> Кстати, Apple вообще не заморачивается гонкой за ядрами. В процессоре того же флагмана iPhone XS (в том числе и в модели XS Max, которая стоит как автомобиль) — 6 ядер, два из которых работают на частоте 2,5 ГГц, а 4 вообще слабенькие — 1,6 ГГц.

Подробнее об ARM-процессорах

Если приведенное в предыдущем разделе объяснение вас устроило, можете смело переходить к следующему разделу, а самым любопытным читателям предлагаю «копнуть» глубже. Без понимания сути вы можете судить о производительности процессора только по его косвенным признакам — например, по рабочей частоте. Но это не совсем правильно.

Итак, процессоры всех Android-устройств, будь то смартфон, будь то планшет, основаны на архитектуре ARM (Advanced RISC Machine). Вы будете поражены, но это более «продвинутая» архитектура, чем архитектура CISC (Complex Instruction Set Computer), на базе которой создано большинство процессоров привычных нам компьютеров. Разница между этими архитектурами в наборе команд и в самом

подходе к построению этих команд. У архитектуры CISC более сложные команды. У архитектуры RISC (Reduced Instruction Set Computing) команды более простые. И то, что в CISC делается одной командой, в RISC делается несколькими. CISC более похожа на мышление человека, а RISC — на «мысли» компьютера. Приведу пример. Вы просите кого-то включить свет. Это пример CISC-команды: «включи свет». Чтобы добиться того же результата в RISC, нужен набор команд: «встань, подойди к выключателю, включи его». В общем, что-то вроде этого. Конечно, я сильно все упростил, но и чрезмерно усложнять рассуждения нам здесь тоже не с руки.

Далее мы рассмотрим популярные ARM-процессоры, которые устанавливаются на современные смартфоны и планшеты.

Архитектура ARM Cortex-A5

Это архитектура процессоров, используемых в устройствах начального уровня. Архитектура достаточно старая, и если сейчас вы видите в продаже устройство на базе Cortex-A5, то перед вами или очень устаревшая модель, или же самая дешевая модель китайского производителя вроде Prestigio. Я не рекомендую покупать устройство на базе этой архитектуры, хотя она совместима с более современными Cortex-A8/A9, что позволяет запускать на таких устройствах современные версии Android.

Архитектура ARM Cortex-A7/Cortex-A8

На бюджетные модели устанавливается одноядерный процессор ARM Cortex-A8 или его модификации вроде Boxchip A13. Рабочая частота Cortex-A8 — от 600 МГц до 1 ГГц (хотя модификации типа A13 могут работать на более высоких частотах — до 1,2 ГГц).

Интересно, что Cortex-A7 лучше по своим характеристикам, чем Cortex-A8, несмотря на «семерку» в наименовании архитектуры. При этом Cortex-A8 менее распространена, чем Cortex-A7, поэтому в большинстве случаев вы встретите модели гаджетов на базе Cortext-A7. Здесь рабочая частота может варьироваться от 600 МГц до 3 ГГц. Однако, как правило, частота современных процессоров на базе Cortex-A7 находится в пределах 1 ГГц. Серьезным преимуществом этой архитектуры является поддержка многоядерных конфигураций. Теоретически на базе этой архитектуры можно построить 8-ядерный процессор, т. к. допускается два кластера на кристалл, а в одном кластере может быть от одного до четырех ядер. Но таких «монстров» на базе Cortex-A7 я не встречал, поскольку производители предпочитают использовать более совершенную архитектуру: Cortex-A9.

Архитектура ARM Cortex-A9

Процессоры семейства на базе архитектуры ARM Cortex-A9 гораздо мощнее своих предшественников. Они, как правило, многоядерные — могут содержать до четырех ядер. Двухъядерная версия Cortex-A9 обычно работает на частоте 1,2 ГГц. Максимальная частота для Cortex-A9 — до 3 ГГц, минимальная — 800 МГц.

Некоторые планшеты поставляются с процессорами Rockchip 29*xx*. Эти процессоры — модифицированная версия Cortex-A8, в то время как Rockchip 3*xxx* — это модифицированная версия Cortex-A9. Ясно, что лучше предпочесть Rockchip 3*xxx*.

Процессор Amlogic 8726-M6 — это модифицированный двухъядерный Cortex-A9.

Есть еще она «темная» лошадка — процессоры MediaTek, в частности MTK8317T, который устанавливается на весьма популярный (судя по продажам) планшет Acer Iconia Tab. На базе процессоров тайваньской фирмы MediaTek построено множество мобильных устройств (о линейке этих процессоров вы можете подробно прочитать здесь: **http://ru.wikipedia.org/wiki/MediaTek**). Так вот, процессор MTK8317T является ничем иным, как Cortex-A9. Процессор двухъядерный, рабочая частота — 1,2 ГГц. То есть вполне нормальный процессор на сегодняшний день.

И еще один процессор, который следует упомянуть, — Action ATM7029. Это четырехъядерный (!) процессор на архитектуре ARM Cortex-A9 с частотой до 1,5 ГГц.

Архитектура Cortex-A15

Процессоры на базе Cortex-A15 могут содержать до четырех ядер, а рабочая частота может варьироваться от 800 МГц до 3 ГГц. Если судить по количеству ядер и частоте, то эта архитектура недалеко ушла от Cortex-A9, но вся суть в вычислительном конвейере, позволяющем выполнять 3 команды за такт, в то время как аналогичный конвейер в Cortex-A9 выполняет только две команды за такт. Ядро Cortex-A15 обрабатывает до восьми микроопераций за такт против четырех микроопераций у Cortex-A9. Это означает, что при одинаковом количестве ядер и одинаковой частоте процессор на Cortex-A15 будет работать в среднем в два раза быстрее, чем на Cortex-A9. Да и кэш второго уровня (L2) здесь 8 Мбайт, а не 4 Мбайт, как у Cortex-A9. Размер кэша первого уровня (L1) у этих двух архитектур одинаковый (32 Кбайт), но у A15 используется 128-битная шина кэша, а у A9 — только 64-битная.

Архитектура Cortex A-17

Архитектура Cortex A-17 предусматривает применение 28-нанометровой технологии. В 2014 году это было круто. Сейчас же, на фоне 7-нанометровых технологий, она выглядит настоящим динозавром.

В смартфоны и планшеты устанавливались процессоры, объединяющие два или четыре ядра Cortex-A17 и два или четыре ядра Cortex-A7. То есть топовые изделия получили до восьми вычислительных ядер. Тактовая частота варьировалась от 1 до 2 ГГц (и выше).

Архитектура Cortex A-32

Последняя архитектура из семейства Cortex-A. Это семейство является 32-разрядным, поэтому все архитектуры (в том числе и эта), рассмотренные ранее, являются 32-разрядными.

Впервые она применена в процессоре Samsung Exynos 5433, который устанавливался в Galaxy Note 4.

Примечание

Подробное сравнение архитектур и семейств ARM можно найти на странице Википедии **https://en.wikipedia.org/wiki/List_of_ARM_microarchitectures**. Далее мы не будем рассматривать все возможные чипы, а ограничимся только некоторыми.

Архитектура Cortex A-53

В отличие от предшественников, это уже 64-разрядное решение. Хотя первым 64-разрядным решением в семействе Cortex-A была архитектура Cortex A-35. Однако она не получила такого распространения, как A-53.

Здесь уже все поинтереснее: 12-нанометровый процесс, возможность выполнения как 64-разрядного, так и 32-разрядного кода. Эта архитектура широко используется в процессорах Qualcomm Snapdragon, MediaTek, Samsung Exynos, HiSilicon Kirin, Allwinner, Broadcom, Freescale i.MX, Rockchip.

Архитектура Cortex A-55

В отличие от A-53, где было возможно применение только от одного до четырех ядер, в этом случае максимальное количество ядер — 8. Размер L1 кэша — 64 Кбайт (для A-53 этот размер был обычно 8 Кбайт, но в некоторых случаях — на усмотрение разработчика процессора — мог достигать 64 Кбайт). Аналогичная история и с кэшем L2 — на A-53 его размер мог быть равен от 128 Кбайт до 2 Мбайт, а на A-55 — это 256 Кбайт на каждое ядро. Другими словами, выбирая процессор на базе A-55, получаешь гарантированный уровень производительности. А вот с A-53 нужно смотреть характеристики конкретного процессора, т. к. сама архитектура не устанавливала жесткие рамки.

Архитектура Cortex A-73

В 2016 году компания ARM представила новый чип ARM Cortex A73, использующий графический чип Mali G71. Для уменьшения энергопотребления Cortex A73 может работать в паре несколькими Cortex A53 или A35, переходя для решения нетребовательных задач в режим энергосбережения. Другими словами, чтобы процессор не «скушал» всю батарейку за один раз, обычно используются несколько ядер A73 и несколько ядер A53. Например, Snapdragon 660 задействует четыре ядра A73 и четыре ядра A53.

Новый чип A73 обладает на 30% лучшей производительностью при 30%-ном снижении энергопотребления. В сравнении с A72 новый A73 работает на частоте в 2,8 ГГц против 2,5 ГГц соответственно.

Конек A73 — использование графики Mali G71. В среднем прирост производительности в играх составляет около 50% и при этом достигается 20% снижения энергопотребления.

Архитектура Cortex A-75

Эта архитектура представлена в 2017 году. По сравнению с A-73 достигается прирост производительности в среднем на 22%. Одним из наиболее существенных архитектурных изменений в ARM Cortex-A75 является возможность использования большего количества энергии. Максимальное энергопотребление увеличено до 2 Вт,

что позволяет повысить производительность на 30% для устройств с большими экранами.

Архитектура Cortex A-76

Это разработка 2018 года, поэтому на базе A-76 можно встретить только самые современные устройства. Новинка полагается на архитектуру Armv8-A (Harvard). При производстве чипов должна применяться 7-нанометровая технология. Тактовая частота может достигать 3 ГГц и более. По сравнению с A-75 производительность увеличена на 35%, энергоэффективность — на 40%.

Процессоры Snapdragon

Процессоры Qualcomm Snapdragon весьма популярны на рынке мобильных устройств. Если не брать во внимание чипы Exynos, которые устанавливаются только на гаджетах производства Samsung, можно сказать, что Snapdragon — самые популярные на рынке. Действительно, если посмотреть характеристики современных устройств, то большая часть из них будет основана на Snapdragon и лишь некоторые из них на базе процессоров MediaTek. А раз так, то нужно ориентироваться в модельной линейке процессоров от Qualcomm, которых в 2019 году насчитается полтора десятка (считаем только актуальные модели).

Табл. В.3 содержит основные характеристики процессоров Qualcomm Snapdragon.

Таблица В.3. *Основные характеристики процессоров Qualcomm Snapdragon*

Модель	Техпроцесс	Ядра	Графика	Память
429	12 нм	4 × A-53 до 1,95 ГГц	Adreno 504	LPDDR3, 1 канал
439	12 нм	8 × A-53: 4 × 1,95 Гц, 4 × 1,45 ГГц	Adreno 505	LPDDR3, 1 канал
450	14 нм	8 × A-53 до 1,8 ГГц	Adreno 506, 600 МГц	LPDDR3, 1 канал
625	14 нм	8 × A-53 до 2 ГГц	Adreno 506, 650 МГц	LPDDR3, 1 канал
626	14 нм	8 × A-53 до 2,2 ГГц	Adreno 506, 650 МГц	LPDDR3, 1 канал
630	14 нм	8 × A-53: 4 × 2,2 Гц, 4 × 1,8 ГГц	Adreno 508, 850 МГц	LPDDR4, 2 канала
632	14 нм	8 ядер, 4 × A-73 до 1,8 ГГц, 4 × A-53 до 1,6ГГц	Adreno 509	LPDDR3, 1 канал
660	14 нм	8 ядер, 4 × A-73 до 2,2 ГГц, 4 × A-53 до 1,84 ГГц	Adreno 512, 850 МГц	LPDDR4, 2 канала
670	10 нм	8 ядер, 4 × A-75 до 2 ГГц, 4 × A-55 до 1,7 ГГц	Adreno 615, 700 МГц	LPDDR4, 2 канала
675	11 нм	8 ядер, 4 × A-75 до 2 ГГц, 4 × A-55 до 1,7 ГГц	Adreno 612	LPDDR4, 2 канала
710	10 нм	8 ядер, 4 × A-75 до 2,2 ГГц, 4 × A-55 до 1,7 ГГц	Adreno 616, 750 МГц	LPDDR4, 2 канала
712	10 нм	8 ядер, 4 × A-75 до 2,3 ГГц, 4 × A-55 до 1,7 ГГц	Adreno 616, 750 МГц	LPDDR4, 2 канала

Таблица В.3 (окончание)

Модель	Техпроцесс	Ядра	Графика	Память
845	10 нм	8 ядер, 4 × A-75 до 2,8 ГГц, 4 × A-55 до 1,8 ГГц	Adreno 630, 710 МГц	LPDDR4, 4 канала
855	7 нм	8 ядер, 4 × A-76 до 2,85 ГГц, 4 × A-55 до 1,8 ГГц	Adreno 640	LPDDR4, 4 канала

Процессоры Samsung Exynos

Процессоры Exynos — это собственная разработка Samsung на базе архитектур ARM Cortex (табл. В.4). Как правило, смартфоны и планшеты Samsung поставляются на базе этого процессора, хотя некоторые модели комплектуются процессорами Snapdragon (например, новейший Samsung Galaxy A70, представленный в 2019 году, оснащен процессором Snapdragon 675).

***Таблица В.4.** Процессоры Samsung Exynos*

Процессор	К-во ядер	Архитектура	Частота, ГГц
Exynos 3	1	Cortex-A8	0,8–1,2
Exynos 4	2 или 4	Cortex-A9	1,2–1,6
Exynos 5	2 или 4	Cortex-A15	до 1,7
Exynos 7	2 или 4	Cortex A-53/A-57	до 2,3
Exynos 8/9	4 или 8	Cortex A-53/A-55/A-75	до 2,9

Выводы

Теперь давайте подытожим. От 32-разрядных процессоров Cortex рекомендую отказаться — они слабые. Остальные же можно выбирать по своему усмотрению и финансовым возможностям. Конечно, если вы планируете заняться разработкой игр, то лучше смотреть на процессоры подороже — они не только мощнее, но и оснащены более совершенным видеоускорителем.

Память

Оперативная память используется для хранения данных, обрабатываемых в текущий момент. Android не очень требовательная к «оперативке» система, просто в последнее время сами данные слишком объемны. У смартфонов и планшетов начального уровня объем оперативной памяти равен 1 Гбайт. В 2019 году минимальный размер оперативной памяти — если вы хотите, чтобы смартфон был именно смартфоном, а не просто «звонилкой», — 2 Гбайт. Более совершенные модели поставляются с тремя и более гигабайтами «оперативки».

Так что, учитывая, что в большинстве случаев вы не можете нарастить оперативную память смартфона/планшета (это не ноутбук, где иногда можно легко и быстро

расширить ОЗУ), и что обрабатываемые данные меньше уже не станут, я бы в любом случае присмотрелся к моделям, где установлено не менее 3 Гбайт ОЗУ.

С объемом встроенной памяти все ясно и просто — чем ее больше, тем больше данных вы сможете хранить до покупки карты microSD. В большинстве случаев Android-устройства поддерживают карты microSD большого размера (до 128 Гбайт). В общем-то, особой разницы нет, сколько встроенной памяти установлено в выбранной вами модели смартфона/планшета, — всегда можно купить карту microSD (разумеется, если устройство имеет слот для ее установки, на наличие которого тоже следует обращать повышенное внимание). У бюджетных моделей смартфонов обычно 16 Гбайт встроенной памяти, у моделей подороже — 32 Гбайт, а у самых дорогих моделей — 64 и более гигабайт.

Дисплей

Размер дисплея приобретаемого устройства зависит от личных предпочтений. Мне вот не сильно нравятся модели с очень большим (5 дюймов и более) дисплеем, но на вкус и цвет, как говорится... Зато тип экрана играет большую роль:

- TFT — общее название дисплеев, обычно под ним скрывается TN-матрица: углы обзора и цветопередача будут не на высоте;
- IPS — еще недавно этот тип дисплея считался слишком дорогим, а сейчас устанавливается уже и не в самые «продвинутые» устройства, что не может не радовать. Отличная цветопередача, глубокий цвет, широкие углы обзора. TN/TFT-матрицу я не могу рекомендовать, а вот IPS — смело можете покупать (или ее усовершенствованную модификацию — PLS);
- SuperLCD — разработка Sony, они потребляют меньше энергии, чем обычные LCD, и также отображают яркую и четкую картинку;
- AMOLED — фирменная светодиодная технология Samsung. Такие экраны считаются одними из лучших, но сейчас пока дорогие и сложные в производстве. Используются только на лучших смартфонах Samsung — начиная с Galaxy S III;
- NOVA — основана на технологии IPS, разработка LG. Обеспечивает высокую яркость и экономию заряда батареи.

Видеоускоритель

При выборе устройства следует обратить внимание не только на процессор, но и на видеоускоритель, о чем упоминалось ранее. Понятное дело, что чем мощнее ускоритель, тем лучшие игры можно запустить на устройстве, где он установлен. Есть и такая зависимость — чем лучше процессор устройства, тем лучше видеоускоритель. Поэтому, даже если в характеристиках устройства не указано, какой ускоритель в нем установлен, о видеоускорителе можно судить по установленному процессору. Например, ускоритель Adreno 200 не может быть установлен с современными процессорами. Когда-то лучшей связкой считалось устройство на базе процессоров MSM7230/MSM8255 и ускорителя Adreno 205. Но это было достаточно давно. Данные табл. В.3 помогут вам определить, какой видеоускоритель уста-

новлен в том или ином устройстве. Сейчас (если вы планируете разрабатывать игры) нужно присмотреться к Adreno 6*xx*, которые поставляются с современными и весьма неслабыми процессорами Snapdragon.

Платформа Snapdragon является наиболее сбалансированной. Устройства, построенные на этой платформе, можно одинаково эффективно использовать как для работы, так и для развлечений.

Впрочем, ускорители Adreno уступают ускорителям PowerVR, но это уже другая песня. Устройства на ускорителях PowerVR строятся больше для игр, нежели для работы. И если поиграть получится, то для работы такие устройства покупать не рационально.

Устройства с видеоускорителем PowerVR не так многочисленны на рынке, как устройства с ускорителем Adreno, и с полным списком таких устройств можно ознакомиться на этой страничке: (здесь приводятся не только мобильные телефоны): **http://en.wikipedia.org/wiki/List_of_PowerVR_products**. То, что ускорители PowerVR установлены на iPhone и iPod Touch Apple, говорит о многом. Продукция от Apple всегда славилась своей графикой. И отменное качество гарантирует здесь именно видеоускоритель PowerVR.

Высокое качество графики видеоускорителей PowerVR послужило предпосылкой для появления их и на некоторых Android-устройствах. Поскольку видеоускорители на этих устройствах и на iPhone одинаковые, то многие игры, ранее написанные для iOS, были портированы на Android. Одним словом, если у вас раньше был iPhone, и вы играли на нем в любимую игрушку, теперь она доступна и на платформе Android.

Но и ускорители PowerVR — это далеко не предел! Все было бы хорошо и спокойно в мире Android, все бы пользовались ускорителями Adreno и PowerVR, но это спокойствие было нарушено компанией NVIDIA. Думаю, не нужно объяснять, что это за компания и на чем она специализируется. Ее платформа NVIDIA Tegra 4 буквально взорвала мир игр Android. Графика, которую способна передавать NVIDIA Tegra 4, сопоставима с графикой персональных компьютеров, только на экране небольшого смартфона. Подробно об этой платформе можно узнать по этой ссылке: **http://www.nvidia.ru/object/tegra-ru.html**.

Говорить о платформе NVIDIA Tegra можно очень долго, но пока лучше этой платформы никто ничего не придумал. Чтобы просмотреть, на что она способна, посетите страничку на YouTube: **http://www.youtube.com/watch?v=YhA0cbu1BxI**.

Теперь подытожим. Что же выбрать из всего этого многообразия? Относительное дешевое решение на базе Adreno, более дорогое устройство с PowerVR или же перейти в высшую лигу и купить устройство на базе Tegra?

Все зависит от того, графику какого класса вы намереваетесь создавать. Для сложных игр лучше подойдет Tegra или PowerVR. А для несложной графики вполне будет достаточно и Adreno.

Архитектура Android

Архитектура Android состоит из четырех уровней: уровня ядра, уровня библиотек и среды выполнения, уровня каркаса приложений (application framework) и уровня приложений. Начнем с ядра.

Система Android основана на ядре Linux версии 2.6. Тем не менее Android не является Linux-системой в прямом смысле этого слова. У Android свои механизмы распределения памяти, другая система межпроцессного взаимодействия (Inter-Process Communication, IPC), специфические модули ядра и т. п. На уровне ядра происходит управление аппаратными средствами мобильного устройства. На этом уровне работают драйверы дисплея, камеры, клавиатуры, Wi-Fi, аудиодрайверы. Особое место занимают драйверы управления питанием и драйвер межпроцессного взаимодействия (IPC).

Уровень ядра — самый низкий уровень архитектуры Android. Следующий уровень — это уровень библиотек и среды выполнения. Он представлен библиотеками Bionic (в Linux она называется glibc), OpenGL (поддержка графики), WebKit (движок для отображения веб-страниц), FreeType (поддержка шрифтов), SSL (зашифрованные соединения), SGL (2D-графика), библиотекой поддержки SQLite, библиотекой Media Framework (нужна для поддержки мультимедиа).

Как можно понять, разработчики Android создали собственную версию библиотеки glibc — Bionic. Дело в том, что эта библиотека загружается в каждый процесс, а стандартная библиотека glibc просто огромна по меркам мобильных устройств, поэтому было принято решение ее переписать и сделать более компактной. Конечно, пришлось кое-чем пожертвовать: Bionic не поддерживает исключения C++ и не совместима с GNU libc и POSIX.

На этом же уровне работает DVM (Dalvik Virtual Machine) — виртуальная машина Java, предоставляющая необходимую функциональность для Java-приложений.

Следующий уровень — уровень каркаса приложений. На этом уровне работают различные диспетчеры:

- диспетчер активности (Activity Manager) — управляет жизненным циклом приложения;
- диспетчер пакетов (Package Manager) — управляет установкой пакетов прикладных программ;
- диспетчер окон (Window Manager) — управляет окнами приложений;
- диспетчер ресурсов (Resource Manager) — используется для доступа к строковым, графическим и другим типам ресурсов;
- контент-провайдеры (Content Providers) — службы, предоставляющие приложениям доступ к данным других приложений;
- диспетчер телефонии (Telephony Manager) — предоставляет API, с помощью которого можно контролировать основную телефонную информацию: статус подключения, тип сети и т. д.;

- диспетчер местоположения (Location Manager) — позволяет приложениям получать информацию о текущем местоположении устройства;
- диспетчер уведомлений (Notification Manager) — позволяет приложению отображать уведомления в строке состояния;
- система представлений (View System) — служит для создания внешнего вида приложения (позволяет организовать кнопки, списки, таблицы, поля ввода и другие элементы пользовательского интерфейса).

На уровне приложений работает большинство Android-приложений: браузер, календарь, почтовый клиент, навигационные карты и т. д. Нужно отметить, что Android не делает разницы между приложениями телефона и сторонними программами, поэтому любую стандартную программу можно заменить альтернативной. При разработке приложений программист имеет полный доступ ко всем функциям операционной системы, что позволяет полностью переделать систему под себя.

Google Play

Популярности платформе Android добавляет сервис Google Play (ранее Android Market, Play Маркет), представляющий собой онлайн-магазин приложений для платформы Android. Сервис Google Play был открыт 22 октября 2008 года. С помощью Google Play вы можете не только распространять свои приложения, но и зарабатывать, получая при этом 70% прибыли от ваших проданных приложений.

На мой взгляд, у Google Play есть единственный недостаток — доступ к этому сервису не одинаков для программистов из разных стран. Программисты из одних стран вообще не имеют право размещать на Google Play свои приложения, из других — могут размещать приложения бесплатно, из третьих — имеют право свои программы продавать. С другой стороны, бесплатное распространение программы тоже хорошо. Вы можете написать ограниченную версию своей программы и распространять ее бесплатно, а заинтересовавшимся продавать ее полнофункциональную версию.

Сервис Google Play будет рассмотрен в *главе 18* этой книги. Сейчас же самое время приступить к установке необходимого программного обеспечения.

ЧАСТЬ I

Подготовка к работе

ГЛАВА 1

Установка необходимого программного обеспечения

1.1. Что нужно для создания Android-приложения?

К сожалению, чтобы начать программировать для Android, недостаточно установить какую-то программу или набор программ. Нужно развернуть целую среду разработки и установить как минимум три основных компонента, а потом настроить их для совместного использования.

И прежде чем мы приступим к установке необходимого программного обеспечения, нужно сделать несколько замечаний:

- во-первых, в этой книге будет рассматриваться установка среды разработчика в операционной системе Windows. Ради справедливости нужно отметить, что среду разработки Android-приложений можно развернуть и в Linux (для этого есть все необходимое программное обеспечение, и работать оно будет точно так же), и в macOS (при этом вам понадобится версия 10.4.8 или более новая), но у большинства читателей этой книги, скорее всего, установлена Windows, поэтому мы сосредоточимся на работе в ее среде. Все приведенные в книге иллюстрации среды разработки будут соответствовать Windows 10;
- во-вторых, мы подразумеваем, что у вас есть навыки программирования на языке Java. Если таковых нет, то перед прочтением этой книги нужно изучить какую-либо книгу по Java, поскольку основы этого языка мы здесь не рассматриваем;
- в-третьих, написать Android-приложение можно и на языке C#. Компания Novell выпустила среду разработки Mono for Android. Благодаря этой среде, разработчики могут создавать приложения для операционной системы Android, используя C# и .NET, однако рассмотрение среды Mono выходит за рамки этой книги. Скачать же Mono for Android можно по адресу: **http://mono-android.net/**.

Итак, в этой книге рассматривается разработка приложений Android в среде Android Studio и на языке Java. При желании вы можете воспользоваться средой разработки Mono for Android, но тогда код программ из книги вам придется адаптировать самостоятельно.

Если раньше для разработки Android-приложений требовалось загрузить с разных сайтов и установить много различных компонентов, то сейчас нужно установить только JDK (Java Development Kit) и Android Studio — один пакет, содержащий в себе все необходимое для начала программирования.

С программным обеспечением мы разобрались. Теперь перейдем к «железу». Теоретически для разработки Android-приложений вам не нужен ни телефон, ни планшет, ни любое другое устройство, поскольку для отладки и запуска приложений будет использоваться эмулятор, входящий в состав Android Studio. Что касается версий Android, то, как было показано во *введении*, сейчас нужно ориентироваться на версии с 5-й по 9-ю. Возможно, вам еще захочется охватить и версию 4.4, но она, можно сказать, уже неактуальна.

Наличие физического устройства, хоть и не обязательно, но весьма желательно — для тестирования программы, так сказать, в «боевых» условиях. Эмулятор есть эмулятор, а реальное устройство может показать недочеты вашей программы, которые невозможно будет заметить в эмуляторе. К тому же не каждый эмулятор позволяет установить сервисы Google, что важно для проверки лицензии программы.

Так что понадобится устройство с поддержкой Android той версии, под которую вы планируете разрабатывать программы. Если же планируется разработка программ под разные версии Android, неплохо бы обзавестись несколькими устройствами — желательно, разных производителей — ведь везде есть свои нюансы, а чем «разношерстнее» оборудование, тем больше вероятность возникновения всякого рода непредвиденных обстоятельств, — то, что и нужно для процесса отладки программы. Понимаю, что все сказанное хорошо только на бумаге, а в реальной жизни — это лишние затраты. Одно дело, если вы работаете в компании, которая занимается (или планирует заниматься) разработкой для Android, — тогда все необходимые устройства будут куплены за счет компании. Другое дело, если вы желаете заняться разработкой самостоятельно — тогда приобретение нескольких Android-устройств разных версий может нанести ощутимый удар по домашнему бюджету. Но тут решать только вам — только вы знаете, сколько можете позволить себе потратить на покупку всевозможных гаджетов. Рассматривайте эти вложения как инвестицию в себя — ведь с помощью Google Play вы всегда сможете продать свои приложения и окупить вложения.

Можно пойти по другому пути. Если сейчас выкладывать большую сумму не хочется, можете использовать эмулятор и распространять свои программы бесплатно. Пользователи вашей программы не замедлят известить вас, если она будет работать неправильно. Когда же все возможные «глюки» будут исправлены, вы сможете распространять свою программу и на коммерческой основе. Однако если вы планируете серьезно заниматься разработкой для Android, рано или поздно на оборудование придется потратиться.

В этой книге мы будем работать со стандартным эмулятором, входящим в состав SDK Android, однако вы должны знать, что это не единственный доступный вариант. Можно также воспользоваться достаточно удобными эмуляторами Genymotion

(**https://www.genymotion.com/**) — он будет рассмотрен в *главе 18*, и Andy (**http://www.andyroid.net/**).

1.2. Установка комплекта разработчика Java-приложений — Java Development Kit (JDK)

Для разработки Java-программ понадобится комплект разработчика Java-приложений — Java Development Kit (JDK), включающий компилятор Java, стандартные библиотеки классов Java, документацию, примеры и среду выполнения Java — Java Runtime Environment (JRE), которая необходима для запуска программ, написанных на Java. Так что вам первым делом нужно скачать и установить Java Development Kit.

Скачать JDK можно по адресу:
http://www.oracle.com/technetwork/java/javase/downloads/index.html.

При загрузке обратите внимание на версию скачиваемого JDK — надо скачать версию, подходящую именно для вашей ОС. На момент подготовки этой книги доступна версия 12.0.1, поэтому пользователям 64-битной версии Windows следует скачать файл jdk-12.0.1_windows-x64_bin.exe.

После загрузки запустите загруженный файл (рис. 1.1). В процессе установки JDK нет ничего сложного — просто нажимайте кнопку **Next** и следуйте инструкциям мастера установки (рис. 1.2).

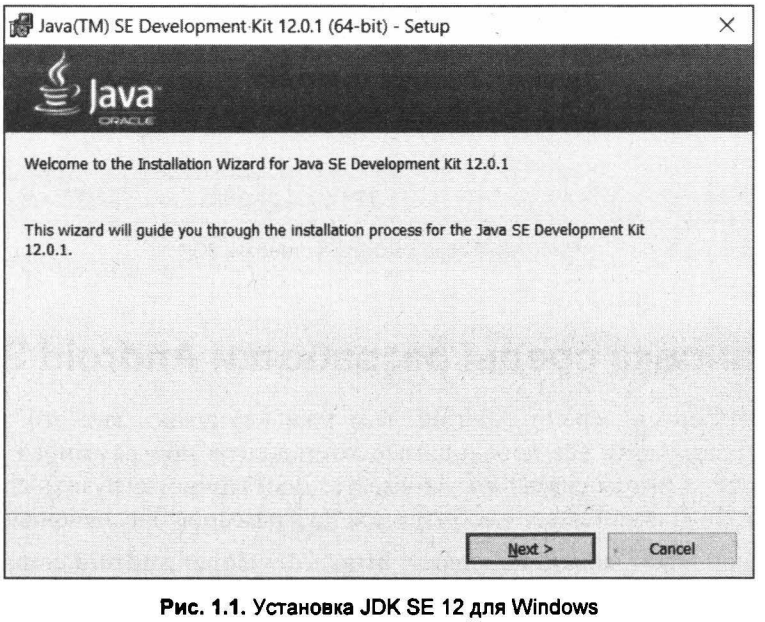


Рис. 1.1. Установка JDK SE 12 для Windows

Системные требования Android Studio

Минимальные системные требования Android Studio выглядят весьма скромно: 3 Гбайт RAM, 2 Гбайт на жестком диске, разрешение экрана 1280×800 пикселов. Но на практи-

ке этого очень и очень мало. Рекомендуемый размер оперативной памяти — 8 Гбайт (особенно, если вы будете производить отладку приложений в эмуляторе, а не на реальном устройстве), минимальный размер свободного дискового пространства — 4 Гбайт плюс место для ваших проектов.

Кстати, нигде не сказано о рекомендуемом типе накопителя. Я бы посоветовал устанавливать Android Studio только на твердотельный диск (SSD). Свои проекты вы можете хранить на жестком или даже на сетевом диске, а вот саму среду следует устанавливать только на SSD — тогда сборка проекта и работа эмулятора не покажутся утомительно медленными.

Теперь о процессоре... Для комфортной работы с Android Studio вполне достаточно процессора Intel Core i5 6-го или 7-го поколения. Другими словами, рекомендуемая конфигурация выглядит так: Intel Core i5, 8 Гбайт ОЗУ, 128 (или больше) Гбайт SSD (более подробно о требованиях к оборудованию рассказано в *разд. 2.10*).

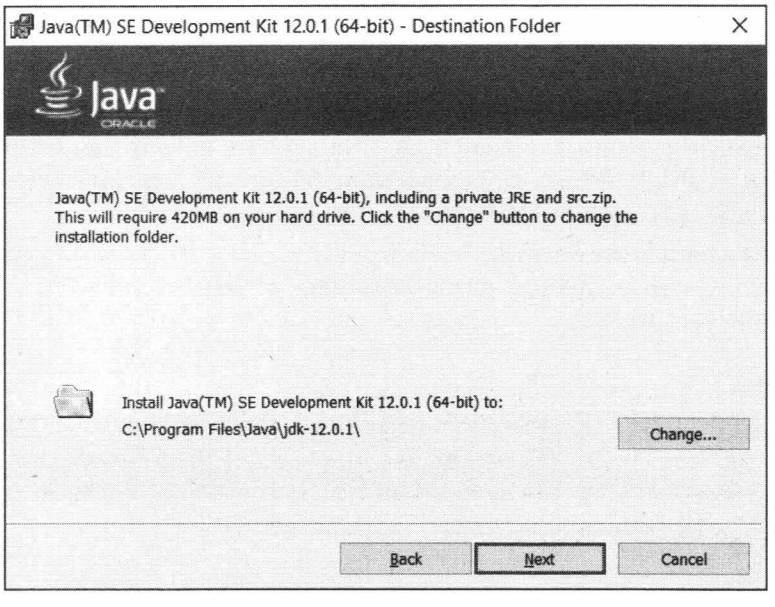


Рис. 1.2. Выбор каталога установки JDK

1.3. Установка среды разработки Android Studio

Начиная с четвертой версии Android, вам уже не нужно, как это приходилось делать ранее, загружать все необходимые компоненты программного обеспечения по отдельности, а потом связывать их вместе. Достаточно загрузить единый пакет, в составе которого имеется все необходимое программное обеспечение.

Загрузить такой пакет можно по адресу: **https://developer.android.com/studio/?hl=i**. Нажмите на открывшейся странице кнопку **Download Android Studio**. На следующей странице нужно согласиться с лицензионным соглашением и подтвердить загрузку.

Вы загрузите исполнимый файл android-studio-ide-<номер_версии>-windows.exe. Его размер действительно огромный — 971 Гбайт (на текущий момент), и с каждой

новой версией размер этого пакета становится все больше и больше. Для установки Android Studio сначала потребует 2,6 Гбайт (рис. 1.3), а затем вам придется (в процессе работы) доустановить дополнительные образы Android, поэтому сразу ориентируйтесь примерно на 8 Гбайт свободного дискового пространства. И да — установите Android Studio на самый быстрый диск (рис. 1.4).

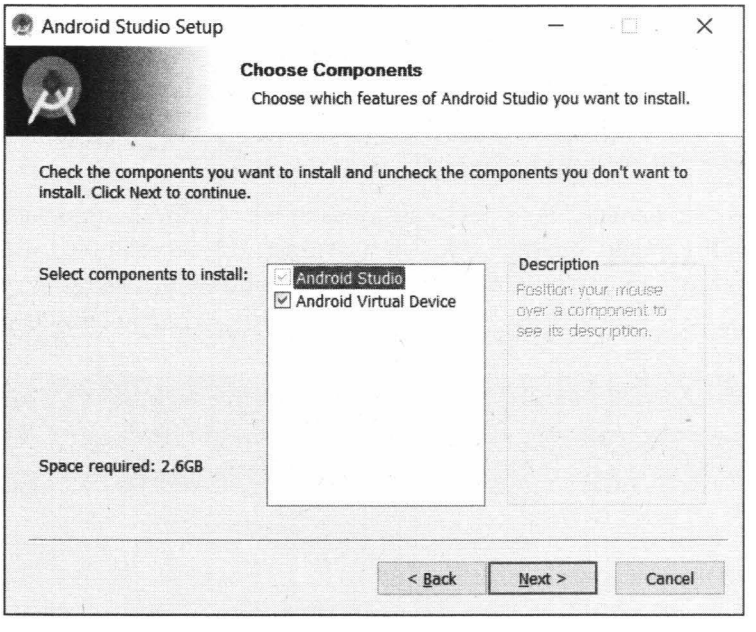


Рис. 1.3. Установка Android Studio

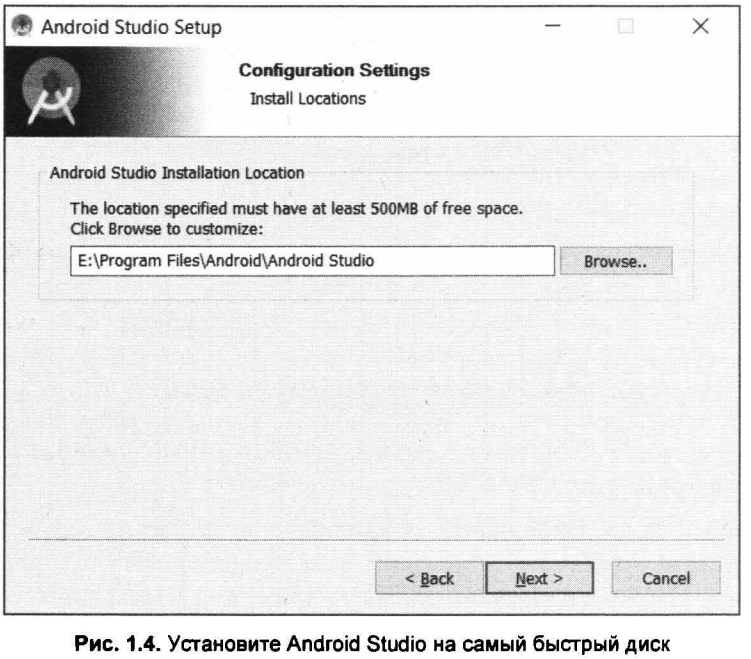


Рис. 1.4. Установите Android Studio на самый быстрый диск

По окончании установки инсталлятор предложит запустить Android Studio. Не отказывайтесь от этой возможности. При первом запуске вам будет предложено импортировать предыдущие настройки Android Studio, если таковые имеются (рис. 1.5).

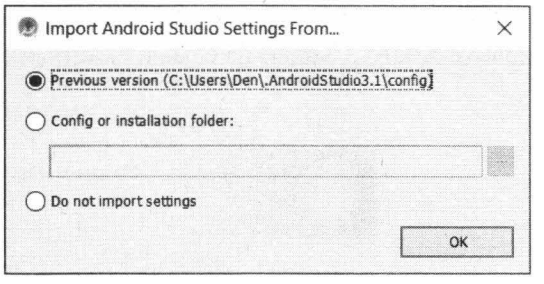


Рис. 1.5. Импортировать предыдущие настройки и домашнюю папку?

Если вы никогда раньше не программировали на Android, понятно, вам нужно выбрать второй вариант (точнее, он будет выбран по умолчанию) и нажать кнопку **OK** — запустится мастер настройки Android Studio (рис. 1.6).

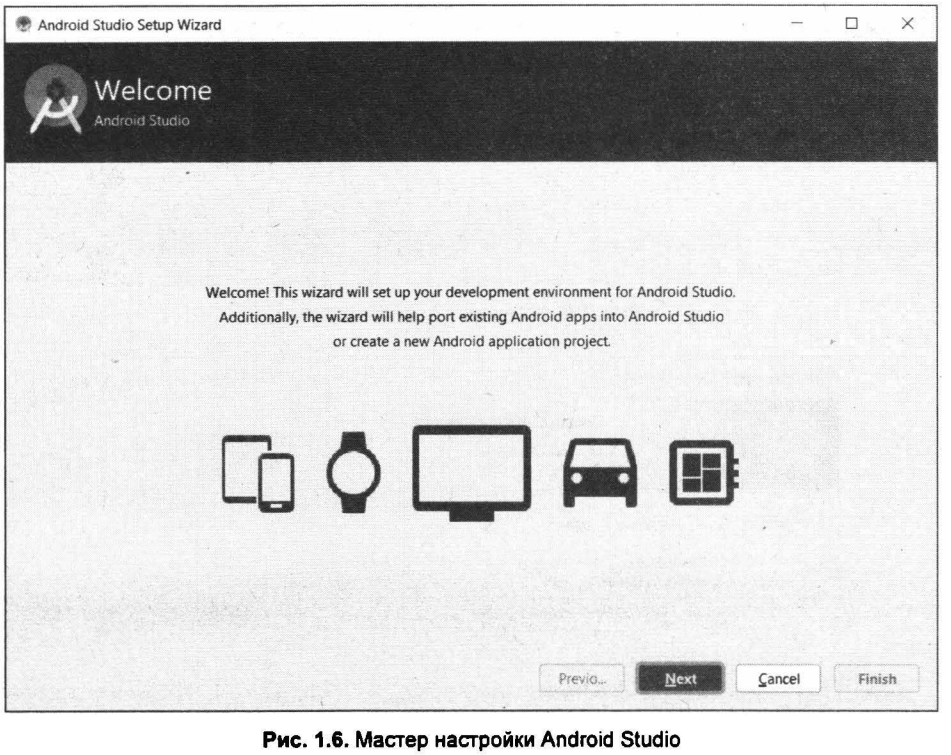


Рис. 1.6. Мастер настройки Android Studio

Первым делом нужно выбрать тип установки: стандартный (**Standard**) или пользовательский (**Custom**) (рис. 1.7).

Лучше выбирать пользовательский (**Custom**) — сразу можно будет установить больше параметров, чтобы потом не тратить на это время. Настройки предлагаются самые разные. Например, можно выбрать тему пользовательского интерфейса (рис. 1.8): светлую (**Light**) или темную (**Darcula**). Темная выглядит стильно, но ее

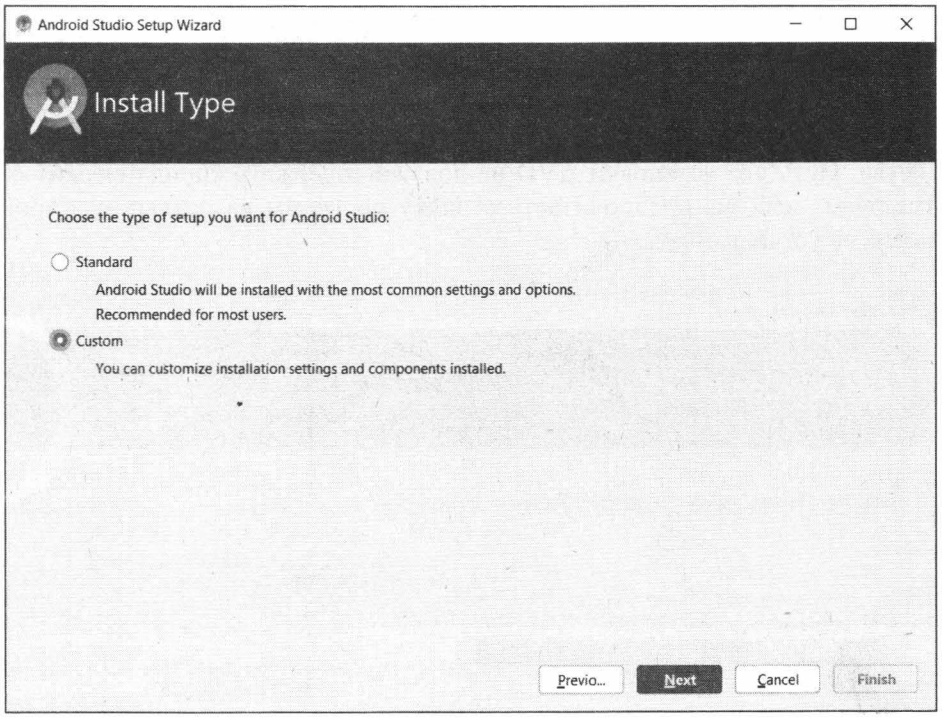


Рис. 1.7. Выберите тип установки

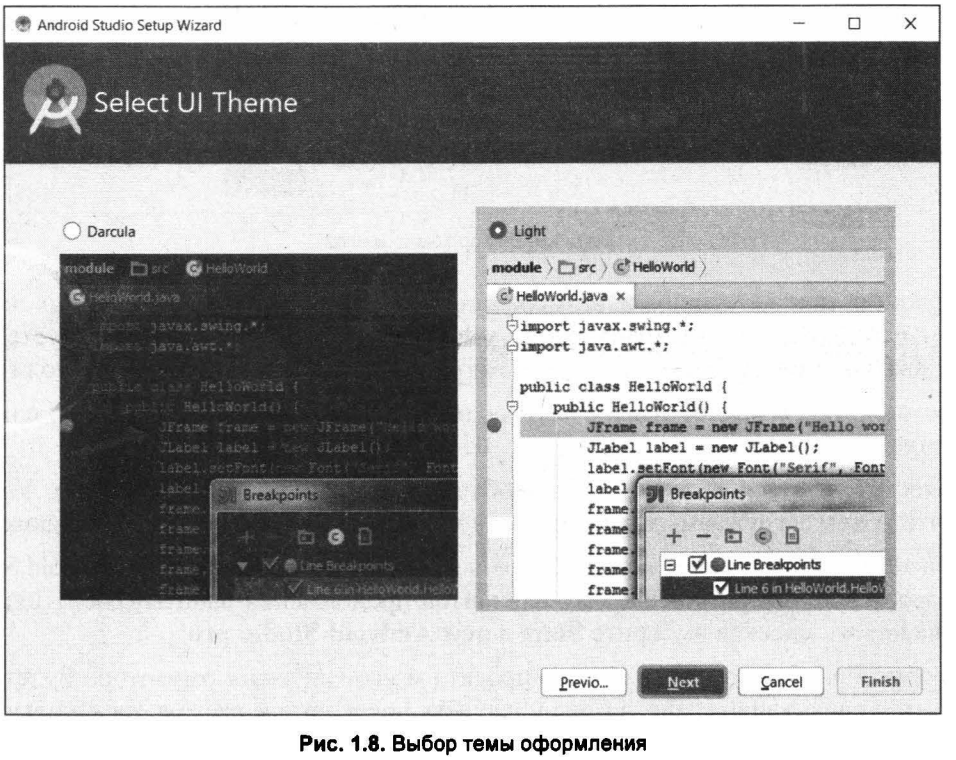


Рис. 1.8. Выбор темы оформления

лучше выбирать, если вы любите работать ночью. Если предпочитаете работать днем, то выбирайте светлую тему оформления — меньше будут уставать глаза.

Далее следует выбрать (рис. 1.9), какие компоненты нужно установить/обновить. Сразу установите Android Virtual Device (AVD) — это эмулятор Android-устройства. Путь для установки AVD не должен содержать кириллических символов. Впрочем, если вы предпочитаете отладку программ на реальном устройстве, можно его не устанавливать.

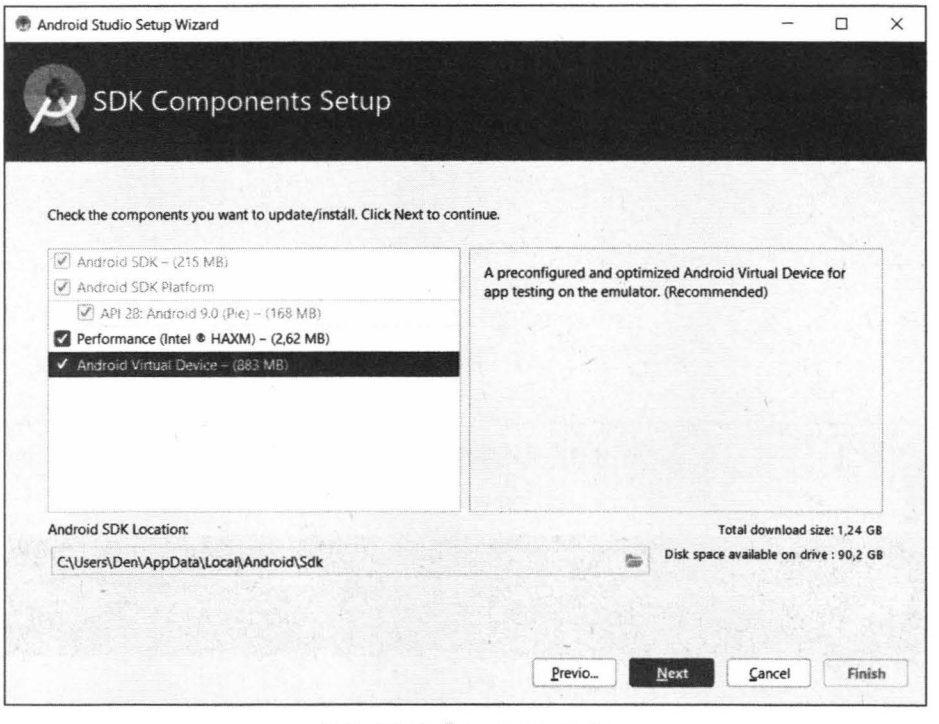


Рис. 1.9. Выбор компонентов

Следующий шаг — установка размера оперативной памяти, доступной для эмулятора Android (рис. 1.10). Значение по умолчанию — 2 Гбайт. Рекомендуется использовать его или выделить больше памяти, если вы можете это себе позволить.

Далее вам будет предоставлена сводка по загружаемым компонентам. Если все в порядке, нажмите кнопку **Finish** (рис. 1.11).

Осталось дождаться загрузки всех необходимых компонентов (рис. 1.12). Учитывая, что объем загружаемых данных — не маленький, придется немного подождать.

Дождитесь установки всех компонентов и нажмите кнопку **Finish** — Android Studio предложит вам выбрать несколько вариантов продолжения работы (рис. 1.13). Для начала нового проекта выберите **Start a new Android Studio project**.

Подробно процесс создания нового проекта и его запуска в эмуляторе будет рассмотрен в следующей главе, а пока у нас есть некоторые моменты, на которые стоит обратить внимание.

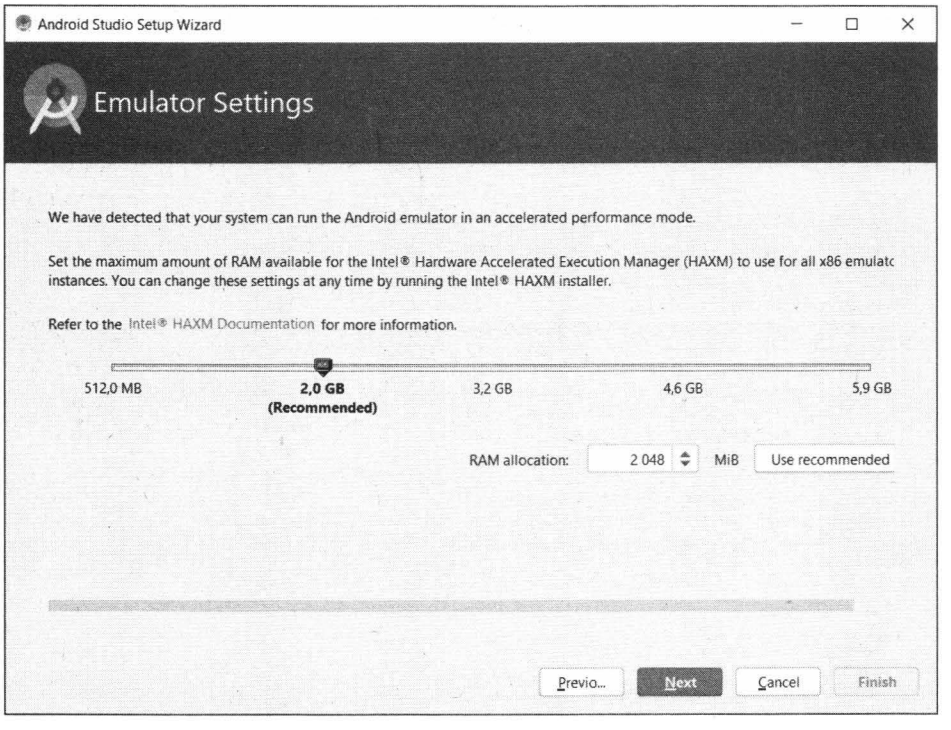


Рис. 1.10. Объем ОЗУ для эмулятора Android

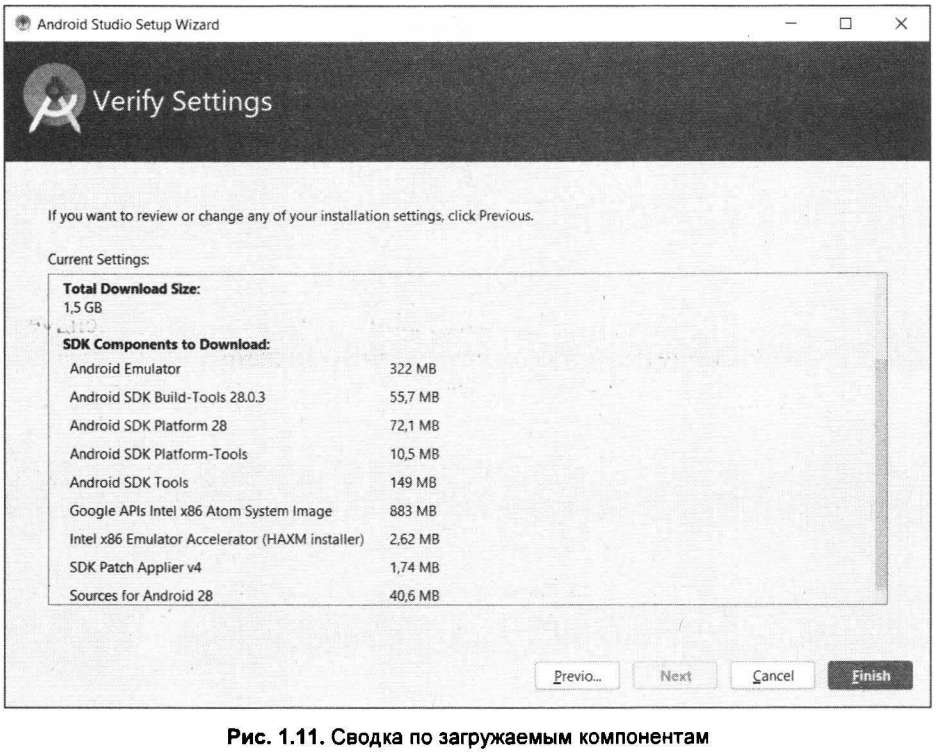


Рис. 1.11. Сводка по загружаемым компонентам

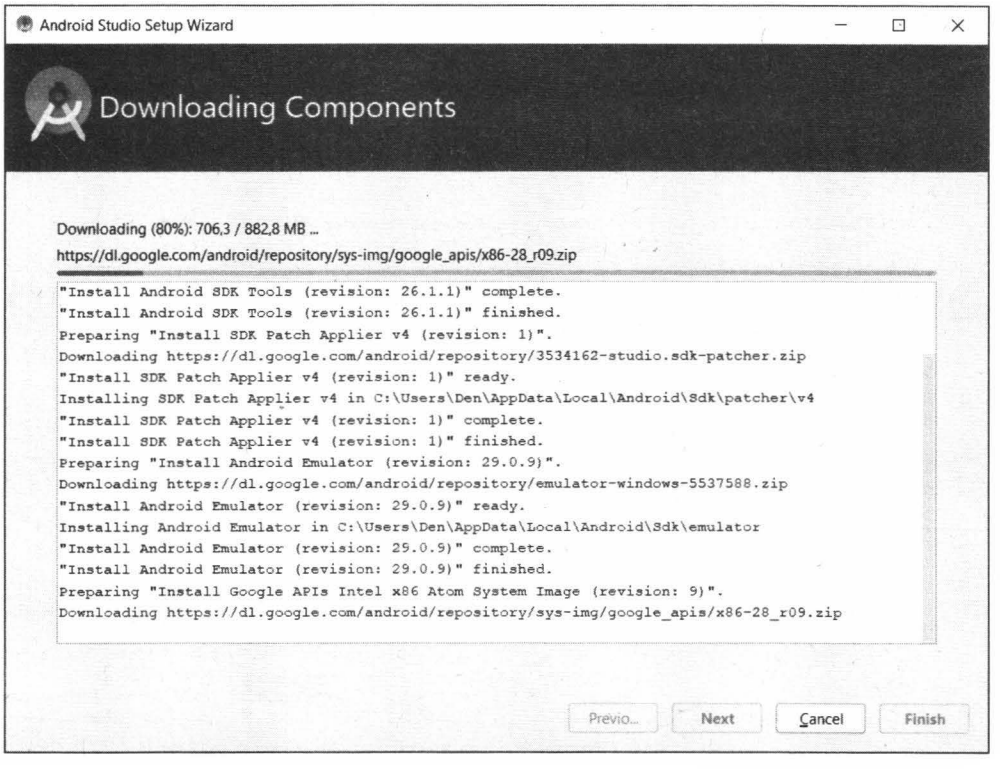


Рис. 1.12. Загрузка необходимых компонентов

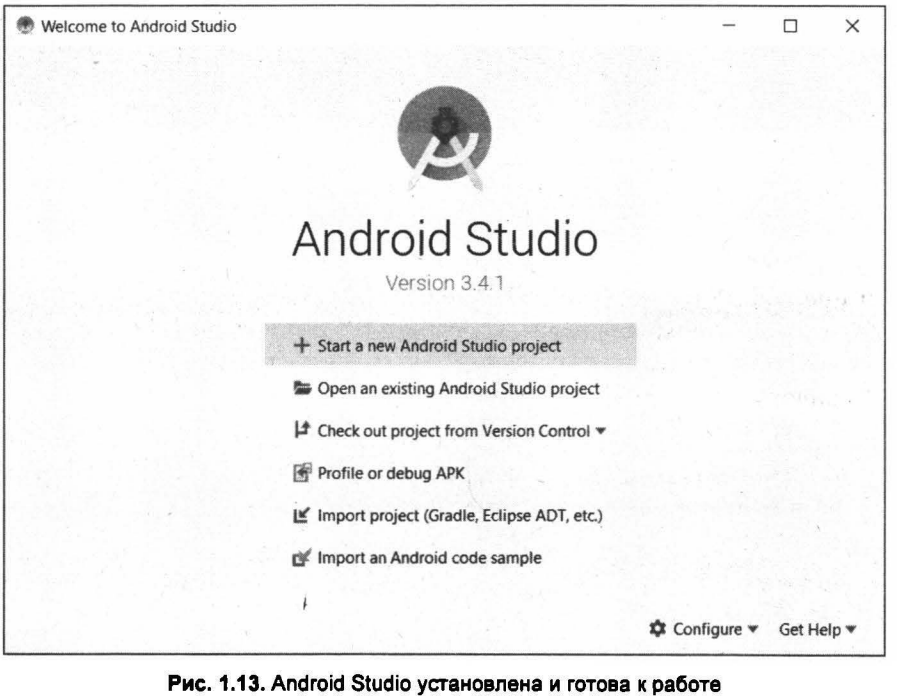


Рис. 1.13. Android Studio установлена и готова к работе

1.3.1. Уровни API

Прежде чем перейти к следующей главе книги, поговорим об уровне API. Уровень API — это число, которое однозначно идентифицирует версию Android. При программировании на Android вам придется иметь дело именно с уровнями API, а не с названиями версий Android. Уровни API представлены в табл. 1.1.

Таблица 1.1. Уровни API

Уровень API	Версия платформы	Уровень API	Версия платформы
1	1.0	16	4.1, 4.1.1
2	1.1	17	4.2, 4.2.2
3	1.5	18	4.3
4	1.6	19	4.4
5	2.0	20	4.4W
6	2.0.1	21	5.0
7	2.1-update1	22	5.1
8	2.2	23	6.0
9	2.3.1	24	7.0
10	2.3.3, 2.3.4	25	7.1
11	3.0	26	8.0
12	3.1	27	8.1
13	3.2	28	9.0
14	4.0, 4.0.1, 4.0.2	29	10.0 (Android Q)
15	4.0.3, 4.0.4		

1.3.2. Состав Android SDK

В состав Android SDK входят библиотеки, документация, эмулятор, примеры программ и инструментальные средства. Сейчас речь пойдет о последнем компоненте — об инструментальных средствах. Конечно, прямо сейчас они вам не понадобятся, не потребуются они и в *главе 2*, когда мы станем разрабатывать первое Android-приложение, но об их существовании вы обязаны знать.

Инструментальные средства Android SDK находятся в каталоге tools каталога, в который вы установили Android SDK (по умолчанию это C:\Users\<Имя пользователя>\AppData\Local\Android\sdk):

- android.bat — позволяет создавать, удалять и настраивать виртуальные устройства из командной строки. Все эти операции можно сделать в окне **Android SDK and AVD Manager** вручную;
- ddms.bat (Dalvik Debug Monitor Service) — позволяет управлять процессами на эмуляторе или на физическом устройстве, что помогает в отладке приложения;

- draw9patch.bat — графический редактор, позволяющий создавать графику для ваших Android-приложений;
- emulator.exe — собственно, сам эмулятор Android-устройства. Не спешите его запускать, он нуждается в предварительной настройке, которая будет описана в *главе 2*;
- hierarchyviewer.bat — позволяет оптимизировать графический интерфейс разрабатываемого приложения;
- mksdcard.exe — инструмент создания образа диска SD-карты, который можно использовать в эмуляторе для имитации внешней карты мобильного устройства;
- sqlite3.exe — инструмент для доступа к файлам данных SQLite.

В каталоге tools вы найдете и другие утилиты, но они менее важные, чем описанные здесь.

* * *

В следующей главе мы поговорим о настройке эмулятора Android-устройства и напишем наше первое приложение.

ГЛАВА 2

Первый проект, первое приложение и эмулятор Android

2.1. Создание нового проекта

В этой главе мы создадим самую простую программу для Android — приложение, выводящую строку «Hello, world!». Такая программа — традиция в мире программирования и отступать от нее мы не станем.

При запуске Android Studio открывается окно, показанное на рис. 2.1. В этом окне можно создать новый проект или открыть ранее созданный, а также открыть проекты, с которыми вы недавно работали (они будут отображены в списке слева). Выберите команду **Start a new Android Studio project** — откроется окно **Create New Project** (рис. 2.2).

> ***Примечание***
>
> Окна Android Studio весьма большие, и если у вас монитор не Full HD (т. е. не поддерживает разрешение 1920×1080), они могут не помещаться на экране. На старом моем ноутбуке (разрешение 1366×768) окно создания проекта на экране не поместилось и пришлось перетаскивать его на другой монитор. Не зря в системных требованиях к Android Studio указывается минимальное разрешение 1200×800.

Прежде всего в этом окне нужно выбрать форм-фактор устройств, для которых предполагается разрабатывать приложения, и определить шаблон будущего приложения. По умолчанию среда подразумевает, что вы будете создавать приложения для телефона и планшета (вкладка **Phone and Tablet**). Созданием именно таких приложений мы здесь и займемся. Среда предоставляет также возможности разработки приложений для Android TV, Wear OS и др., однако их рассмотрение выходит за рамки этой книги.

Какой шаблон выбрать? Сейчас имеет смысл работать или с **Empty Activity**, или с **Basic Activity**. Первый шаблон создает пустое окно приложения без каких-либо элементов управления, а второй — приложение с меню. Как можно видеть на рис. 2.2, мы выбрали шаблон **Empty Activity**.

В окне, открывающемся после выбора шаблона проекта (рис. 2.3), нужно ввести следующую информацию:

- **Name** — имя приложения, это название увидят пользователи. Поэтому придумайте достойное название. Пока что — для нашего первого проекта — можно оставить название, предложенное по умолчанию;

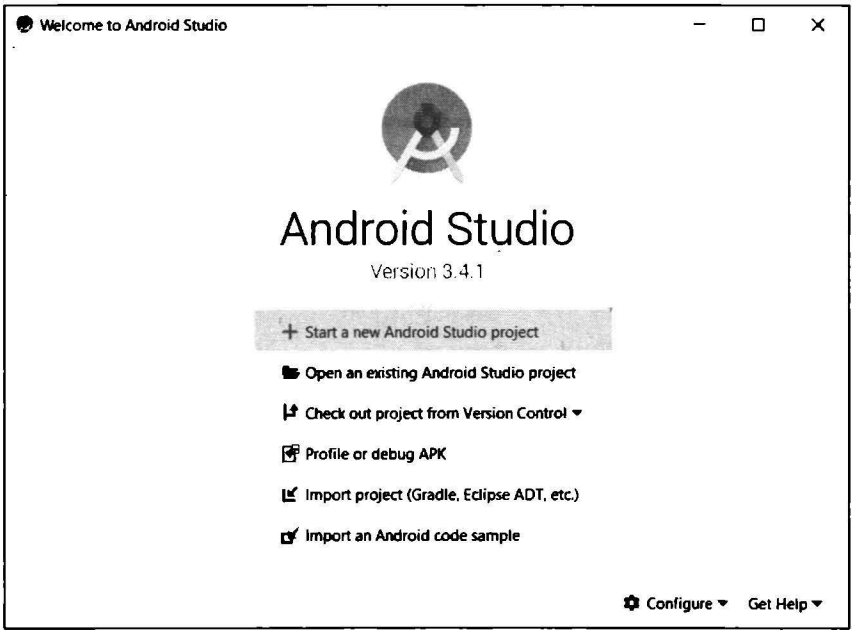


Рис. 2.1. Выберите действие Start a new Android Studio project

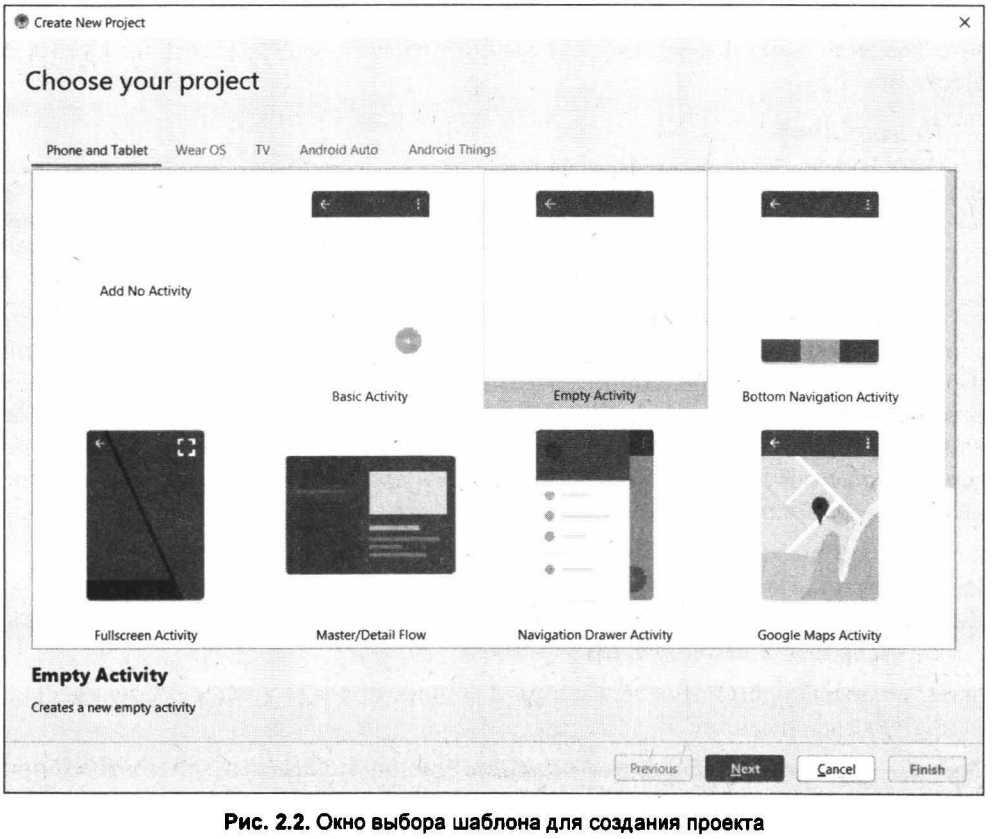


Рис. 2.2. Окно выбора шаблона для создания проекта

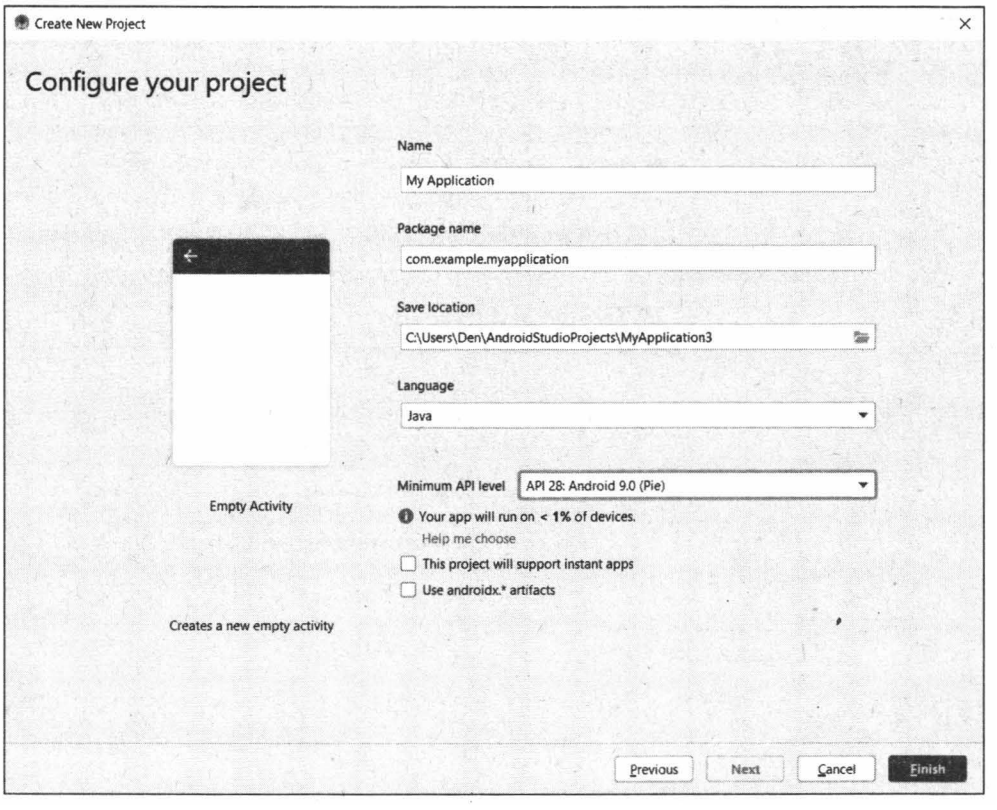


Рис. 2.3. Информация о проекте

- **Package name** — имя пакета, которое должно состоять из имени компании и названия приложения;
- **Save location** — по умолчанию проект сохраняется в каталоге C:\Users\<имя пользователя>\AndroidStudioProjects\<название приложения>. Вы можете изменить этот каталог;
- **Language** — язык, на котором вы будете создавать приложение. Сейчас среда поддерживает два языка: Kotlin и Java. Вы можете выбрать любой из них, но в этой книге мы будем создавать Android-приложения на Java;
- **Minimum API level** — минимальный уровень API Android, т. е. минимальная версия Android, на которой сможет запускаться приложение (см. табл. 1.1). Обратите внимание, что версию 9.0 пока поддерживают только 1% устройств (это по данным последней версии Android Studio, реально устройств уже значительно больше). Любопытно, что если выбрать версию 5.1, то вы получите поддержку 80,2% устройств (рис. 2.4). Так что Android Studio считает версию 5.1 в настоящее время наиболее универсальной.

> ***Примечание***
>
> В предыдущей версии Android Studio сначала нужно было ввести информацию о проекте (см. рис. 2.3), а уже потом выбирать шаблон, но в последней версии разработчики Android Studio несколько изменили мастер создания проекта.

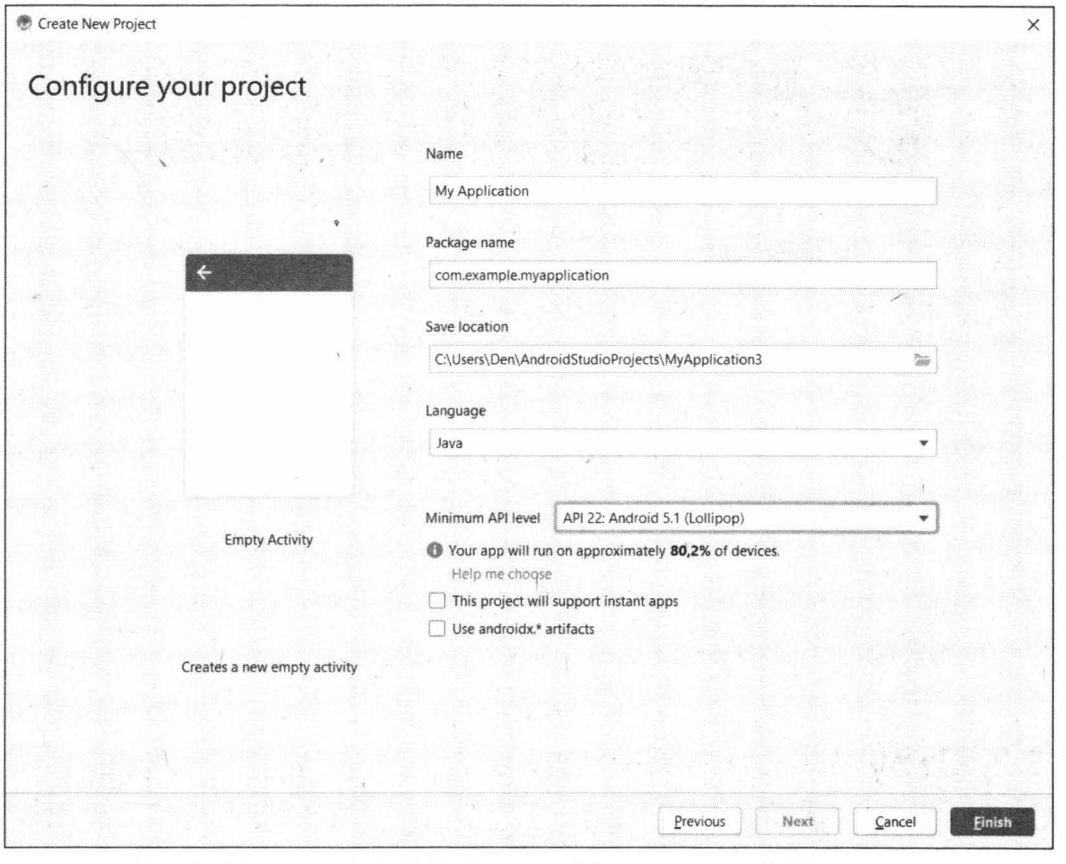


Рис. 2.4. По мнению Android Studio версию 5.1 поддерживает 80,2% устройств

Нажатие на кнопку **Finish** приведет к созданию проекта с выбранным шаблоном. Если у вас не очень быстрый компьютер, придется запастись терпением — создание нового проекта занимает некоторое время. Тем более, что при создании первого проекта среда подгружает из Интернета дополнительные файлы, поэтому во многом время создания проекта зависит от скорости соединения с Интернетом. В дальнейшем создание проекта будет занимать не более одной минуты.

Во время первого создания проекта нужно разрешить Java-машине доступ к сети. Если у вас установлен сторонний брандмауэр, окно, изображенное на рис. 2.5, может несколько отличаться от вашего.

> ***Примечание***
>
> Android Studio можно использовать в качестве benchmark-утилиты для измерения производительности системы. После создания проекта обратите внимание на строку состояния — в ней будет записано: **Gradle build finished in <время>**. Так вот, на моем стареньком ноутбуке (Intel Celeron 1000M 1.8 Ghz, 8 Gb DDR3, 500 Gb HDD) время сборки составляло 51 секунду. А на более современном (Intel Core i5 7200U up to 3.1 GHz, 8 Gb DDR4, 256 Gb SSD) — всего 15 секунд.

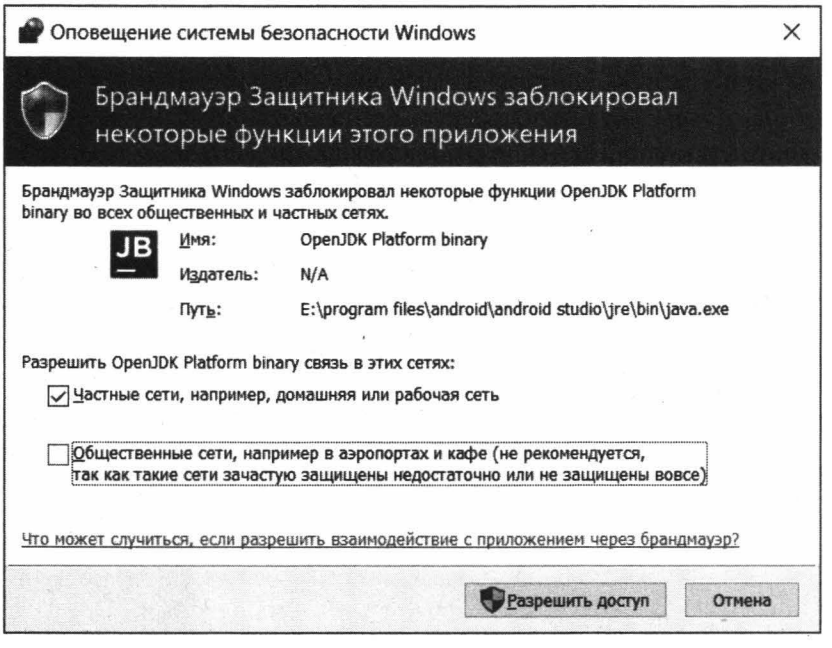


Рис. 2.5. Предоставление доступа Java-машине к сети

2.2. Структура приложения

Прежде чем продолжить, рассмотрим структуру нашего первого приложения. Приложение состоит из множества файлов — вот самые важные из них (все эти файлы находятся в каталоге проекта):

- app\res\layout\activity_main.xml — это XML-разметка шаблона (**Activity**), который вы выбрали при создании проекта. Имя файла может отличаться, если вы изменили имя шаблона;
- app\java\com\имя_компании\название_приложения\MyActivity.java — Java-код вашего приложения. Напомню, что приложения для Android разрабатываются на Java;
- app\manifest\AndroidManifest.xml — файл манифеста, он описывает фундаментальные характеристики вашего приложения. Далее мы рассмотрим его более подробно.

В каталоге app\res сосредоточены ресурсы вашего приложения. Так, в его подкаталоге layout хранятся XML-файлы разметки приложения, они описывают графический интерфейс пользователя. В подкаталоге menu содержатся файлы, определяющие элементы меню приложения. А в каталоге values — различные ресурсы, такие как строковые ресурсы и определения цветов. В файле strings.xml задаются строковые константы, в том числе и строка `My Application` — название вашего приложения.

2.3. Основное окно Android Studio

Надеюсь, пока вы читали предыдущий раздел, среда Android Studio уже успела подготовить первый проект, и вы видите основное окно Android Studio (рис. 2.6). По сути, среда уже подготовила для нас приложение `My Application`, и нам здесь ничего делать не требуется. Но хотя бы разберемся, что есть что.

Рис. 2.6. Основное окно среды Android Studio

В верхней части окна находятся меню и панель инструментов. Далее располагается навигатор по типу «хлебных крошек», позволяющий быстро перемещаться по каталогу файлов проекта. Сейчас, как показано на рис. 2.6, открыт файл app\java\com.example.myapplication\MainActivity.java.

Слева находится панель вкладок, позволяющая переключаться между структурой проекта, самим проектом, избранными проектами и т. д. Основной вашей областью будет именно вкладка **Project** (обратите внимание: вкладки здесь расположены вертикально по левой границе окна). Справа от нее находится расширенная версия навигатора, позволяющая более наглядно получать доступ к тем или иным файлам проекта.

Правее области навигатора расположена основная рабочая область среды. Она содержит вкладки редактируемых файлов. Как можно видеть на рис. 2.6, сейчас в нее загружен файл MainActivity.java.

О создании графического интерфейса приложения мы поговорим в следующих главах этой книги, а пока попробуем обратить ваше внимание на некоторые важные и интересные моменты. Щелкните двойным щелчком на файле content_main.xml, чтобы открыть графическую разметку приложения, и вы увидите, как приложение будет выглядеть на экране смартфона (рис. 2.7).

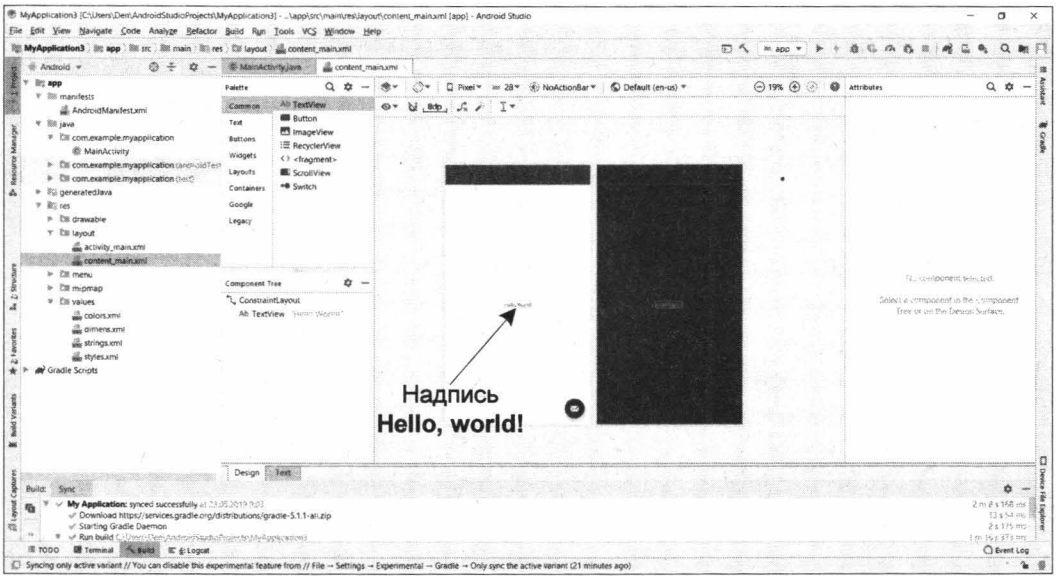


Рис. 2.7. Графическая разметка приложения и надпись **Hello, world!**

Наверное, вам интересно знать, как изменить надпись «Hello, world!»? Щелкните на ней (см. рис. 2.7), и в область **Attributes** (находится справа) будут загружены свойства выделенного элемента интерфейса, в настоящий момент — это наша надпись. Найдите свойство **text** и отредактируйте его так, как вам нужно (рис. 2.8).

Создать разметку (графический интерфейс) можно двумя способами: визуально и путем редактирования XML-файла. На рис. 2.7 и 2.8 показан графический способ

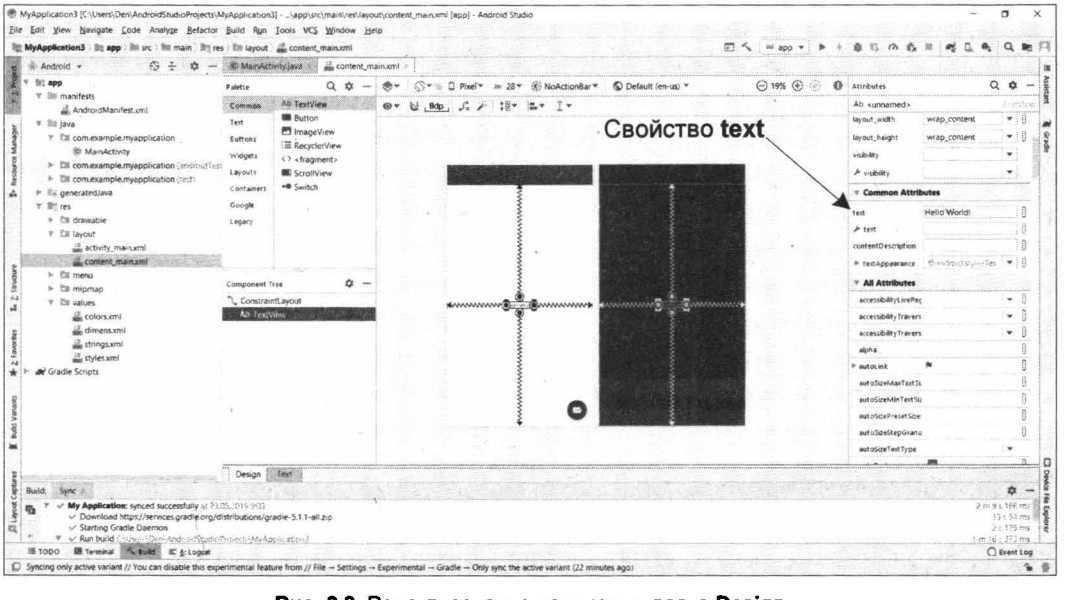


Рис. 2.8. Редактирование надписи: вкладка **Design**

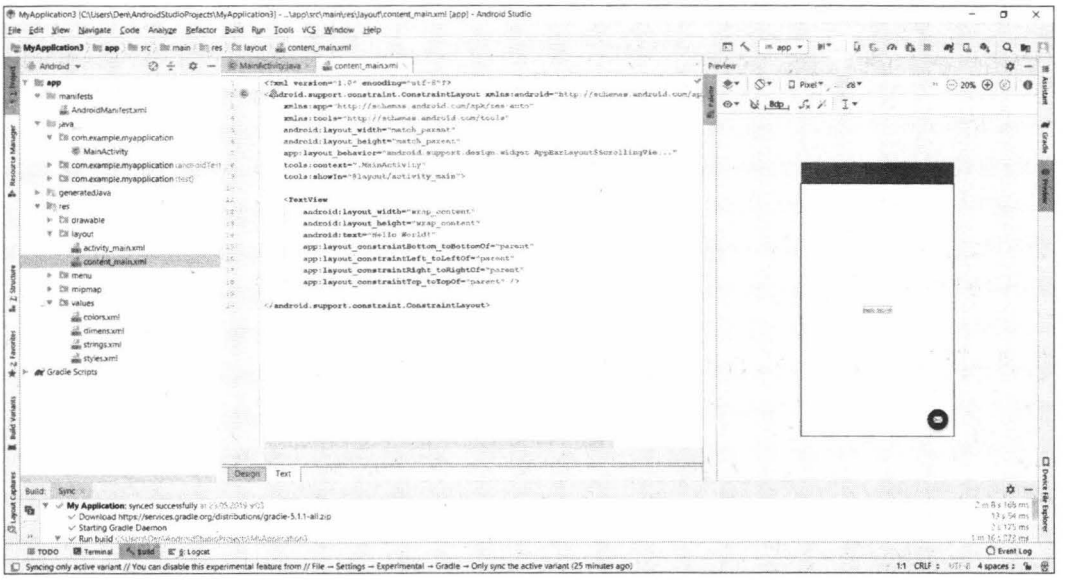


Рис. 2.9. Вкладка **Text**

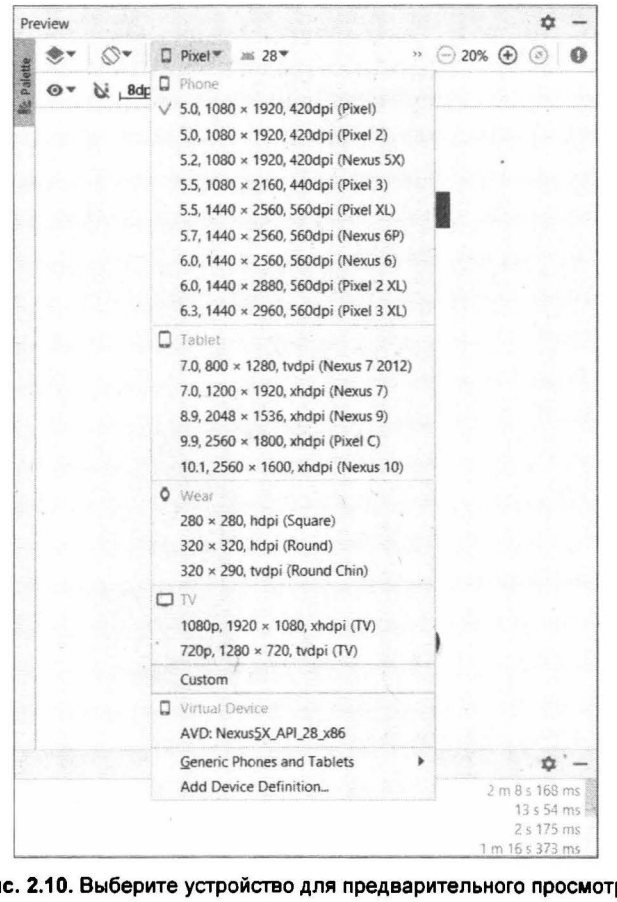


Рис. 2.10. Выберите устройство для предварительного просмотра

редактирования разметки — в нижней части рабочей области открыта вкладка **Design**. Для редактирования XML-кода переключитесь на вкладку **Text** (рис. 2.9). Область предварительного просмотра **Preview** сразу же отобразит внесенные вами изменения.

Область предварительного просмотра **Preview** на вкладке **Pixel** позволяет просмотреть, как будет выглядеть ваше приложение на разных устройствах (рис. 2.10).

Для запуска нашего приложения нужно подготовить эмулятор, чем мы и займемся в следующем разделе.

2.4. Подготовка эмулятора Android

Самая важная утилита из набора Android SDK — это эмулятор Android-устройства. Эмулятор позволяет отлаживать и тестировать приложения в реальной среде выполнения без необходимости их установки на физическое устройство. Даже если у вас есть физическое устройство, не спешите на нем запускать свое приложение — оно может на нем работать нестабильно. Сначала нужно запустить устройство в эмуляторе, и если оно будет работать нормально, можно пробовать запускать его на реальном устройстве.

Создать эмулятор можно с помощью окна **Android Virtual Device Manager**, открываемого по команде меню **Tools | AVD Manager**. По умолчанию при установке системы создается одно виртуальное устройство — **Android Virtual Device** под управлением Android 9.0 (уровень API 28) (рис. 2.11).

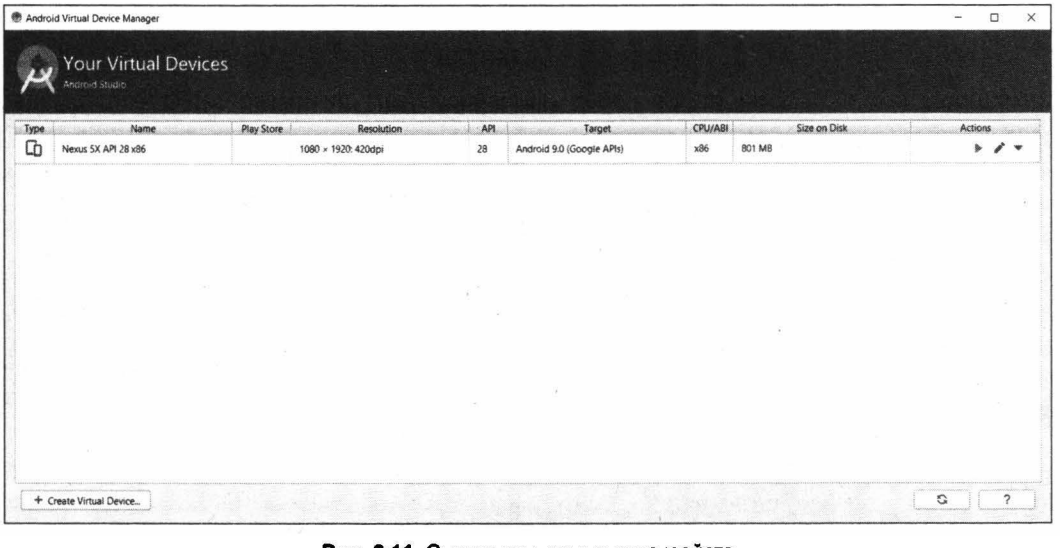


Рис. 2.11. Список виртуальных устройств

Можно использовать это устройство, а можно создать дополнительное. Вы можете создать несколько эмуляторов — например, один для платформы 6.0, а другой — для 7.0. Нажмите в окне **Android Virtual Device Manager** (см. рис. 2.11) кнопку

+ **Create Virtual Device** — откроется первое окно мастера создания нового виртуального устройства (рис. 2.12). Здесь нужно выбрать один из аппаратных профилей. Как можно видеть, на рис. 2.12 выбран профиль устройства с размером экрана 5.0" и разрешением 1080×1920 — это смартфон (вкладка **Phone**). Обратите внимание на колонку **Play Store** — значок Play Store в этой колонке означает, что образ поддерживает Play Store, т. е. вы сможете загружать в эмулятор приложения из магазина приложений, если это, конечно, вам нужно. Нажмите кнопку **Next**.

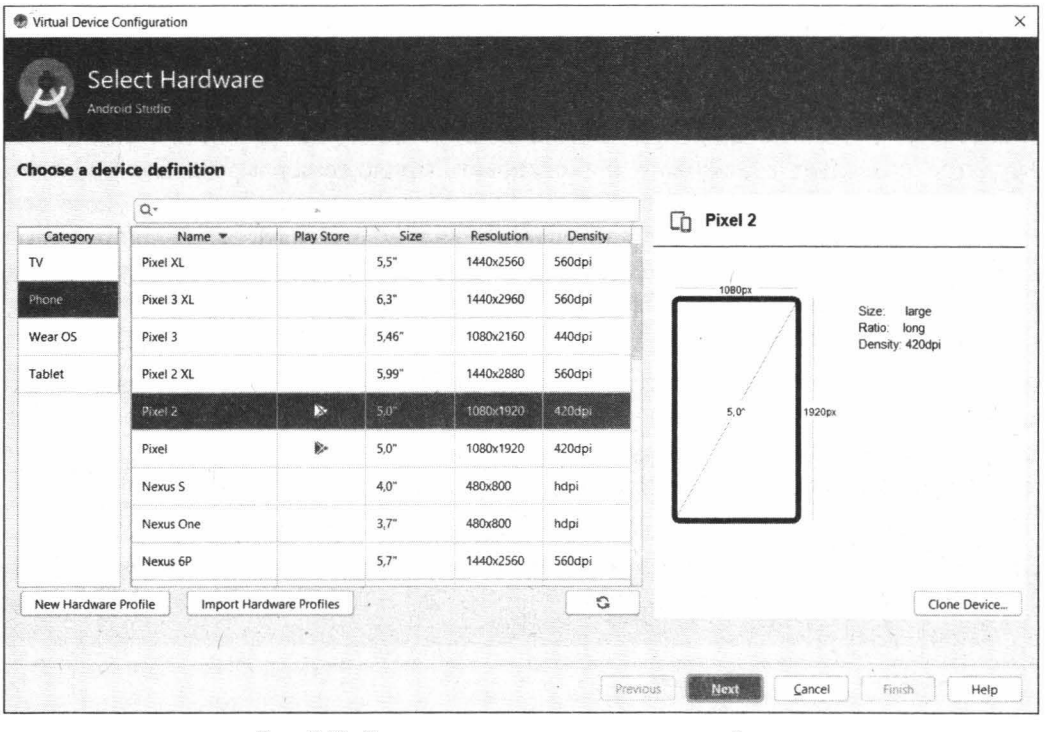


Рис. 2.12. Создание нового виртуального устройства

В следующем окне вы увидите список загруженных образов Android (рис. 2.13). На вкладке **Recommended** отображается список рекомендуемых образов. В настоящее время это образы для версий 7.0, 7.1.1, 8.0, 8.1, 9.0 и Q (следующая после P). Если вам нужны другие образы (например, для версии 4.4), их можно найти на вкладке **Other Images**. Для загрузки образа щелкните на ссылке **Download** у его названия.

Каждый образ Android занимает весьма много места (образ P, например, занимает 864 Мбайт), что особенно важно, если у вас SSD небольшого размера! Не нужно загружать все образы — загружайте только необходимые! Однако как минимум вам понадобится загрузить образы для версий 6.0 и 7.0.

Итак, щелкните на ссылке **Download** у нужного вами образа и дождитесь его загрузки (рис. 2.14). На рис. 2.15 показано, что загружен образ для Android 9.0 (API 28). Нажмите кнопку **Finish**.

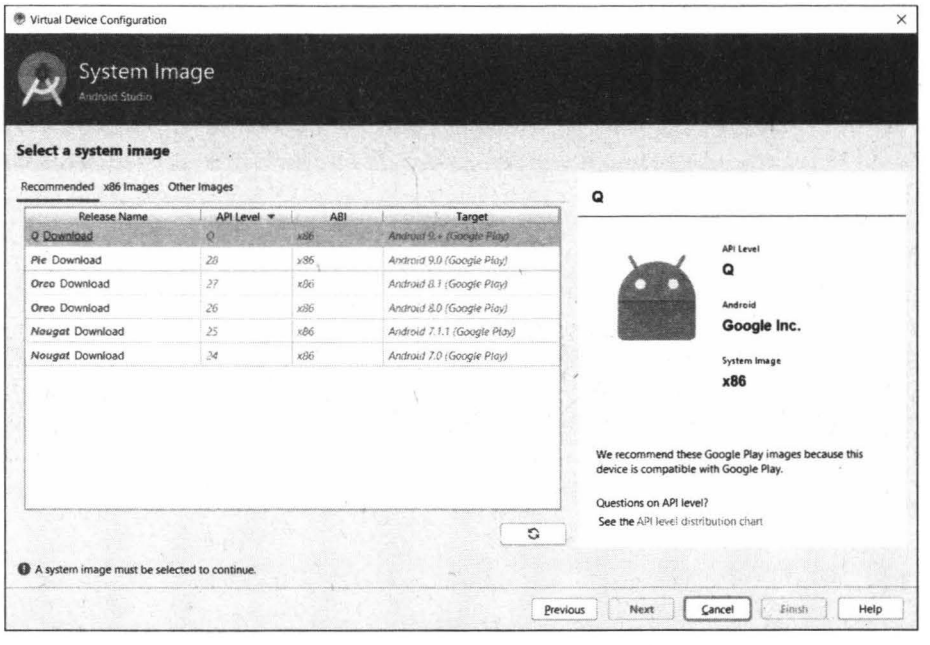


Рис. 2.13. Список доступных для загрузки образов

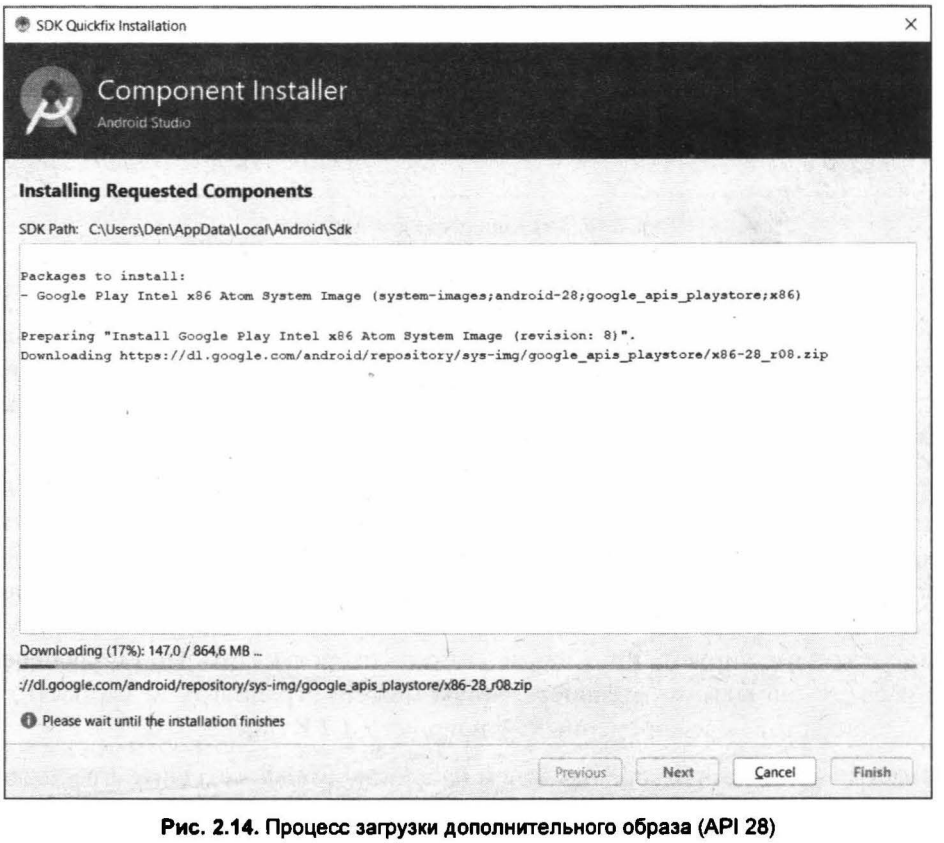


Рис. 2.14. Процесс загрузки дополнительного образа (API 28)

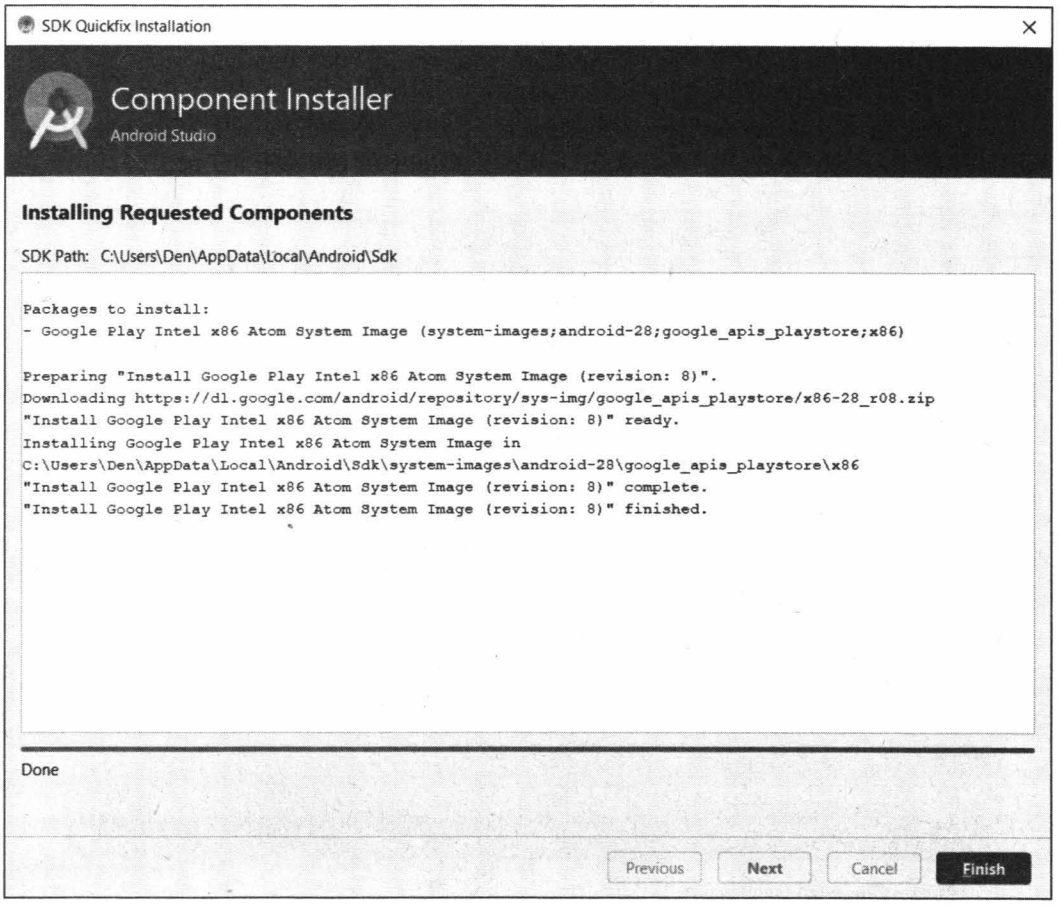


Рис. 2.15. Загружен образ для Android 9.0

По окончанию всех действий по загрузке системных образов в вашем списке должны появиться образы версий: 5.1, 6.0, 7.0, 8.1, 9.0 — это самые востребованные в настоящий момент версии. На рис. 2.16 видно, что пока загружены две версии: **Pie** (9.0) и **Nougat** (7.0). Образ для Q можно не загружать — особого смысла в этом сейчас нет.

Выберите версию **Pie** и нажмите кнопку **Next**. Введите в открывшемся окне название для эмулятора (в поле **AVD name**), а затем нажмите кнопку **Show Advanced Settings**. В расширенных настройках вы можете указать размер оперативной памяти, размер SD-карты, выбрать камеры устройства (переднюю и заднюю), причем камеры могут быть как реальными веб-камерами, подключенными к вашему компьютеру, так и эмулируемыми, а также выбрать скорость сети. По умолчанию сеть будет работать на полную мощность, но вы можете ограничить ее скорость, задав один из стандартов передачи данных — например, **LTE** (рис. 2.17).

Завершите создание эмулятора нажатием на кнопку **Finish**. Создайте еще несколько эмуляторов для разных версий Android.

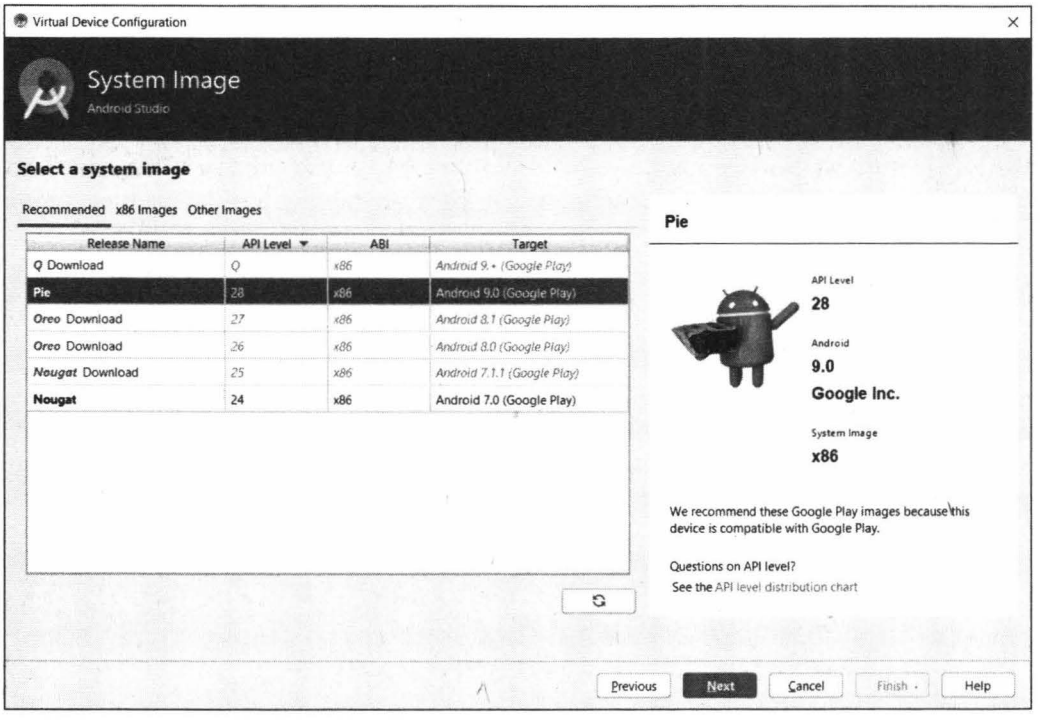


Рис. 2.16. Загружены образы **Pie** и **Nougat**

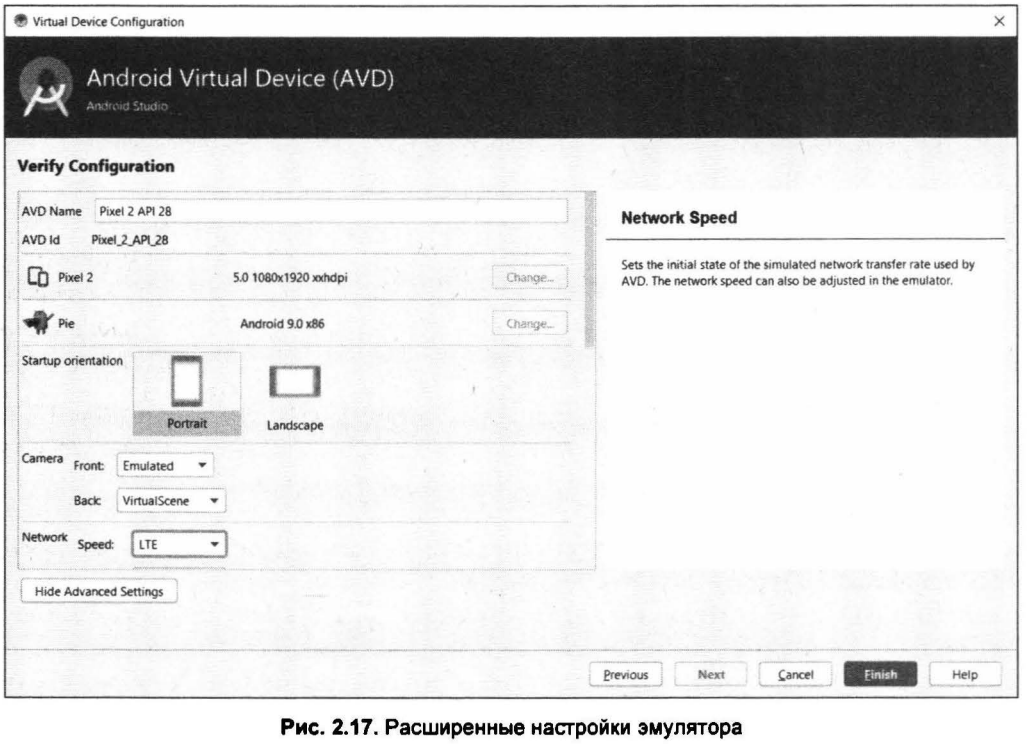


Рис. 2.17. Расширенные настройки эмулятора

2.5. Запуск приложения

Вот и настал момент истины. Сейчас мы запустим ваше первое приложение. Запустить приложение можно как в эмуляторе, так и на реальном устройстве. Сначала мы посмотрим на запуск приложения в эмуляторе, а уже потом — на реальном устройстве. Я настоятельно рекомендую сначала тестировать приложения именно в эмуляторе.

2.5.1. В эмуляторе...

Нажмите на панели инструментов Android Studio кнопку **Run** (или комбинацию клавиш <Shift>+<F10>). В открывшемся окне (рис. 2.18) вы увидите как список физических устройств, так и список эмуляторов, созданных ранее. Выберите, где именно нужно запустить приложение: на реальном устройстве или в эмуляторе (мы договорились начать с эмулятора), и нажмите кнопку **OK**.

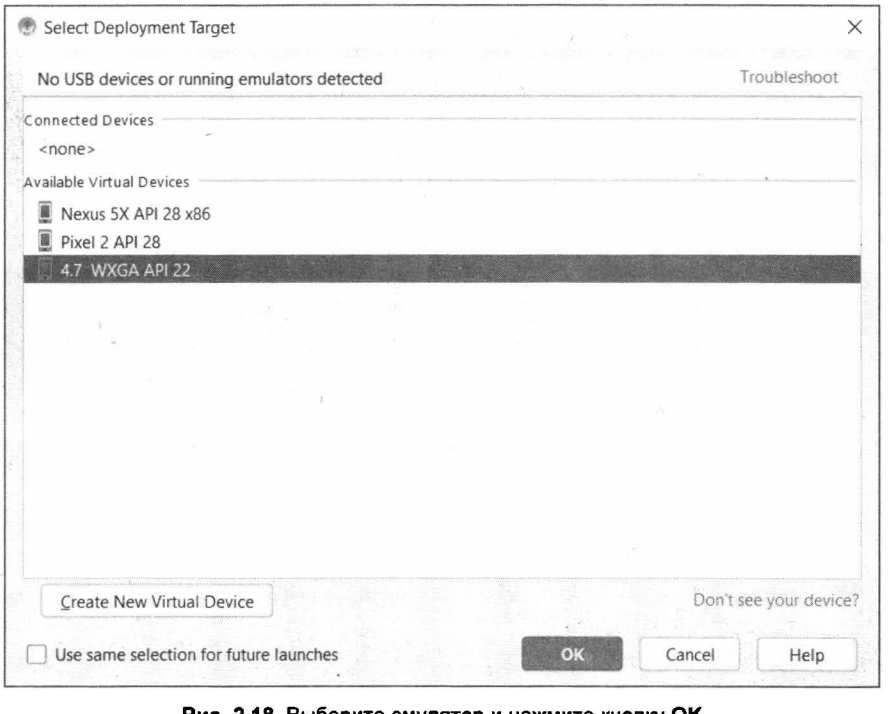


Рис. 2.18. Выберите эмулятор и нажмите кнопку OK

Начнется процесс запуска эмулятора (рис. 2.19) — вам будет показан индикатор этого процесса. Придется немного подождать. Запуск эмулятора, скажем так, не самый быстрый процесс. Если вы хотите профессионально заниматься разработкой приложений для Android, надо готовиться к модернизации своего компьютера (рекомендации по конфигурации подходящего оборудования приведены в *разд. 2.10*). Впрочем, ожидание вызвано не только запуском виртуального устройства — ведь проект нужно скомпилировать, сгенерировать APK-файл и поместить его по-

средством ADB (Android Debug Bridge) в выбранное виртуальное устройство (см. *разд. 2.7*). Эмулятор будет запущен в новом окне и придется опять немного подождать — на этот раз, пока внутри эмулятора запустится Android. Надо отметить, что в последней версии SDK эмулятор запускается быстрее.

> ***HARDWARE ACCELERATED EXECUTION MANAGER***
>
> Настоятельно рекомендуется установить Hardware Accelerated Execution Manager (HAXM) для ускорения работы эмулятора. Обычно HAXM устанавливается при установке Android Studio, но в некоторых случаях это может не произойти. Тогда при первом запуске любого эмулятора среда предложит его установить. Просто нажмите кнопку **OK** и подождите, пока HAXM установится. При успешной установке вы получите сообщение: **Intel HAXM installed successfully**.

Рис. 2.19. Эмулятор загружается

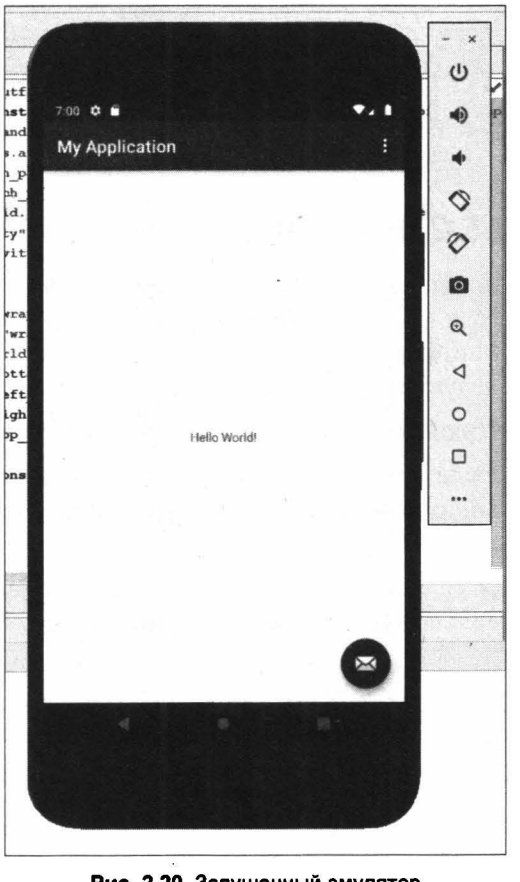


Рис. 2.20. Запущенный эмулятор

На рис. 2.20 показан уже загруженный эмулятор с запущенным в нем приложением «Hello World!». Управлять эмулятором можно с помощью кнопок, которые выводятся справа от окна эмулятора, или соответствующих им клавиатурных комбинаций (рис. 2.21).

Я вас поздравляю, вы только что создали свое первое Android-приложение! Причем не только создали, но и откомпилировали, и запустили его в эмуляторе.

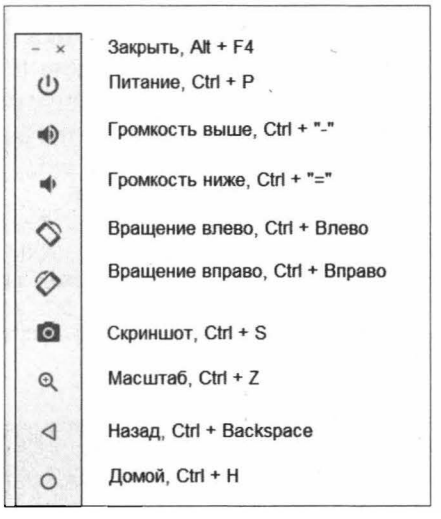


Рис. 2.21. Управление эмулятором с помощью клавиатурных комбинаций

2.5.2. На реальном устройстве...

Чтобы запустить приложение на вашем устройстве, нужно не просто подключить его к компьютеру. Первым делом нужно установить USB-драйверы для вашего смартфона/планшета. Помочь найти и установить подходящий драйвер сможет следующая страничка:

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

После установки нужно включить режим отладки по USB (об этом будет отдельный разговор) и подключить ваше устройство к компьютеру с помощью USB-кабеля.

Далее, как обычно, используйте команду **Run**, в открывшемся окне выберите реальное устройство и нажмите кнопку **OK**.

2.6. Проблемы при запуске приложения

2.6.1. Нюансы запуска приложения в эмуляторе

При запуске приложения (даже такого простого) нужно помнить о следующих нюансах:

- сначала в эмулятор производится загрузка операционной системы Android, а потом уже приложения. Опять-таки, если вы увидели черный/синий фон, подождите немного, ваше приложение будет запущено;
- в зависимости от версии платформы эмулятор может быть заблокирован. Для его разблокировки нажмите кнопку **Menu**, после чего вы сможете получить доступ к своему приложению. Опять-таки, если вы не увидели приложение, попытайтесь его отыскать в меню **Applications** и запустить оттуда;

- если после успешного запуска приложения вы внесли изменения в проект (например, изменили тестовую строку), но после нажатия кнопки **Run** вы видите старое приложение (без изменений), закройте окно эмулятора, а в Android Studio выполните команду **Build | Clean Project**. После этого можно опять нажимать **Run** для запуска проекта.

Нужно отметить, что проблема медленного запуска эмулятора в Android Studio полностью решена. Теперь эмулятор (во всяком случае, на более или менее современном ПК) запускается достаточно быстро. Ранее же приходилось ждать запуска несколько минут.

2.6.2. Включение режима отладки по USB на реальном устройстве

До версии 4.1 (включительно) для включения режима отладки по USB достаточно было зайти в меню **Настройки | Параметры разработчика** (Settings | Developer options), переместить переключатель в положение **ВКЛ** (ON), после чего включить параметр **Отладка USB** (USB Debugging) и нажать кнопку **OK**.

Однако, начиная с версии 4.2, пункт **Параметры разработчика** скрыт, и просто так включить отладку по USB уже нельзя. Чтобы открыть меню с параметрами для разработчиков, перейдите в раздел **Настройки | Об устройстве** (Settings | About phone). Несколько раз нажмите на поле **Build number**, пока не увидите сообщение, что вы уже разработчик. После этого в меню появится раздел **Параметры разработчика**, в котором нужно включить параметр **Отладка USB**.

2.7. Управление виртуальным устройством с помощью команды *adb*

Сейчас я расскажу вам об одной очень полезной утилите, позволяющей определить, установлено ли в эмуляторе приложение или нет, а также установить в эмулятор любой произвольный APK-файл (созданный как вами, так и сторонними разработчиками).

В каталоге C:\Users\<имя пользователя>\AppData\Local\Android\Sdk\platform-tools\ находится утилита `adb`. ADB (Android Debug Bridge) — это «отладочный мост» для Android (см. также *разд. 16.2*). Утилита полезна тем, что она может вывести список подключенных устройств (как виртуальных, так и физических), а также выполнить различные действия над ними. Например, программа может установить приложение на устройство. Вот список самых полезных действий:

- `adb install <путь к APK-файлу>` — устанавливает приложение на устройство;
- `adb uninstall <Имя APK-файла>` — удаляет приложение с устройства;
- `adb devices` — отображает список подключенных устройств;
- `adb reboot` — перезагрузка Android-устройства;

- `adb reboot recovery` — быстрая перезагрузка Android-устройства в режим **Recovery**;
- `adb shell` — вход в консоль. Вы должны увидеть символ # в строке;
- `adb shell cat /proc/mtd` — смотрим информацию о «разделах» памяти устройства;
- `adb shell df` — информация о разделах и свободных ресурсах;
- `adb push` — копирует файл в устройство.

На рис. 2.22 приведен вывод команды `adb devices`, демонстрирующий список подключенных устройств. Сейчас подключено всего одно устройство — эмулятор.

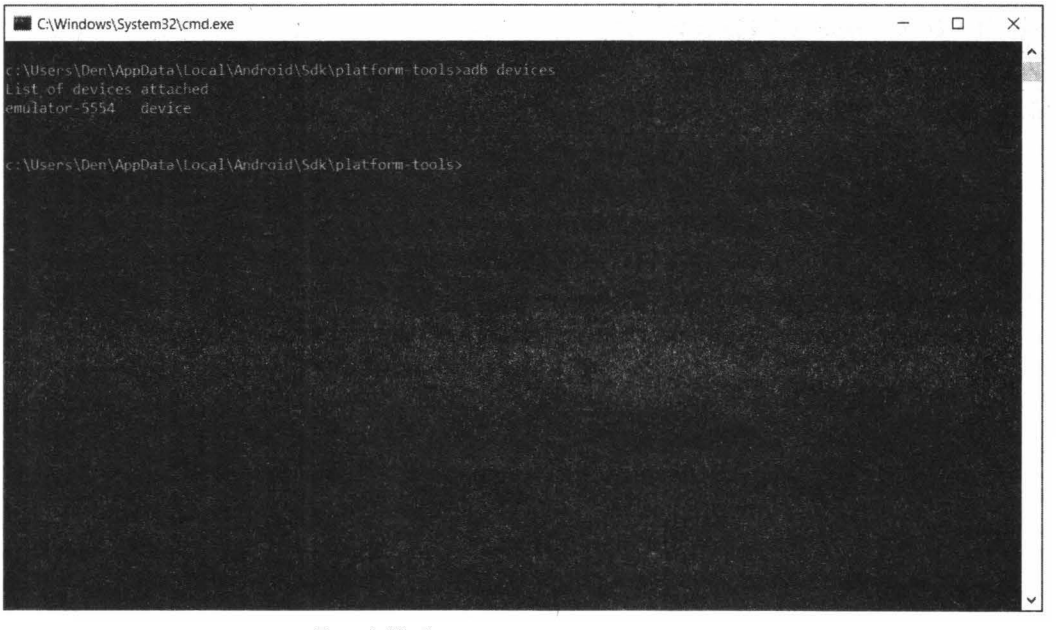


Рис. 2.22. Вывод команды `adb devices`

Как уже отмечалось, с помощью `adb` можно установить в эмулятор любой APK-файл, а не только созданный вами. Если у вас не получилось подключить смартфон к своему компьютеру, вы можете загрузить созданный вами APK-файл на ваш смартфон через Интернет (кстати, создаваемые вами APK-файлы хранятся на компьютере в каталоге app\build\outputs\apk\) и уже после этого установить его с помощью `adb` (но перед этим нужно разрешить установку приложений из неизвестных источников в настройках смартфона).

На рис. 2.23 показан процесс установки произвольного APK-файла в эмулятор (в нашем случае это файл cybersafesoft.cybersafe.mobile.apk). Как можно видеть, операция завершилась успешно, о чем говорит сообщение **Success**, выведенное после выполненной команды. Установленное приложение автоматически не запускается, поэтому для его запуска нужно перейти в меню **Applications** (рис. 2.24) и запустить его оттуда (рис. 2.25).

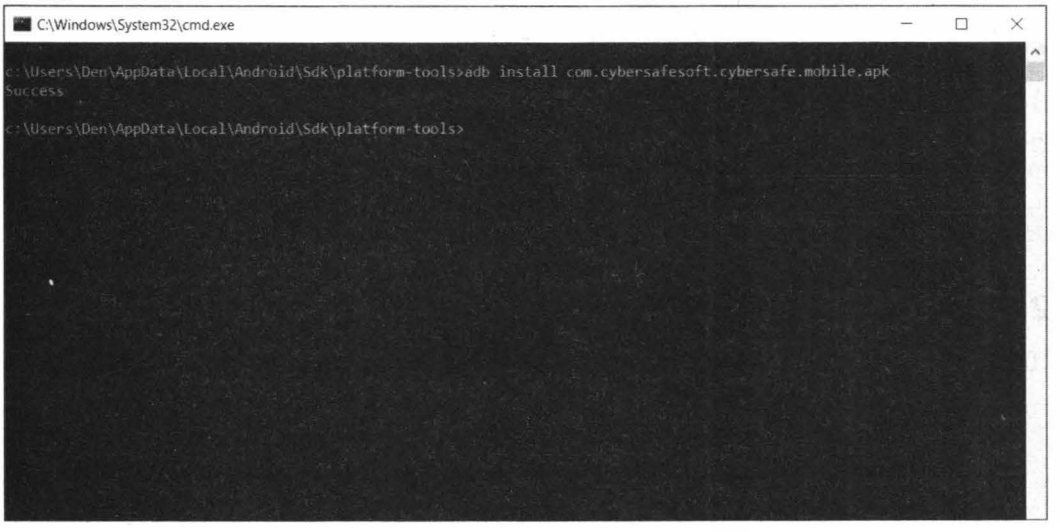


Рис. 2.23. Установка произвольного APK-файла cybersafesoft.cybersafe.mobile.apk в эмуляторе

Рис. 2.24. Приложение **Cybersafe Mobile** в эмуляторе установлено

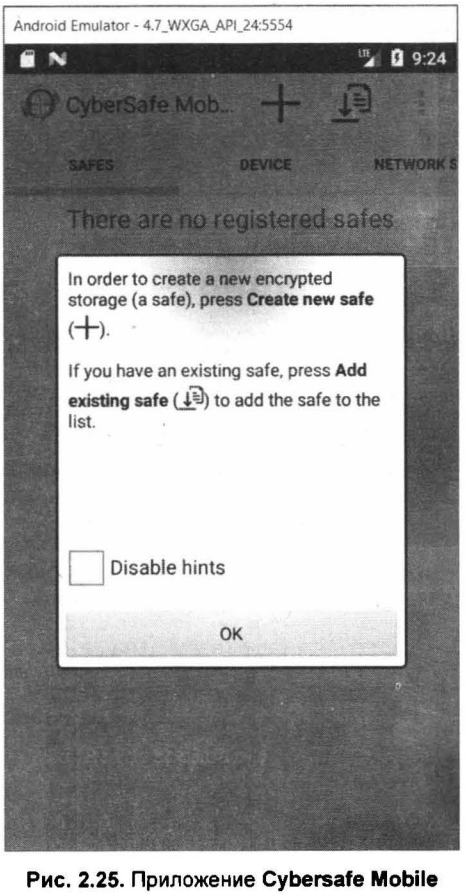


Рис. 2.25. Приложение **Cybersafe Mobile** в эмуляторе запущено

2.8. Командная строка

В предыдущем разделе мы уже частично коснулись вопроса управления устройством из командной строки. Но там описаны не все операции, которые можно выполнить с ее помощью. Оказывается, из командной строки можно создать и запустить проект. Думаю, интерфейс командной строки будет более привычен Linux-пользователям, поэтому специально им и посвящается этот раздел.

2.8.1. Создание проекта из командной строки

Для создания проекта из командной строки служит команда `android`:

```
android create project --target <target-id> --name MyFirstApp \
--path <path-to-workspace>/MyFirstApp --activity MyActivity \
--package com.example.myfirstapp
```

2.8.2. Запуск проекта из командной строки

Для запуска проекта нужно выполнить сценарий `gradlew`, который и создаст APK-файл:

```
$ ./gradlew assembleDebug
```

Затем созданный APK-файл нужно передать утилите `adb` для его запуска в эмуляторе:

```
adb install app/build/outputs/MyFirstApp-debug.apk
```

2.9. Создание снимка экрана виртуального устройства

Для создания снимка экрана виртуального устройства нужно в области **Logcat** окна Android Studio нажать на кнопку **Screen Capture** (рис. 2.26), а в открывшемся окне — на кнопку **Save** для сохранения полученного снимка экрана (рис. 2.27).

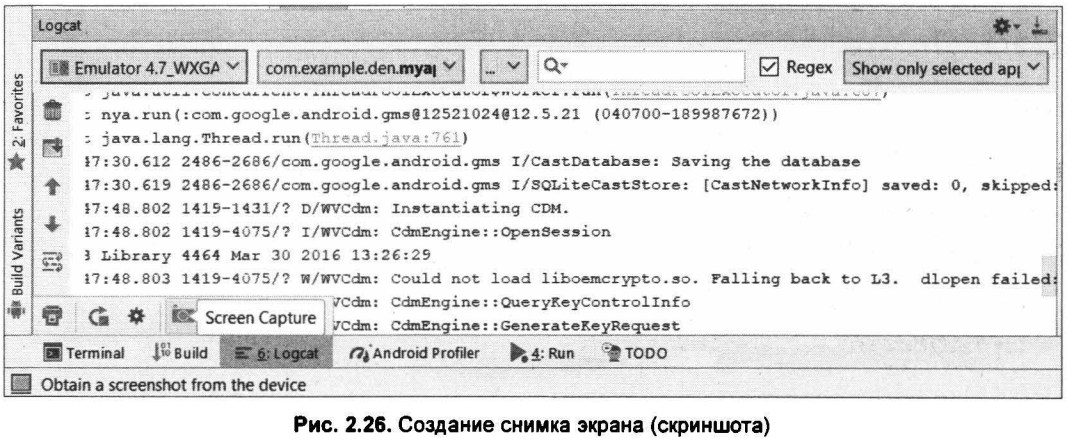


Рис. 2.26. Создание снимка экрана (скриншота)

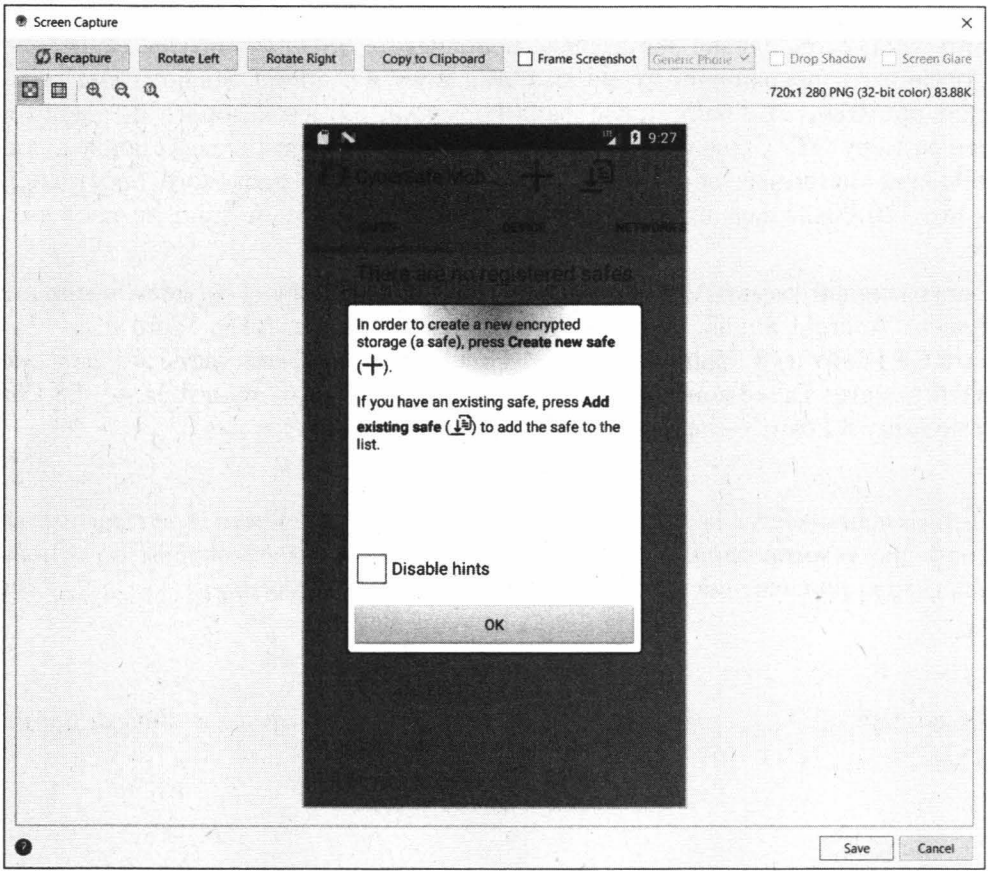


Рис. 2.27. Созданный снимок экрана

Этот способ создания снимка экрана считается стандартным. Однако можно просто «сфотографировать» окно эмулятора с помощью комбинации клавиш <Alt>+<Print Screen>, а потом в Paint вырезать нужную часть снимка и сохранить результат. Так что выбор способа создания снимков экрана остается на ваше усмотрение.

2.10. Подробно о системных требованиях

Наверное, вы заметили, что среда Android Studio и ее эмулятор работают не так быстро, как бы нам хотелось. Если вы собираетесь профессионально заниматься разработкой приложений для Android, то вам придется модернизировать свой компьютер или даже купить новый. Конфигурация мне видится примерно такая:

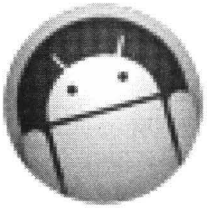
- процессор Intel Core i5 (i7 — еще лучше);
- оперативная память 8 Гбайт (это самый минимум);
- твердотельный жесткий диск (SSD) хотя бы 120 Гбайт;
- два монитора размером минимум 19": на одном будет отображаться Android Studio, на втором — эмулятор.

Понимаю, что покупать новое «железо» не всегда хочется. И для начала можно не менять процессор, однако обязательно приобретите SSD-накопитель. На него нужно установить операционную систему Windows и Android Studio. Сама система займет примерно 20 Гбайт (плюс файл подкачки, размер которого по умолчанию равен размеру ОЗУ), еще примерно 10 Гбайт отнимет сама среда (установка дополнительных системных образов также потребует своего дискового пространства). Так что 120 Гбайт вам должно хватить — такие накопители стоят не так уж и дорого.

На оперативной памяти сэкономить не получится. Сейчас на моем компьютере запущена Android Studio, браузер Chrome, текстовый редактор Word и всем этим занято 6,6 Гбайт из 8 Гбайт доступных. Только эмулятор занимает 1,4 Гбайт оперативной памяти. Та же конфигурация, но без запущенного эмулятора, — 4,8 Гбайт. Вот поэтому 8 Гбайт — необходимый минимум.

* * *

На этом заканчивается первая часть книги. В следующей части будут рассмотрены основы программирования в Android. Так, в *главе 3* мы поговорим об основных компонентах приложения и о том, как их применять на практике.



ЧАСТЬ II

Базовое программирование для Android

ГЛАВА 3

Основы построения приложений

В предыдущей главе мы создали и запустили первое Android-приложение. Ничего страшного, что оно было слишком простым, — главное, мы разобрались, как его запустить и как преодолеть некоторые проблемы, с которыми можно столкнуться при работе с эмулятором Android.

3.1. Структура Android-проекта

Просмотреть структуру Android-проекта можно в области **Project** основного окна Android Studio (рис. 3.1). Структура проекта меняется в зависимости от уровня API, т. е., фактически, от версии платформы Android, для которой ведется разработка. На рис. 3.1 показана структура проекта для платформы 8.1 (несмотря на наличие версии Android P, 8.1 — это версия, с которой вам сейчас придется работать на практике).

Обратите внимание, что на рис. 3.1 отображается не вся структура каталога приложения, а только те файлы, которые вы можете отредактировать с помощью Android Studio. Если просмотреть структуру каталога через файловый менеджер, то вы заметите, что каталоги, представленные в структуре, показанной в области **Project**, разбросаны по разным подкаталогам папки проекта, а также увидите отсутствующую в структуре папку build, в которой сосредоточены результаты сборки проекта, а именно — созданные APK-файлы (если быть предельно точным, то APK-файлы хранятся в каталоге app\build\outputs\apk).

Самый важный каталог — это каталог res (точнее, app\src\main\res\), представляющий собой каталог ресурсов приложения. В этом каталоге находятся следующие подкаталоги:

- res\drawable* — содержит изображения, адаптированные для разных разрешений экрана. По умолчанию в этих каталогах размещены файлы icon.png, адаптированные для разных разрешений;
- res\mipmap — здесь лучше держать значки приложений, которые будут отображены в меню Android;

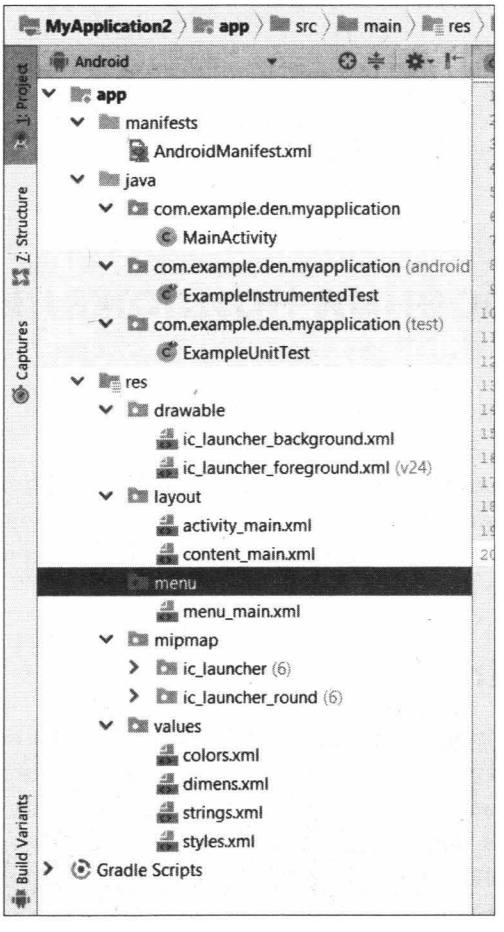


Рис. 3.1. Структура Android-проекта

- res\layout — содержит разметку элементов пользовательского интерфейса приложения;
- res\menu — здесь содержатся XML-файлы меню;
- res\values — здесь хранятся различные значения — например, строковые значения, массивы и т. д. В предыдущей главе мы уже познакомились с файлом strings.xml, содержащим строковые значения;
- res\xml — другие XML-файлы (этот каталог не создается автоматически, и его нужно создать вручную в случае необходимости).

В следующей главе мы подробно поговорим о создании пользовательского интерфейса. А пока только загляните в каталог res\layout. Разметку интерфейса пользователя можно создать как визуально (рис. 3.2), так и путем редактирования файла activity_main вручную — в текстовом режиме (переключение между режимами редактирования осуществляется с помощью вкладок **Design** и **Text** окна Android Studio). Как можно видеть, в каталоге res\layout кроме общего файла разметки activity_main.xml (листинг 3.1, *а*), существует также файл разметки контента content_main.xml (листинг 3.1, *б*). В нем как раз и содержится наша текстовая надпись

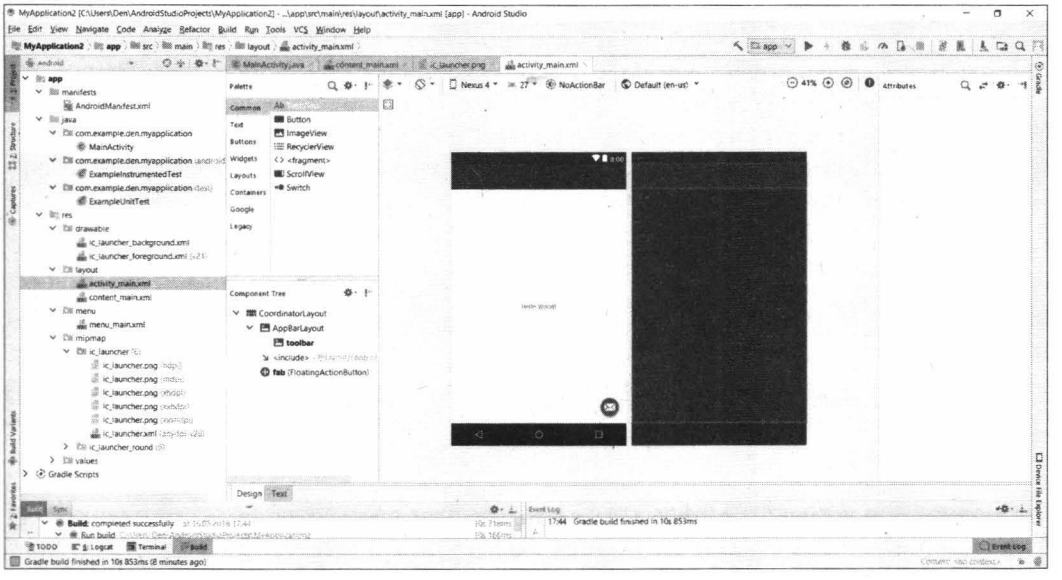


Рис. 3.2. Визуальная разметка проекта

«Hello World!» — текстовую разметку для отображения этой строки среда создает по умолчанию.

Листинг 3.1, *а*. Файл разметки activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
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"
    tools:context=".MainActivity">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>
```

```
    <include layout="@layout/content_main" />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/fab"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="@dimen/fab_margin"
        app:srcCompat="@android:drawable/ic_dialog_email" />

</android.support.design.widget.CoordinatorLayout>
```

Листинг 3.1, *б*. Файл разметки content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Обратите внимание, что в разметку activity_main.xml (см. листинг 3.1, *а*) разметка content_main.xml (см. листинг 3.1, *б*) уже включена.

Собственно текстовая разметка задается элементом `TextView` (см. листинг 3.1, *б*). Обратите внимание: в нем выводится текстовое значение `"Hello world!"`. При желании можно создать константу `@string/hello_world`, которую нужно потом не забыть определить в файле res\values\strings.xml (листинг 3.2).

Листинг 3.2. Файл res\values\strings.xml

```
<resources>
    <string name="app_name">My Application</string>
```

```
    <string name="action_settings">Параметры</string>
    <string name="hello_world">Hello world!</string>
</resources>
```

После компиляции проекта в каталоге app\build\generated\source\r\debug\<имя пакета> создается файл R.java (листинг его здесь не приводится из-за большого размера). Этот файл используется для обращения программы к своему каталогу ресурсов, и его не следует пытаться изменить, поскольку при следующей компиляции проекта среда его снова перезапишет. Впрочем, редактировать его и не требуется — при добавлении нового ресурса среда должным образом автоматически изменит и файл R.java.

Среда также автоматически создает JAVA-файл для главного окна приложения. Имя каталога, в котором находится этот файл, зависит от имени пакета. Поскольку я указал при создании проекта такое имя пакета: com.example.den.myapplication, то файл основного окна приложения у меня сохранен в каталоге app\src\main\java\com\example\den\myapplication. Имя JAVA-файла зависит от названия проекта — в нашем случае файл называется MainActivity.java (листинг 3.3). Если в вашем приложении предусмотрено несколько окон, тогда будут созданы отдельные JAVA-файлы для каждого окна (комментарии и прочие строковые константы в листинге 3.3 сгенерированы автоматически, поэтому не переведены и оставлены как есть).

Листинг 3.3. Файл MainActivity.java

```
package com.example.den.myapplication;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
                                                        Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

Нам осталось познакомиться только с одним файлом, который называется одинаково для любого Android-приложения: AndroidManifest.xml (листинг 3.4). В этом файле задается конфигурация приложения, приводится его название и т. п.

Листинг 3.4. Файл AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.den.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
```

```
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

В *разд. 3.4* мы файл манифеста рассмотрим подробнее, а пока вам нужно знать следующее.

Если вы раньше программировали для Android, то наверняка заметили, что больше в файле манифеста не определяется уровень API приложения. Так и есть. Теперь уровень API задается в разделе **Gradle Scripts** Android Studio (описание Gradle — системы автоматической сборки приложений — выходит за рамки этой книги).

Так вот, если вы все еще пытаетесь задать уровень API приложения в файле манифеста, то вы окончательно отстали от времени. Все уже давно определяют версию API в разделе **Gradle Scripts** через файл build.gradle (листинг 3.5).

Листинг 3.5. Файл build.gradle

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 27
    defaultConfig {
        applicationId "com.example.den.myapplication"
        minSdkVersion 24
        targetSdkVersion 27
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner
"android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                                                        'proguard-rules.pro'
        }
    }
}
```

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    implementation 'com.android.support:design:27.1.1'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:
                                                    espresso-core:3.0.2'
}
```

Как можно видеть, приведенный здесь файл задает минимальную версию SDK — 24 и целевую — 27. Получается, что наше приложение будет работать в версиях от 7.0 до 8.1.

3.2. Компоненты Android-приложения

Если вы программировали на языке C, то знаете, что у C-приложения есть одна точка входа (entry point) — функция `main()`. У Android-приложения нет единственной точки входа. Android-приложение состоит из компонентов, которые система может запускать, когда ей это необходимо. Одно приложение может также вызывать компоненты другого приложения, если, конечно, приложение разрешило использование своих компонентов.

Всего существует четыре типа компонентов:

- `Activity` — деятельность;
- `Service` — служба;
- `ContentProvider` — контент-провайдер;
- `BroadcastReceiver` — приемник широковещательных намерений.

> ***Примечание***
>
> *Намерения* — это асинхронные сообщения, активирующие деятельности, службы и приемники широковещательных намерений. Намерение является объектом класса `Intent`, представляющим собой содержание сообщения.

Рассмотрим компоненты Android-приложения подробнее.

- Начнем с первого компонента — `Activity`. Деятельность (активность) представляет собой визуальный интерфейс пользователя, т. е., попросту говоря, окно. Окно обычно заполняет весь экран мобильного устройства, но может быть меньших размеров, чем экран устройства.

  Если у приложения несколько окон, то у него будет несколько деятельностей. Каждая деятельность независима от других, при открытии новой деятельности (нового окна) работа предыдущей деятельности приостанавливается.

- У следующего компонента приложения — службы (`Service`) — нет графического интерфейса пользователя. Служба выполняется в фоновом режиме до тех пор, пока не завершит свою работу (подробно о службах мы поговорим в *главе 11*).

  Другие приложения могут подключаться к службе. После подключения к службе вы можете использовать функции, предоставляемые службой. Приложение может также запустить или остановить службу.

- Следующий компонент приложения — контент-провайдер (`ContentProvider`). Он служит для предоставления доступа к данным программы другим приложениям. Данные программы могут храниться как в файловой системе, так и в базе данных SQLite (могут быть и другие способы хранения данных).

- Для получения информации о внешних событиях и реакции на них служит компонент `BroadcastReceiver`. Источником события могут быть службы и другие приложения. Приложение может реагировать на любые события, которые оно посчитает важным.

  У приемников широковещательных намерений нет пользовательского интерфейса, но в ответ на событие они могут запустить деятельность, т. е. отобразить окно.

3.3. Процессы в ОС Android

Если хотя бы один компонент приложения будет востребован, ОС Android запустит процесс, содержащий единственный основной поток для его выполнения. Все компоненты приложения работают в этом процессе и потоке.

Однако так выглядит поведение по умолчанию. Вы можете создать многопотоковое приложение — т. е. сделать так, чтобы компоненты работали в других процессах и порождали дополнительные потоки. В случае нехватки памяти — если нужно выделить память более важным процессам — операционная система может завершить тот или иной процесс. Компоненты, выполняющиеся в этом процессе, будут уничтожены.

Но какой процесс можно уничтожить, а какой нельзя? Согласитесь, будет некрасиво, если Android уничтожит процесс, с которым пользователь работает в настоящий момент. Поэтому Android оценивает важность процесса с точки зрения пользователя и никогда не уничтожит видимый пользователем компонент процесса, но может завершить процесс с невидимыми деятельностями, которые более не отображаются на экране. Также может быть завершена служба, выполняющаяся в другом процессе.

Какой процесс будет завершен в первую очередь, зависит от его приоритета. У активного процесса критический приоритет, поэтому он не может быть завершен, у видимого и сервисного процесса высокий приоритет, поэтому система, скорее всего, его тоже не завершит. Главными кандидатами на завершение являются процессы с низким приоритетом: фоновые и пустые процессы. Сначала система удалит пустые процессы, а затем фоновые.

Полагаю, с приоритетами все ясно, но нужно разобраться с терминологией.

- *Активным процессом* (Foreground Process) называется процесс, с которым в текущий момент работает пользователь. Система считает процесс активным, если процесс выполняет деятельность, с которой работает пользователь, или же процесс выполняет службу, которую использует активная деятельность. Заметьте, что процесс А может выполнять службу, которая используется деятельностью, выполняющейся процессом Б. С этой деятельностью работает пользователь, поэтому процессы А и Б считаются активными.

  Процесс также считается активным, если он имеет объект `Service` и выполняется один из методов обратного вызова, который определен в этом объекте. Если процесс имеет объект `Broadcast Receiver` и выполняется его метод обратного вызова для приема намерения, то процесс тоже считается активным.

  Активные процессы система может удалить лишь в том случае, если памяти осталось так мало, что все активные процессы уже не могут выполняться одновременно.

- *Видимым* (Visible Process) называется процесс, не имеющий никаких приоритетных компонентов. Например, процесс А запустил деятельность не на весь экран. Пользователь работает с процессом Б, который также отображает активность. Активность процесса А не имеет фокуса, но все еще видна пользователю, поэтому процесс А называется видимым. Также видимым считается процесс, выполняющий службу, которая связана с деятельностью, находящейся на переднем плане, но не активна или частично закрыта другой деятельностью.

- *Сервисным* (Service Process) называется процесс, в котором выполняется служба и который не относится к активным и видимым. Обычно сервисные процессы не имеют интерфейса, но они выполняют задания, необходимые пользователю, — например, сервисным процессом может считаться фоновая работа проигрывателя. Поэтому система старается сохранить сервисные процессы.

- *Фоновым* (Background Process) считается процесс, в котором выполняется деятельность, которая в текущий момент не видна пользователю. Фоновый процесс может быть уничтожен в любой момент, если понадобилась память активному, сервисному или видимому процессу.

- *Пустой* (Empty Process) процесс вообще не содержит активных компонентов приложения. Система использует такие процессы в качестве кэша — для более быстрой последующей инициализации компонентов приложения. Однако если понадобится память, то такие процессы будут удалены в первую очередь.

3.4. Подробнее о файле *AndroidManifest.xml*

Перед запуском компонента приложения операционная система должна убедиться, что он существует. Для описания компонентов и служит файл AndroidManifest.xml. У каждого Android-приложения есть свой файл манифеста — даже у самых простых приложений.

Как уже отмечалось ранее, с появлением Gradle Scripts важность файла манифеста AndroidManifest.xml существенно уменьшилась. Если раньше в нем описывались различные компоненты приложения и зависимости, описывалась версия SDK, то сейчас в нем описываются только сугубо «косметические» данные: название приложения, его значок, тема оформления, главная активность (которая будет вызвана первой). Тем не менее файл манифеста все еще весьма важен, и он есть даже у самых простых приложений вроде нашего «Hello, World!».

В среду Android Studio встроен редактор файла манифеста (рис. 3.3). Для его запуска двойным щелчком щелкните на файле AndroidManifest.xml в окне Android Studio.

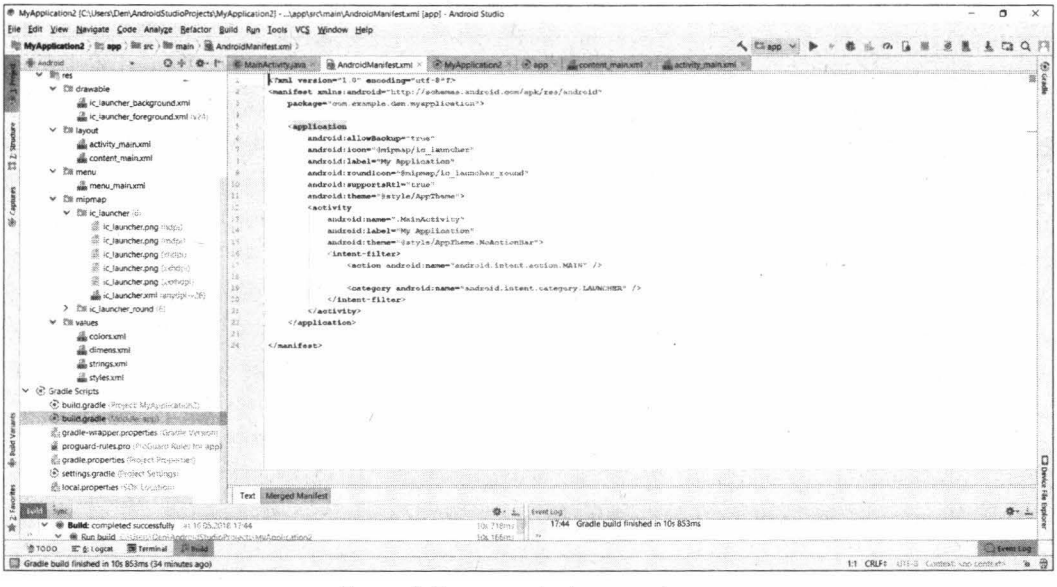


Рис. 3.3. Редактор файла манифеста

Основными элементами файла AndroidManifest.xml являются `<manifest>` и `<application>`. Остальные элементы этого файла используются при необходимости.

- Элемент `<manifest>` — корневой элемент файла AndroidManifest.xml. По умолчанию среда генерирует элемент `<manifest>` так:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.den.myapplication">
```

  Первый атрибут (`xmlns:android`) определяет пространство имен Android, этот атрибут всегда остается неизменным для всех Android-приложений. Второй атрибут (`package`) задает имя пакета приложения, которое вы указали при создании проекта.

- Самый главный элемент файла манифеста — `<application>`. В этом элементе описываются:
  - `<activity>` — деятельность;
  - `<activity-alias>` — псевдоним для деятельности;

- <service> — служба;
- <receiver> — приемник намерений;
- <provider> — контент-провайдер;
- <uses-library> — подключаемые библиотеки.

В нашем простом приложении, которое мы создали в предыдущей главе, элемент <application> представлен так:

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme.NoActionBar">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

В нашем случае указаны атрибуты: android.icon — значок приложения, android.label — название приложения, android:theme — тема оформления, а также некоторые другие.

- Элемент <activity> описывает деятельность. Его атрибут android:name задает имя класса, которое должно включать полное обозначение пакета, но поскольку имя пакета уже определено в <manifest>, то имя класса можно записывать в сокращенном виде, что и делает среда.

  Атрибут android:label задает текстовую метку, которая будет показана пользователю. Кроме упомянутых атрибутов элемент <activity> может содержать множество других, определяющих разрешение экрана, ориентацию экрана и т. д.

  > ***Примечание***
  >
  > Объявляя деятельности вручную, помните, что если деятельность не описана в элементе <manifest>, то пользователь не сможет ее увидеть.

- Следующий элемент — <intent-filter>. Он определяет типы намерений, на которые будут отвечать деятельность, сервис или приемник намерений. Его атрибут фильтр намерений (<intent-filter>) определяет, что может сделать деятельность или служба. В этом элементе могут быть дочерние элементы: <action>, <category>, <data>:

- элемент <action> добавляет действие к фильтру намерений. В фильтре намерений могут быть несколько элементов <action>, но как минимум должен быть один такой элемент;
- в элементе <category> определяется категория компонента, который должен обработать намерение. Как правило, задаются строковые константы (в качестве имени намерения), определенные в классе Intent, например:

```
<category android:name="android.intent.category.LAUNCHER" />
```

- элемент <data> используется, чтобы добавить спецификацию данных к фильтру намерений. Спецификация может быть (URL), типом данных или и интернет-адресом, и типом данных.

Мы рассмотрели основные элементы файла манифеста. Остальные элементы будут рассмотрены далее по мере необходимости.

3.5. Разрешения Android-приложений

Существуют два способа определения разрешений: при запуске приложения и во время его выполнения. При запуске разрешения выполняются так, как было показано ранее, — через файл манифеста:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.snazzyapp">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application ...>
        ...
    </application>
</manifest>
```

В Android 6 (API 23) появился еще один способ определения разрешений — во время выполнения приложения, или «на лету».

> ***Внимание!***
>
> Все описанное в этом разделе далее верно только для API 23, и если у вас устройство или целевой API SDK приложения 22 или ниже, система попросит пользователя при установке или обновлении приложения предоставить все опасные разрешения.

Прежде всего нужно проверить, есть ли у приложения нужное ему разрешение. Это делается с помощью метода ContextCompat.checkSelfPermission(). Проверим, есть ли у приложения разрешение на запись события в календарь:

```
// Считаем, что активность thisActivity является текущей
int permissionCheck = ContextCompat.checkSelfPermission(thisActivity,
        Manifest.permission.WRITE_CALENDAR);
```

Если у приложения есть нужное разрешение, метод возвращает PackageManager.PERMISSION_GRANTED, если же разрешения нет, возвращается значение PackageManager.PERMISSION_DENIED.

Для запроса разрешений используется метод `requestPermissions()`:

```
ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);
```

Здесь `MY_PERMISSIONS_REQUEST_READ_CONTACTS` — определенная приложением константа.

Далее приведен более полный код запроса разрешений:

```
// Здесь thisActivity является текущей
// Проверяем, есть ли у приложения нужные разрешения (READ_CONTACTS)
if (ContextCompat.checkSelfPermission(thisActivity,
                Manifest.permission.READ_CONTACTS)
        != PackageManager.PERMISSION_GRANTED) {

    // Если разрешений нет, выводим пользователю пояснение, что
    // нам нужно. Определить, нужно ли пояснение, можно с помощью
    // метода shouldShowRequestPermissionRationale
    if (ActivityCompat.shouldShowRequestPermissionRationale(thisActivity,
            Manifest.permission.READ_CONTACTS)) {

        // Показываем объяснение асинхронно, не нужно блокировать
        // весь поток в ожидании ответа пользователя

    } else {

        // Объяснение не нужно, можем запросить разрешение
        ActivityCompat.requestPermissions(thisActivity,
                new String[]{Manifest.permission.READ_CONTACTS},
                MY_PERMISSIONS_REQUEST_READ_CONTACTS);

    }
}
```

После того как мы запросили разрешения, нужно определить, были ли они нам предоставлены. Обработать ответ позволяет следующий код:

```
@Override
public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[] grantResults) {
    switch (requestCode) {
        case MY_PERMISSIONS_REQUEST_READ_CONTACTS: {
            // Если запрос отменен, результирующие массивы будут пусты
            if (grantResults.length > 0
                && grantResults[0] == PackageManager.PERMISSION_GRANTED) {

                // разрешения предоставлены, можно сделать то,
                // что мы собирались
```

```
            } else {

                // разрешения не предоставлены, нужно отключить функционал,
                 // требующий разрешений
            }
            return;
        }
    }
}
```

В качестве дополнительного материала рекомендую ознакомиться со следующими документами:

- **http://developer.android.com/training/permissions/requesting.html** — запрос разрешений;
- **http://developer.android.com/training/permissions/best-practices.html** — лучшая практика;
- **http://developer.android.com/guide/topics/security/permissions.html#normal-dangerous** — обычные и опасные разрешения.

ГЛАВА 4

Интерфейс пользователя

4.1. Разметка интерфейса

4.1.1. Редактор разметки

Графический интерфейс пользователя формируется с помощью объектов: `View` (представление) — класс `android.view.View` и `ViewGroup` (группа представлений) — класс `android.view.ViewGroup`. Класс `ViewGroup` является дочерним классом для `View`.

Класс `View` — основа для подклассов, которые называются *виджетами* и представляют собой элементы пользовательского интерфейса: текстовые поля, кнопки и т. п.

Каждый объект `ViewGroup` — это контейнер, содержащий и упорядочивающий дочерние объекты `View`. Контейнерами являются такие элементы, как `RelativeLayout`, `LinearLayout`, `GridLayout`, `ConstraintLayout` и ряд других.

Разметка интерфейса пользователя — это процесс размещения элементов интерфейса (виджетов) в конкретном окне приложения (для конкретной деятельности). Разметку можно выполнять двумя способами. Первый заключается в редактировании XML-файла окна приложения. Для главного окна приложения это файл res\layout\main.xml. Этот способ по началу вам покажется более «суровым». Начинающим программистам больше понравится второй способ — графический. Android Studio предлагает очень удобный редактор разметки интерфейса пользователя (рис. 4.1).

> ***Примечание***
>
> Если вы программировали на MS Visual Studio, где также можно создавать приложения для Android, то наверняка заметили, что визуальный редактор Visual Studio все же удобнее редактора разметки Android Studio. Хотя, возможно, это дело привычки.

Проблем с использованием визуального редактора, полагаю, у вас не возникнет, поэтому далее мы будем рассматривать разметку интерфейса пользователя путем редактирования XML-файла. На практике же вы можете использовать тот способ, который вам больше нравится.

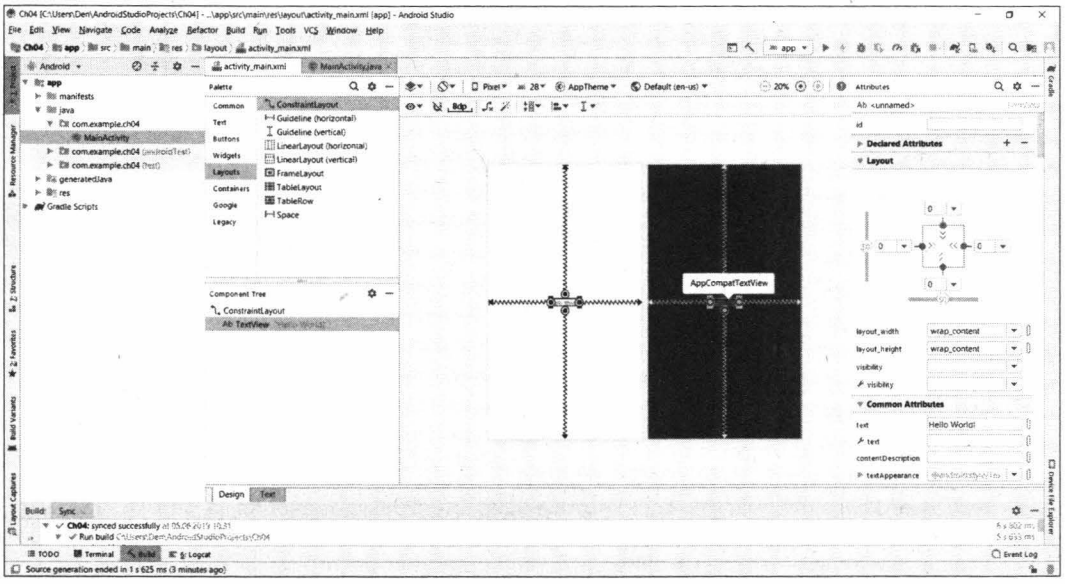


Рис. 4.1. Графический редактор разметки интерфейса пользователя

Итак, каждый элемент файла разметки является объектом класса `View` или класса `ViewGroup`. Если представить все элементы интерфейса пользователя в виде иерархии, то объекты класса `ViewGroup` будут родительскими объектами, а `View` — дочерними для них. Иерархия созданного интерфейса отображается в области **Component Tree** редактора интерфейса. На рис. 4.2 видно, что добавлена разметка `ConstraintLayout` и одна надпись. При этом разметка (компонент `ConstaintLayout`) принадлежит к классу `ViewGroup`, а надпись — к классу `View`.

Рассмотрим файл разметки (res\layout\activity_main.xml) проекта с двумя кнопками: **Start** и **Stop** (листинг 4.1). На этот раз будет задействована линейная (горизонтальная или вертикальная) разметка.

Рис. 4.2. Иерархия созданных объектов

Листинг 4.1. Файл разметки res\layout\activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?><LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"    android:layout_width="fill_parent"
android:layout_height="fill_parent"    android:weightSum="1">    <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"        android:text="Старт"
android:id="@+id/start" />    <Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"        android:text="Стоп"
android:id="@+id/button" /></LinearLayout>
```

> ***Примечание***
>
> При указании надписей кнопок и других элементов рекомендуется использовать строки ресурсов, а не конкретные строковые значения. Для этого следует в файле *res\values\strings.xml* создать соответствующие строковые значения, а затем задать их в качестве значения атрибута `text`. Однако в листинге 4.1 я прописал обычные строковые значения — чтобы не приводить еще и листинг файла *strings.xml*.

В каждом файле разметки должен быть только один корневой элемент. В нашем случае таким элементом является `LinearLayout` (линейная разметка). После определения корневого элемента вы можете добавить в него дополнительные объекты разметки или виджеты в качестве дочерних элементов.

Рассмотрим атрибуты элемента `LinearLayout`:

- `xmlns:android` — объявление пространства имен Android. Это стандартный атрибут и стандартное значение для Android-приложения;
- `android:layout_width` — ширина объекта `View` или `ViewGroup`. В нашем случае объект занимает весь экран, поэтому используется значение `fill_parent`;
- `android:layout_height` — высота объекта;
- `android:text` — текст, который будет отображен на кнопке.

У каждого объекта `View` или `ViewGroup` свой набор атрибутов. С ними мы познакомимся чуть позже — когда будем рассматривать базовые виджеты.

Установить атрибуты объектов можно как с помощью файла разметки, так и в Java-коде. В большинстве случаев имя атрибута XML соответствует названию метода в классе Java. Например, для установки атрибута `android:height` (задает высоту объекта) используется метод `setHeight(int)`, для установки атрибута `android:text` — метод `setText(int)` и т. д. Но бывают исключения. Так, для установки значения атрибута `android:textColorHighlight` используется метод `setHighlightColor(int)`. Как видите, название метода похоже на название атрибута, но не соответствует ему полностью. Все это говорит о том, что перед написанием Java-кода желательно обратиться к документации и не полагаться только на одну интуицию. О Java-коде мы поговорим позже, пока же рассмотрим типы разметки.

4.1.2. Разные типы разметки

Вы можете использовать один из следующих типов разметки:

- `ConstraintLayout` — разметка привязки;
- `FrameLayout` — разметка фрейма;
- `LinearLayout` — линейная разметка;
- `TableLayout` — табличная разметка;
- `GridLayout` — разметка в виде сетки (устаревший тип разметки);
- `RelativeLayout` — относительная разметка (устаревший тип разметки).

По умолчанию используется разметка `ConstraintLayout`. Вместе с ней можно задействовать направляющие линии — `Guideline` (рис. 4.3). Сами по себе направляющие линии не являются разметкой, и их можно использовать только вместе с `ConstraintLayout`. Направляющие линии бывают вертикальными и горизонтальными. Они помогают расположить элементы интерфейса более точно. Направляющие линии не отображаются на устройстве, а виды только в визуальном редакторе. Аналогичные направляющие линии есть в Visual Studio, а в Android Studio их нужно добавлять самостоятельно.

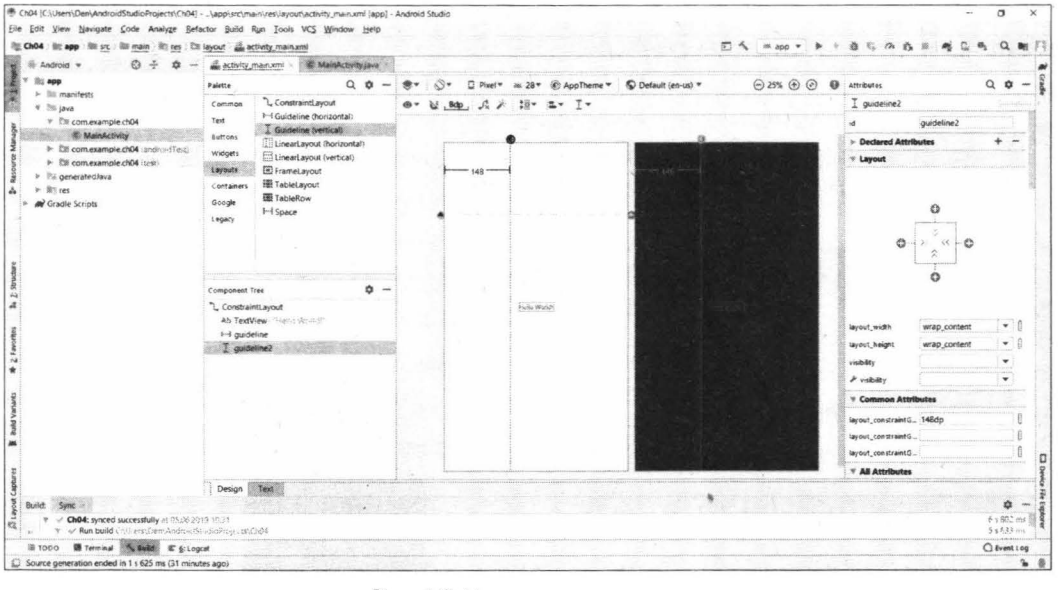


Рис. 4.3. Направляющие линии

Типы разметки `GridLayout` и `RelativeLayout` в приведенном ранее списке помечены как устаревшие, и сейчас — если они вам нужны — их можно найти в разделе **Legacy**.

Разметка `ConstraintLayout`, как уже было отмечено, используется по умолчанию, но для полноты картины нужно рассмотреть и другие способы разметки, в том числе и устаревшие.

Разметка *FrameLayout*

`FrameLayout` — самый простой тип разметки. Все дочерние элементы `FrameLayout` будут прикреплены к верхнему левому углу экрана.

Этот тип разметки настолько примитивен, что даже самые простые приложения ориентируются на другие типы разметки. Единственное применение для `FrameLayout` — это ее использование внутри ячейки таблицы (см. далее), а для создания полноценной разметки приложения эта разметка не годится.

Разметка *LinearLayout*

Линейная разметка (`LinearLayout`) размещает виджеты горизонтально или вертикально, в зависимости от атрибута `android:orientation`:

```
android:orientation="vertical"
```

Посмотрите на рис. 4.4 — на нем задана линейная вертикальная разметка, поэтому кнопки размещаются друг под другом — вертикально. Чтобы задать горизонтальную разметку, нужно изменить атрибут так:

```
android:orientation="horizontal"
```

В листинге 4.1 приведен файл разметки для интерфейса с двумя кнопками, показанного на рис. 4.4. Измените атрибут `android:orientation`, установив значение `horizontal`, чтобы получился такой код:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal" >
```

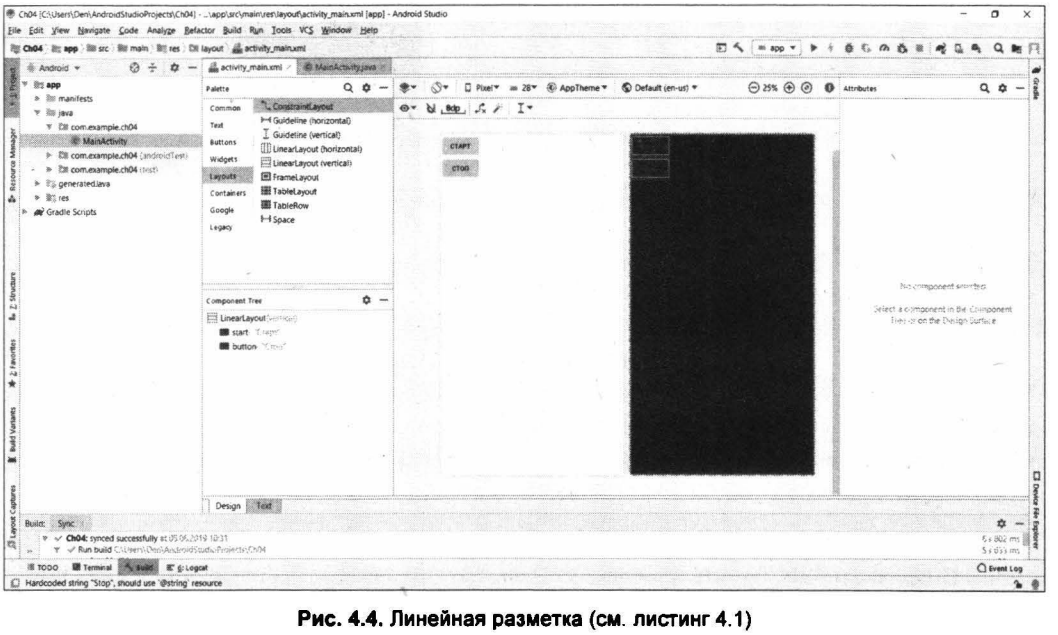


Рис. 4.4. Линейная разметка (см. листинг 4.1)

Затем перейдите на вкладку **Deisgn** или воспользуйтесь областью **Preview**, и вы увидите, что наши кнопки выстроились горизонтально.

Но это еще не все. У разметки `LinearLayout` есть атрибут `android:layout_weight`, задающий «вес» отдельного дочернего элемента. Чем выше «вес», тем важнее элемент. Элемент с максимальным «весом» займет всю оставшуюся часть родительского элемента. По умолчанию вес элемента равен 0. Установите для второй кнопки вес, отличный от 0:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Stop"
    android:id="@+id/button" />
```

В результате последняя кнопка будет растянута на всю оставшуюся ширину или высоту экрана — в зависимости от типа разметки (рис. 4.5).

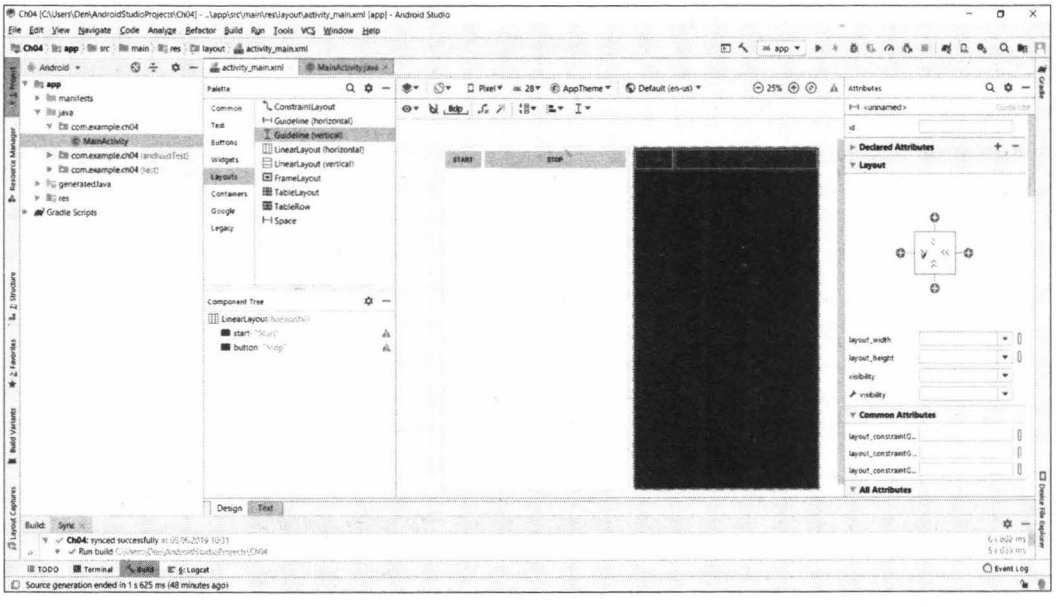


Рис. 4.5. Кнопка растянута на всю оставшуюся ширину экрана (для случая горизонтальной разметки)

Разметка *TableLayout*

Разметка `TableLayout` размещает виджеты в строки и столбцы таблицы. Границы самой таблицы не отображаются, поскольку таблица здесь используется не для представления данных, а сугубо для размещения виджетов.

Таблица может иметь строки с разным количеством ячеек. Для формирования строк таблицы разметки используются объекты `TableRow`. Каждый такой объект — это одна строка таблицы. В строке может вообще не быть ячеек, может быть одна

или несколько ячеек. В свою очередь ячейка может быть объектом класса `ViewGroup` — другими словами, ячейка допускает разметку. То есть, в ячейку таблицы вы можете поместить объект `LinearLayout` и с его помощью разместить элементы внутри ячейки. Можно также задействовать в качестве элемента ячейки другой объект `TableLayout` — получится вложенная таблица.

Сейчас мы продемонстрируем использование табличной разметки на примере создания клавиатуры для простейшего калькулятора. Создайте новый проект или откройте наш простой проект, разработанный в *главе 2*. Перейдите к редактору разметки (для этого откройте в структуре проекта файл **\res\layout\activity_main.xml** и щелкните на нем двойным щелчком). Затем перейдите на вкладку `activity_main.xml` и удалите из XML-файла все, кроме элемента `<?xml>`.

Теперь создайте разметку — для этого в группе `Layouts` выберите `TableLayout`. В эту разметку требуется добавить четыре элемента `TableRow`. Для каждого элемента `TableRow` установите следующие атрибуты:

- `Gravity = center`;
- `Layout width = fill_parent`;
- `Layout height = wrap_content`.

Свойство (атрибут) `Gravity`, определение которого показано на рис. 4.6, задает выравнивание элемента в контейнере, — мы устанавливаем с его помощью выравнивание по центру: `[center]`.

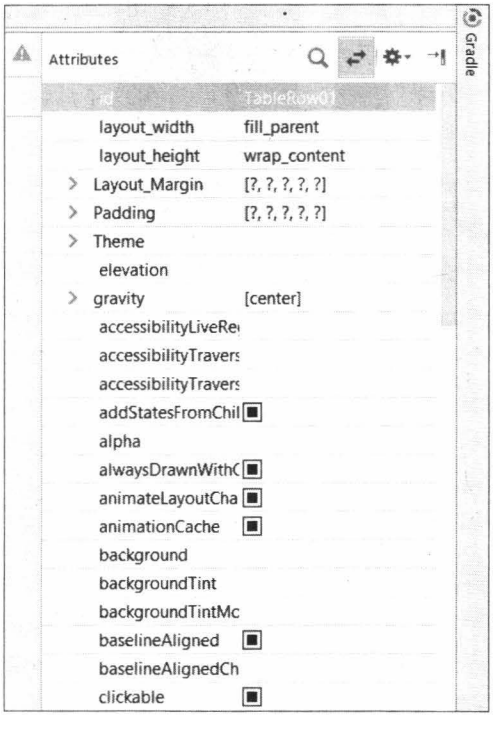


Рис. 4.6. Установка значения атрибута `Gravity`

Далее в каждую строку (в каждый контейнер `TableRow`) надо добавить по четыре кнопки (кнопки находятся в группе `Widgets`) и установить ширину каждой кнопки в 20 pt (это значение зависит от типа платформы Android и от размера экрана). Наша клавиатура готова (рис. 4.7) — обратите внимание на иерархию графического интерфейса (**Component Tree**).

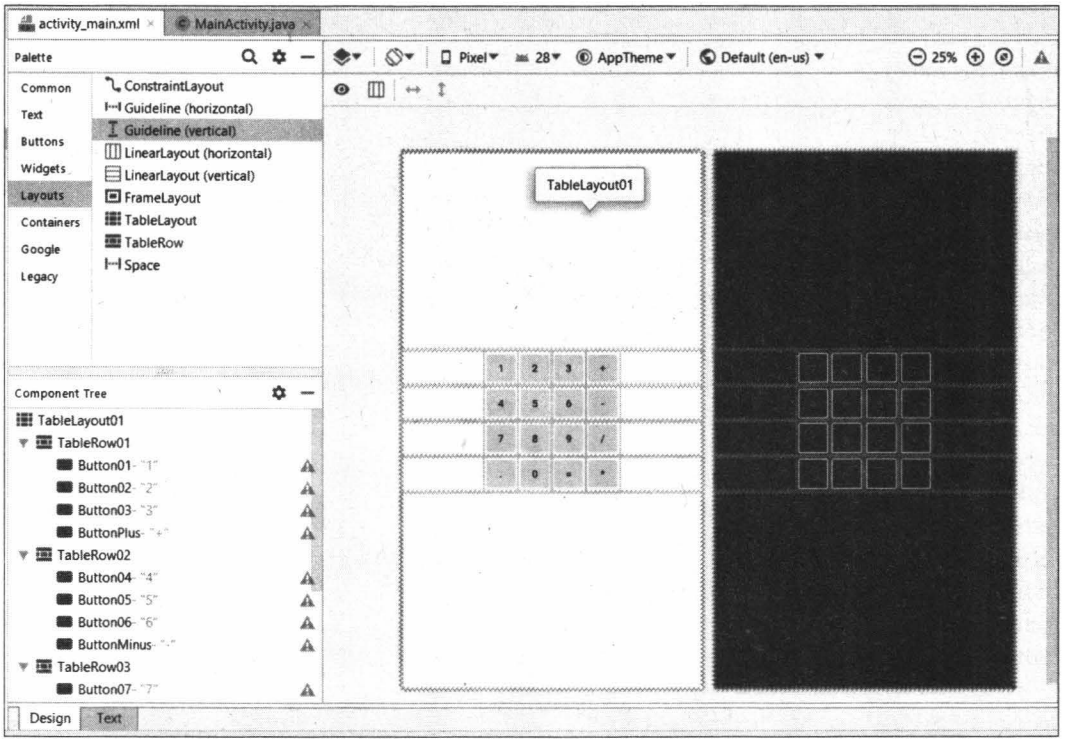


Рис. 4.7. Созданная клавиатура калькулятора

В XML-файле разметка `TableLayout` будет объявлена так (листинг 4.2).

Листинг 4.2. Полный код XML-файла разметки клавиатуры калькулятора

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/TableLayout01"
android:layout_width="fill_parent"
android:gravity="center"
android:layout_height="fill_parent">
<TableRow
android:id="@+id/TableRow01"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
```

```
<Button
android:id="@+id/Button01"
android:layout_height="wrap_content"
android:text="1"
android:layout_width="20pt"/>
<Button
android:id="@+id/Button02"
android:layout_height="wrap_content"
android:text="2"
android:layout_width="20pt"/>
<Button
android:id="@+id/Button03"
android:layout_height="wrap_content"
android:text="3"
android:layout_width="20pt"/>
<Button
android:id="@+id/ButtonPlus"
android:layout_height="wrap_content"
android:text="+"
android:layout_width="20pt"/>
</TableRow>
<TableRow
android:id="@+id/TableRow02"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button04"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="4"/>
<Button
android:id="@+id/Button05"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="5"/>
<Button
android:id="@+id/Button06"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="6"/>
<Button
android:id="@+id/ButtonMinus"
android:layout_height="wrap_content"
android:text="-"
android:layout_width="20pt"/>
</TableRow>
```

```
<TableRow
android:id="@+id/TableRow03"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button07"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="7"/>
<Button
android:id="@+id/Button08"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="8"/>
<Button
android:id="@+id/Button09"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="9"/>
<Button
android:id="@+id/ButtonDivide"
android:layout_height="wrap_content"
android:text="/"
android:layout_width="20pt"/>
</TableRow>
<TableRow
android:id="@+id/TableRow04"
android:layout_height="wrap_content"
android:layout_width="fill_parent"
android:gravity="center">
<Button
android:id="@+id/Button10"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="."/>
<Button
android:id="@+id/Button11"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="0"/>
<Button
android:id="@+id/Button12"
android:layout_height="wrap_content"
android:layout_width="20pt"
android:text="="/>
```

```
<Button
android:id="@+id/ButtonMul"
android:layout_height="wrap_content"
android:text="*"
android:layout_width="20pt"/>
</TableRow>
</TableLayout>
```

Когда вы вникнете в структуру XML-файла, то обнаружите, что редактировать разметку интерфейса вручную даже проще, чем использовать интерфейс Android Studio. Во всяком случае, создать разметку для Android-приложения вручную не сложнее, чем создать HTML-страницу. Я предпочитаю создавать разметку вручную, а устанавливать некоторые свойства — уже с помощью редактора разметки, поскольку не всегда удается запомнить все необходимые свойства и их значения.

Разметка *GridLayout*

Этот тип разметки впервые появился в Android 4 и очень похож на `TableLayout`. Разметка относится к классу `android.widget.GridLayout` и имеет колонки, ряды, клетки как в `TableLayout`, но при этом элементы могут гибко настраиваться. Получается, что новый тип разметки гораздо удобнее `TableLayout`.

Чтобы попробовать `GridLayout` на практике, давайте воссоздадим с его помощью клавиатуру нашего калькулятора. Посмотрите, насколько элегантнее стал код (листинг 4.3). Сравните его с кодом из листинга 4.2.

Листинг 4.3. Клавиатура калькулятора с использованием `GridLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:columnCount="4"
    android:orientation="horizontal" >
    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="+" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="-" />
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button android:text="/" />
    <Button android:text="." />
    <Button android:text="0" />
```

```
<Button android:text="=" />
<Button android:text="*" />
```

Результат этой разметки показан на рис. 4.8.

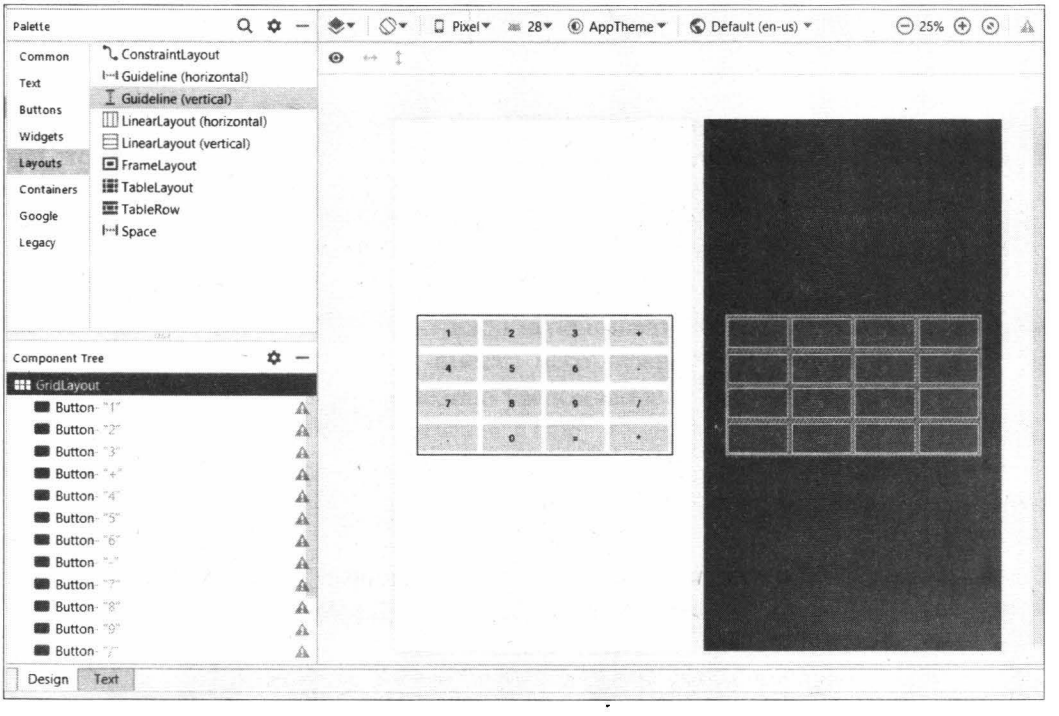


Рис. 4.8. Клавиатура калькулятора с использованием разметки `GridLayout`

Обратите внимание: мы просто определили, сколько у нас будет столбцов (`android:columnCount="4"`), а потом указали элементы, которые нужно поместить в столбцы.

Но это слишком тривиальный пример. С помощью `GridLayout` можно создавать и более сложные варианты разметки. Посмотрите на листинг 4.4.

Листинг 4.4. Более сложная клавиатура калькулятора

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:columnCount="4"
    android:orientation="horizontal" >

    <Button
        android:layout_column="3"
        android:text="*" />
```

```
    <Button android:text="1" />
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="/" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="-" />
    <Button android:text="7" />
    <Button android:text="8" />
    <Button android:text="9" />
    <Button
        android:layout_rowSpan="3"
        android:text="+" />
    <Button
        android:layout_columnSpan="2"
        android:text="0" />
    <Button android:text="^2" />
    <Button
        android:layout_columnSpan="3"
        android:text="=" />
</GridLayout>
```

Результат этой разметки показан на рис. 4.9. С помощью `TableLayout` организовать что-либо подобное тоже можно, но код будет более громоздким.

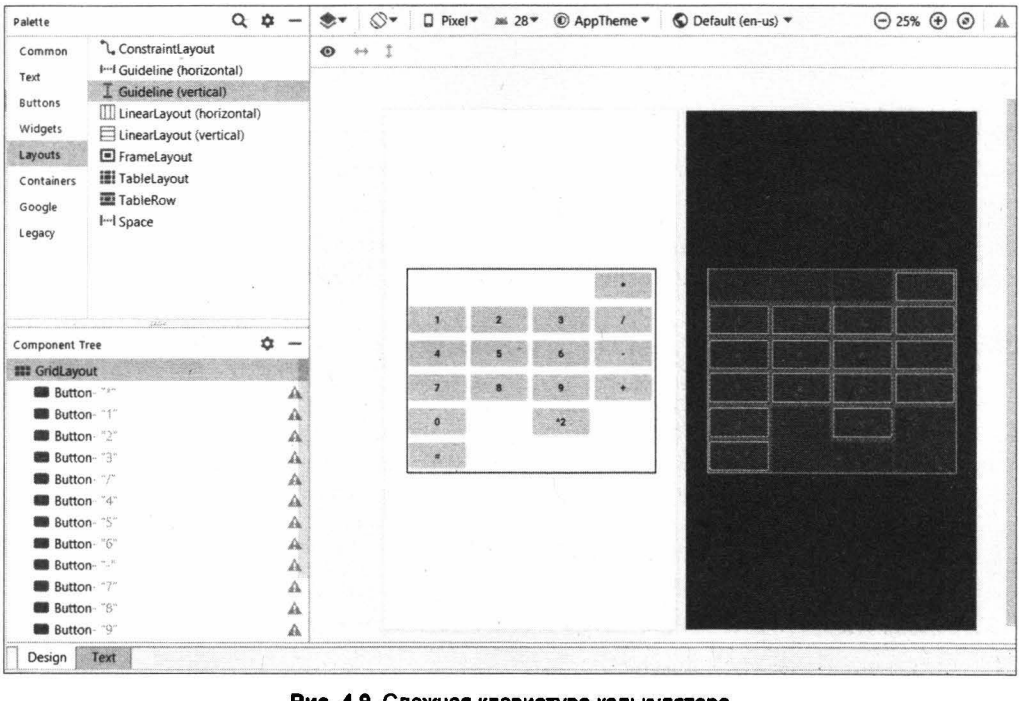


Рис. 4.9. Сложная клавиатура калькулятора

Разметка *RelativeLayout*

При относительной разметке (`RelativeLayout`) позиция дочерних виджетов определяется относительно родительского элемента. При использовании этого типа разметки элементы графического интерфейса расположены так, что если первый элемент расположен по центру, другие элементы, выравнивание которых задается относительно первого элемента, будут также выровнены по центру экрана.

> ***Внимание!***
>
> **К настоящему моменту относительная разметка считается устаревшей, и лучше избегать ее использования.**

Лучше всего продемонстрировать относительную разметку на примере. На рис. 4.10 показан интерфейс пользователя — надпись, поле ввода и две кнопки. Обратите внимание на иерархию интерфейса. Файл разметки интерфейса представлен в листинге 4.5.

Листинг 4.5. Файл разметки интерфейса, показанного на рис. 4.10

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Ваше имя:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Отмена" />
</RelativeLayout>
```

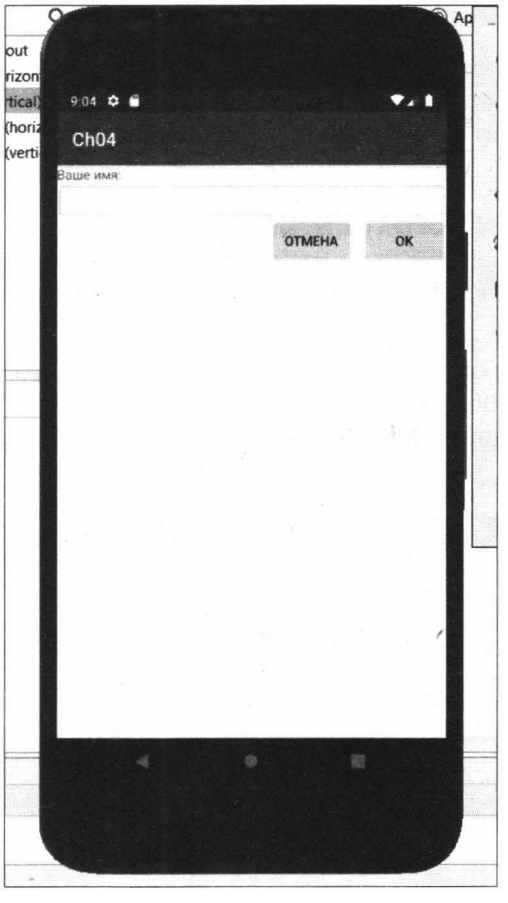


Рис. 4.10. Пример относительной разметки (в эмуляторе)

Разметка *ConstraintLayout*

Для позиционирования элемента внутри `ConstraintLayout` необходимо указать ограничения (constraints). Есть несколько типов ограничений. В частности, для установки позиции относительно определенного элемента используются следующие ограничения (привязки):

- `layout_constraintLeft_toLeftOf` — левая граница позиционируется относительно левой границы другого элемента;
- `layout_constraintLeft_toRightOf` — левая граница позиционируется относительно правой границы другого элемента;
- `layout_constraintRight_toLeftOf` — правая граница позиционируется относительно левой границы другого элемента;
- `layout_constraintRight_toRightOf` — правая граница позиционируется относительно правой границы другого элемента;
- `layout_constraintTop_toTopOf` — верхняя граница позиционируется относительно верхней границы другого элемента;

- layout_constraintBottom_toBottomOf — нижняя граница позиционируется относительно нижней границы другого элемента;
- layout_constraintBaseline_toBaselineOf — базовая линия позиционируется относительно базовой линии другого элемента;
- layout_constraintTop_toBottomOf — верхняя граница позиционируется относительно нижней границы другого элемента;
- layout_constraintBottom_toTopOf — нижняя граница позиционируется относительно верхней границы другого элемента;
- layout_constraintStart_toEndOf — аналог layout_constraintLeft_toRightOf;
- layout_constraintStart_toStartOf — аналог layout_constraintLeft_toLeftOf;
- layout_constraintEnd_toStartOf — аналог layout_constraintRight_toLeftOf;
- layout_constraintEnd_toEndOf — аналог layout_constraintRight_toRightOf.

Позиционирование может производиться относительно границ самого контейнера ContentLayout (в этом случае ограничение имеет значение "parent") либо же относительно любого другого элемента внутри ConstraintLayout, тогда в качестве значения ограничения указывается id этого элемента.

Чтобы указать отступы от элемента, относительно которого производится позиционирование, применяются следующие атрибуты:

- android:layout_marginLeft — отступ от левой границы;
- android:layout_marginRight — отступ от правой границы;
- android:layout_marginTop — отступ от верхней границы;
- android:layout_marginBottom — отступ от нижней границы;
- android:layout_marginStart — отступ от левой границы;
- android:layout_marginEnd — отступ от правой границы.

В качестве примера рассмотрим листинг 4.6.

Листинг 4.6. Пример разметки ConstraintLayout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, world"
```

```
        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="30dp"

        android:layout_marginLeft="80dp"
        app:layout_constraintLeft_toLeftOf="parent" />

</android.support.constraint.ConstraintLayout>
```

Здесь устанавливаются два ограничения относительно родительского контейнера `ConstraintLayout`, поэтому ограничения имеют значение `"parent"`. Соответственно, все отступы, которые определены у элемента `TextView`, устанавливаются относительно верхнего и левого краев контейнера (рис. 4.11).

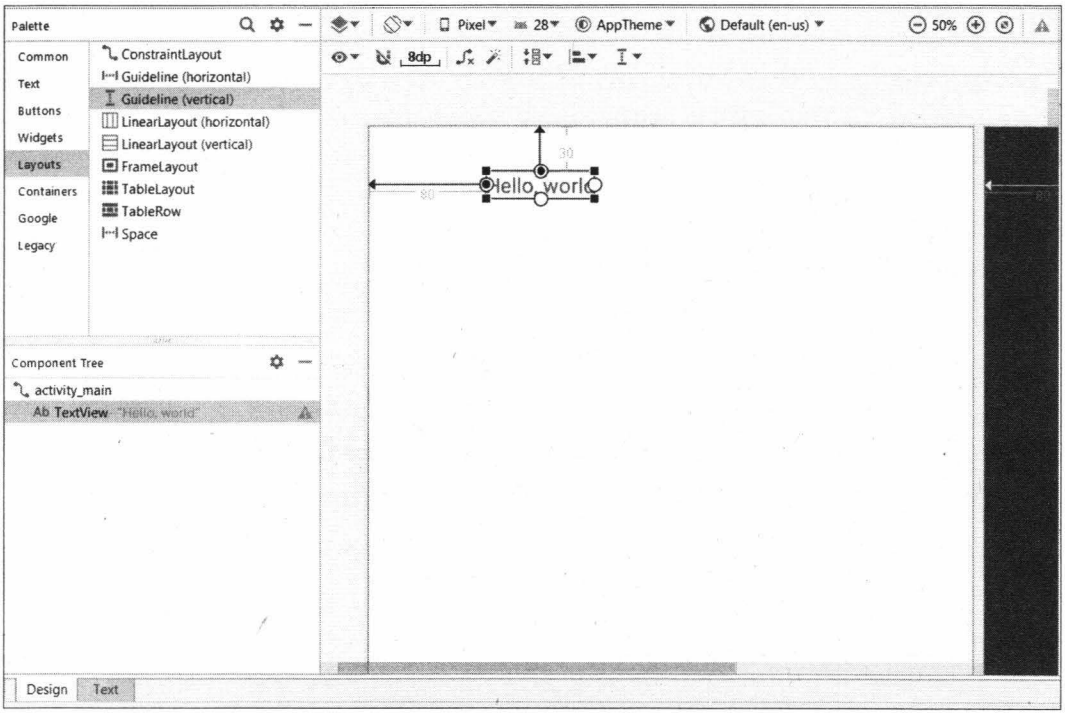


Рис. 4.11. Разметка `ConstraintLayout`

Причем сами по себе отступы ничего не дадут — нам обязательно надо установить ограничение, которое и будет указывать, относительно какого элемента идет отступ.

Рассмотрим более сложный пример (листинг 4.7), его разметка показана на рис. 4.12.

Примечание

Текст в листинге 4.7 сгенерирован с помощью генератора так называемого «рыба текста» (lorem ipsum) и не несет никакой смысловой нагрузки.

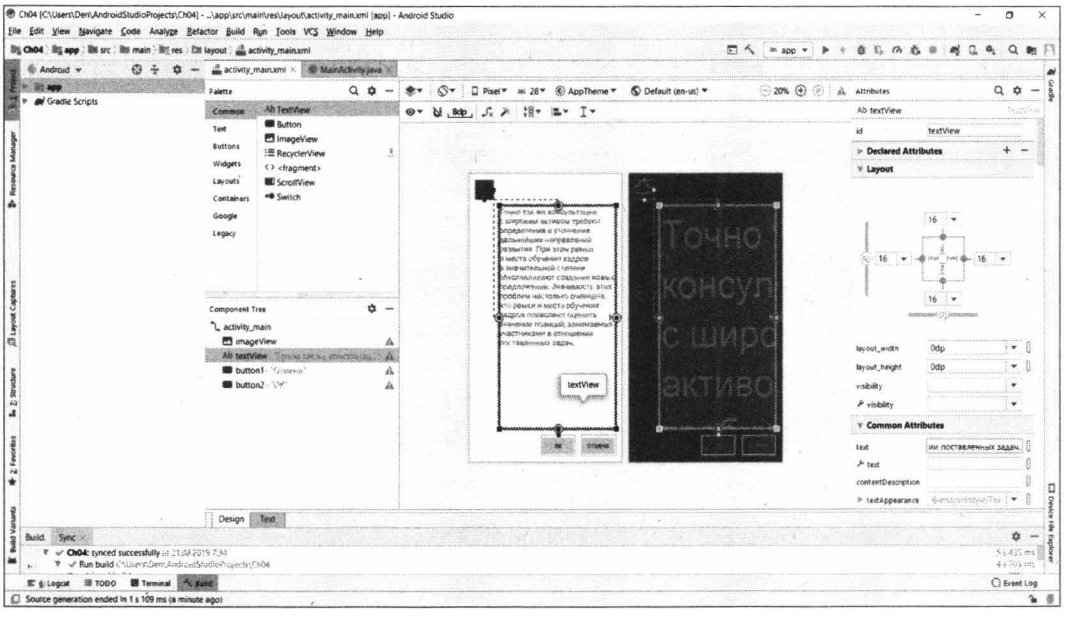


Рис. 4.12. Разметка из листинга 4.7

Листинг 4.7. Сложный пример `ConstraintLayout`

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    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:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:text="TextView"
        android:background="#3F51B5"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:id="@+id/imageView"

        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginLeft="16dp"

        app:layout_constraintTop_toTopOf="parent"
        android:layout_marginTop="16dp" />

    <TextView
        android:layout_width="0dp"
```

```
        android:layout_height="0dp"
        android:textSize="20sp"
        android:id="@+id/textView"
        android:text="Точно так же консультации с широким активом требуют
          определения и уточнения дальнейших направлений развития. При этом
          рамки и места обучения кадров в значительной степени обуславливают
          создание новых предложений. Значимость этих проблем настолько
          очевидна, что рамки и места обучения кадров позволяют оценить
          значение позиций, занимаемых участниками в отношении поставленных
          задач."
         app:layout_constraintLeft_toRightOf="@+id/imageView"
        android:layout_marginLeft="16dp"

        app:layout_constraintTop_toBottomOf="@+id/imageView"
        android:layout_marginTop="16dp"

        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"

        app:layout_constraintBottom_toTopOf="@+id/button2"
        android:layout_marginBottom="16dp" />

    <Button
        android:text="Отмена"
        android:layout_width="93dp"
        android:layout_height="53dp"
        android:id="@+id/button1"

        app:layout_constraintRight_toRightOf="parent"
        android:layout_marginRight="16dp"

        app:layout_constraintBottom_toBottomOf="parent"
        android:layout_marginBottom="16dp" />

    <Button
        android:text="OK"
        android:layout_width="93dp"
        android:layout_height="53dp"
        android:id="@+id/button2"

        app:layout_constraintRight_toLeftOf="@+id/button1"
        android:layout_marginRight="16dp"
        app:layout_constraintBaseline_toBaselineOf="@+id/button1" />
</android.support.constraint.ConstraintLayout>
```

Обратите в этом листинге внимание на некоторые моменты. Во-первых, здесь элемент позиционируется относительно контейнера `ConstraintLayout`: от верхней и левой границ контейнера до соответствующих границ `ImageView` задано по 16 dp.

Во-вторых, относительно контейнера позиционируется и кнопка с `id=button1`: от правой и нижней границ контейнера до соответствующих границ `Button` также по 16 dp.

В-третьих, вторая кнопка с `id=button2` позиционируется относительно первой кнопки: от правой границы второй кнопки до левой границы первой кнопки (`app:layout_constraintRight_toLeftOf="@+id/button1"`) также 16 dp. И чтобы обе кнопки находились на одном уровне, им задано выравнивание по базовой линии: `app:layout_constraintBaseline_toBaselineOf="@+id/button1"`.

Элемент `TextView` с фрагментом текста позиционируется сразу относительно контейнера, элемента `ImageView` и второй кнопки.

Преимущество такой разметки в том, что она будет действительна и при альбомной ориентации (рис. 4.13).

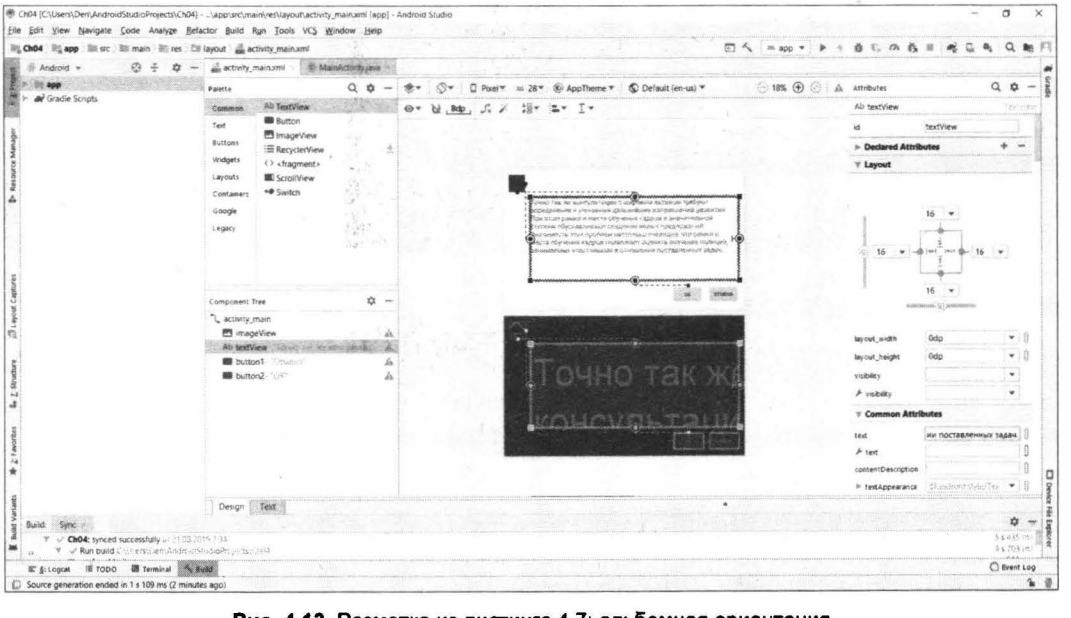


Рис. 4.13. Разметка из листинга 4.7: альбомная ориентация

4.2. Основные виджеты графического интерфейса

Виджеты, как нам уже известно, — это элементы графического интерфейса. Ранее мы уже познакомились с двумя виджетами: текстовым полем и кнопкой. Сейчас мы продолжим наше знакомство с виджетами.

Создайте новый проект с именем Ch4, деятельность этого проекта будет называться MainActivity, а пакет — com.example.<имя>.ch4 — так вам будет понятнее, откуда взялись некоторые идентификаторы.

4.2.1. Текстовые поля

В Android вы можете использовать два основных текстовых поля: TextView и EditText (оба находятся на вкладке **Text Fields** редактора разметки). Первое служит для отображения текста без возможности его редактирования, а второе — это классическое текстовое поле с возможностью ввода и редактирования текста. Есть множество и других полей, основанных на этих, — например, Password (используется для ввода пароля, при этом вводимые пользователем символы не отображаются), Password (Numeric) — для ввода числового пароля, Date — для ввода даты, Time — для ввода времени, E-mail — для ввода адреса электронной почты и т. п.

Несмотря на свою простоту, виджет TextView весьма часто задействуется в Android-программах для вывода инструкций по использованию программы и другого текста.

В файле разметки значение TextView можно установить так:

```
android:text="Text";
```

В Java-коде значение виджета устанавливается так:

```
TextView text = (TextView)findViewById(R.id.text1);
text.setText("Hello");
```

Совсем другое дело, если вы планируете создать приложение с многоязыковой поддержкой пользовательского интерфейса. Тогда непосредственно в файле разметки значение (текстовую строку) указывать не нужно. Вместо этого создается ссылка на текстовый XML-ресурс:

```
android:text="@string/str_value"
```

Здесь str_value — это имя строкового ресурса, описанного в файле strings.xml. В Java-коде установить имя ресурса можно тем же методом setText():

```
TextView text = (TextView)findViewById(R.id.text1);
text.setText(R.string.str_value);
```

У элемента TextView есть много методов и свойств. Мы рассмотрим только основные свойства, относящиеся к отображению текста.

- Размер шрифта можно задать свойством android:textSize. Размер задается в пикселах (px), независимых от плотности пикселах (dp), независимых от масштабирования пикселах (sp), пунктах (pt), дюймах (in) и миллиметрах (mm):

  ```
  android:textSize="14pt";
  ```

- Стиль текста задается свойством android:textStyle:
  - normal — обычное начертание символов;
  - bold — полужирное начертание символов;
  - italic — курсив.

Пример:

```
android:textStyle="italic"
```

- Цвет шрифта задается свойством `android:textColor`. Цвет указывается в шестнадцатеричной кодировке в формате #RGB или #ARGB. Во втором случае значение A — это прозрачность. Если A = 0, то прозрачность 100% (элемент практически не будет виден).

Теперь немного практики. Создайте новый проект. По умолчанию будет создан проект, выводящий строку "Hello World!" с помощью `TextView` — как раз то, что нам и нужно.

Перейдите к файлу разметки проекта res\layout\main.xml. Попробуем изменить свойства `TextView`. Если вы забыли, как называется то или иное свойство, — Android Studio это вам подскажет: наберите `android:` и немного подождите — откроется выпадающий список, из которого вам нужно будет выбрать нужное свойство (рис. 4.14).

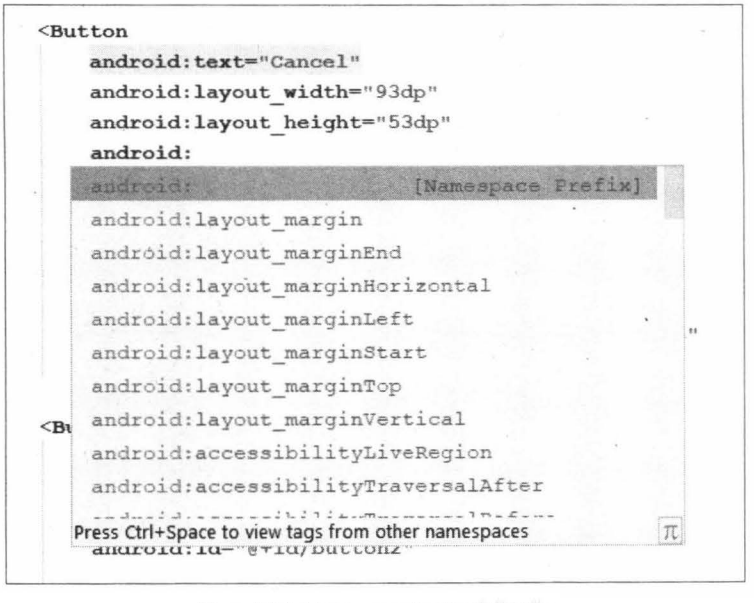


Рис. 4.14. Подсказка Android Studio

Установим свойства `textSize` и `textStyle`, как показано в листинге 4.8. Кроме этого мы также присвоили имя нашему текстовому полю (`txt1`), чтобы к нему можно было обратиться из Java-кода.

Листинг 4.8. Файл activity_main.xml: установка атрибутов `TextView`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/txt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="Hello"
    />
</LinearLayout>
```

Результат внесенных в код изменений показан на рис. 4.15.

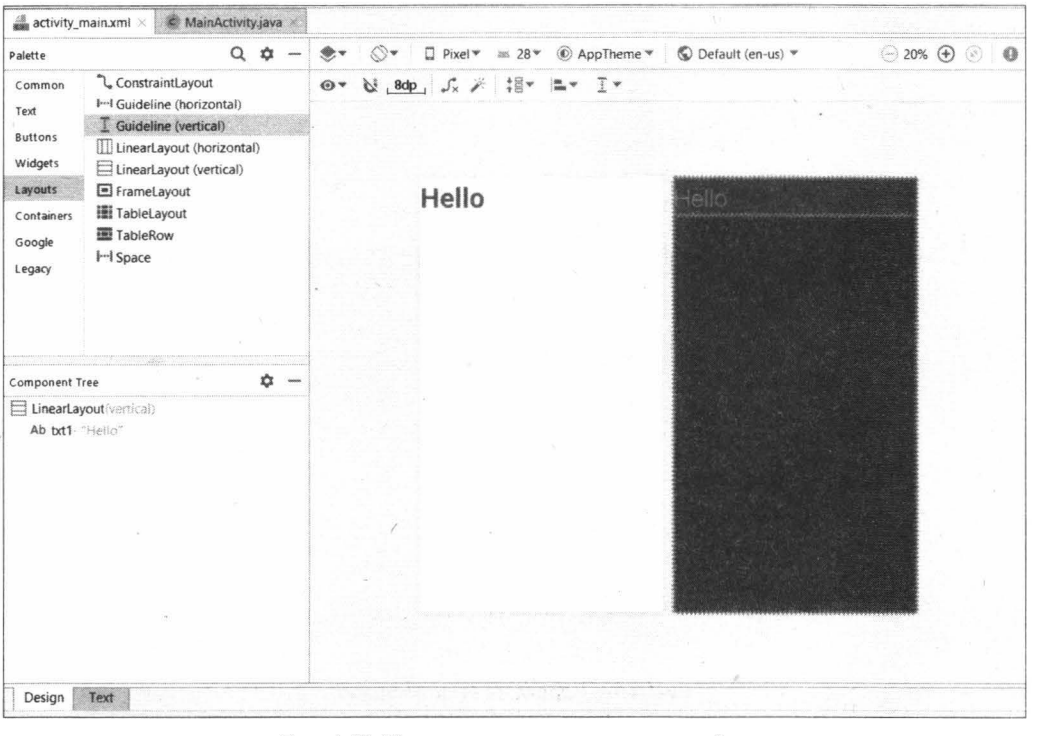


Рис. 4.15. Изменение размера и стиля шрифта

Теперь попробуем усложнить нашу задачу и изменить текст нашего виджета из Java-кода. Для этого откройте файл, содержащий Java-код нашего приложения: <название пакета>\MainActivity.java. В моем случае этот файл называется com.example.den.ch4\MainActivity.java. На рис. 4.16 показано его содержимое по умолчанию, а также область навигации, объясняющая, как до него «добраться».

Измените Java-файл так, чтобы он выглядел, как показано в листинге 4.9. Строки, которые нужно добавить в ваш Java-файл, выделены полужирным шрифтом.

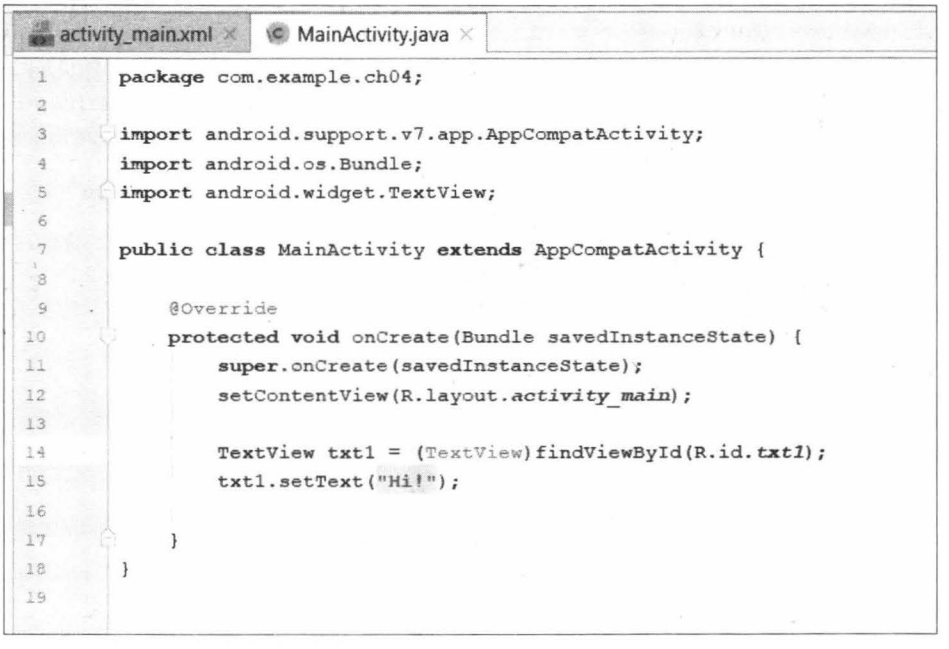


Рис. 4.16. Файл MainActivity.java

Листинг 4.9. Файл MainActivity.java

```
package com.example.ch04;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView txt1 = (TextView)findViewById(R.id.txt1);
        txt1.setText("Hi!");

    }
}
```

Теперь запустите приложение. В качестве значения текстового поля будет установлена строка "Hi!" (рис. 4.17).

Текстовое поле `EditText` позволяет вводить и редактировать текст. Основной метод этого поля — метод `getText()`, позволяющий получить введенный пользователем

текст. Значение, возвращаемое методом `getText()`, имеет тип `Editable`. По сути, `Editable` — это надстройка над `String`, но в отличие от него, значение типа `Editable` может быть изменено в процессе выполнения программы (`String` является неизменяемым типом — при изменении значения типа `String` попросту создается новый экземпляр `String`, содержащий новое значение).

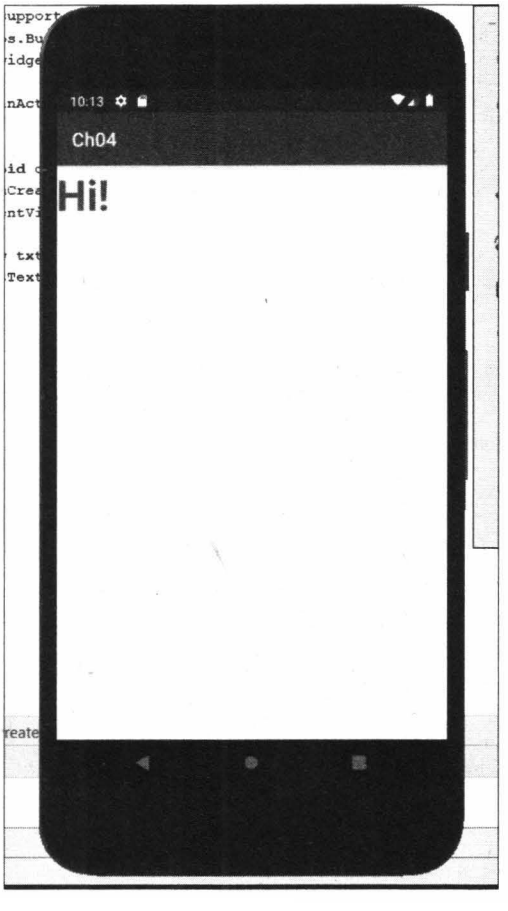


Рис. 4.17. Результат работы программы из листинга 4.9

Кроме метода `getText()` вам может понадобиться метод `selectAll()`, выделяющий весь текст в окне. Если весь текст выделять не нужно, можно использовать метод `setSelection()`:

```
setSelection(int start, int stop)
```

Этот метод выделяет участок текста, начиная с позиции `start` до позиции `stop`.

Установить тип начертания шрифта можно с помощью метода `setTypeface` — например:

```
txt1.setTypeface(null, Typeface.NORMAL);
```

Вместо `NORMAL` можно указать `BOLD` и `ITALIC`. Для добавления поля `EditText` в разметку окна нужно добавить следующий код в файл разметки:

```
<EditText
    android:id="@+id/entry1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Enter text"/>
```

Далее мы научимся работать с этим полем — займемся получением и установкой текста поля. А пока ограничимся только тем, что добавим поле в файл разметки. Как выглядит это поле, вы уже видели ранее.

4.2.2. Кнопки

Кнопки — очень важные элементы пользовательского интерфейса, поскольку к ним относятся не только непосредственно сами кнопки, но и переключатели (независимые и зависимые). Кнопки определяются пятью классами: `Button`, `CheckButton`, `ToggleButton`, `RadioButton`, `ImageButton`.

Классы `CheckButton`, `ToggleButton` и `RadioButton` являются потомками класса `CompoudButton`, который, в свою очередь, является потомком класса `Button`. А вот класс `ImageButton` является потомком класса `ImageView`, поэтому `ImageButton` — это больше изображение, нежели кнопка в прямом понимании этого слова.

Button — обычная кнопка

Начнем с обычной кнопки и продемонстрируем создание относительно простого приложения. Java-кода в этом приложении будет чуть больше, чем обычно, поэтому читателям, не знакомым с Java, нужно быть внимательными, чтобы не допустить ошибку.

Добавьте в наш проект, демонстрирующий изменение значения текстового поля с помощью Java-кода (см. листинг 4.9), одну кнопку. Теперь изменение значения текстового поля будет происходить не при запуске приложения, а при нажатии на кнопку.

Графическая разметка проекта показана на рис. 4.18, а XML-разметка представлена в листинге 4.10.

Листинг 4.10. Разметка проекта с кнопкой

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/txt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="Hello"
    />

<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />

</LinearLayout>
```

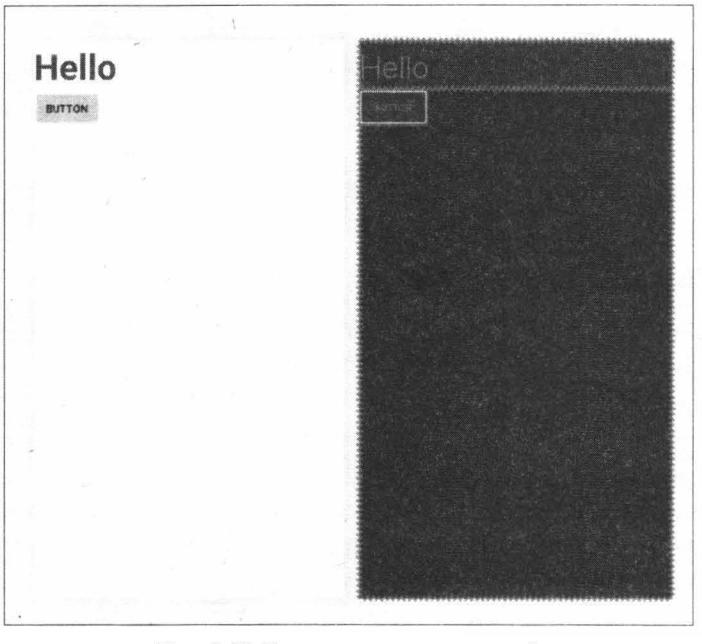


Рис. 4.18. Разметка проекта с кнопкой

Обратите внимание на идентификаторы текстового поля и кнопки. Текстовое поле называется `txt1`, а кнопка — `button1`. Эти идентификаторы мы будем использовать и в Java-коде. Кстати, приятно, когда Android Studio пишет код за вас — автодополнение кода очень удачно реализовано в этой среде. Java-код нашего приложения приведен в листинге 4.11.

Листинг 4.11. Пример установки обработчика события кнопки

```
package com.example.ch04;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
import android.view.View;
import android.widget.TextView;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button button1 = (Button)findViewById(R.id.button1);

        button1.setOnClickListener(new View.OnClickListener() {

            public void onClick(View v) {
                final TextView txt1;
                txt1 = (TextView) findViewById(R.id.txt1);
                txt1.setText("Hi!");
            }
        });


    }
}
```

Обработчик нажатия кнопки устанавливается методом `setOnClickListener`. Далее с помощью `new View.OnClickListener()` создается код нового обработчика, который просто указывается ниже. В самом обработчике мы находим текстовое поле `txt1` и устанавливаем для него новый текст. При запуске приложения вы увидите надпись **Hello** и кнопку. По нажатию кнопки текст надписи будет изменен на **Hi!**.

Думаю, как установить обработчик нажатия кнопки, понятно. Однако код получился весьма громоздкий. Скажем, когда у вас одна или две кнопки, то такой код — это не проблема. А вот когда у вас 5 кнопок (или больше), то очень легко запутаться — уж очень много скобок. Хотя Android Studio делает все возможное, чтобы вам помочь, есть один способ уменьшить код и существенно упростить его для восприятия.

Представим, что вы в файле разметки объявили пять кнопок с именами от `button1` до `button5`. Сначала нужно найти кнопки:

```
final Button button1 = (Button)findViewById(R.id.button1);
final Button button2 = (Button)findViewById(R.id.button2);
...
final Button button5 = (Button)findViewById(R.id.button5);
```

Потом устанавливаем один обработчик для всех кнопок:

```
button1.setOnClickListener(this);
button2.setOnClickListener(this);
...
button5.setOnClickListener(this);
```

Затем анализируем, какая кнопка была нажата, и выполняем соответствующее действие:

```
@Override
public void onClick(View v) {
  switch (v.getId()) {
     case R.id.button1: txt1.setText("Button 1"); break;
     case R.id.button2: txt1.setText("Button 2"); break;
     case R.id.button3: txt1.setText("Button 3"); break;
     case R.id.button4: txt1.setText("Button 4"); break;
     case R.id.button5: txt1.setText("Button 5"); break;
  }
}
```

Существует еще один, более компактный, способ установки обработчика нажатия кнопки. В XML-разметке с помощью атрибута `onClick` нужно указать имя метода-обработчика:

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click"
        android:onClick="sendMessage" />
```

Далее в коде просто объявить метод `sendMessage()`:

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // Обработка нажатия кнопки
    public void sendMessage(View view) {
        TextView textView = (TextView) findViewById(R.id.textView);
        EditText editText = (EditText) findViewById(R.id.editText);
        textView.setText("Hello, " + editText.getText());
    }
}
```

Какой метод использовать — решать вам. На мой взгляд, последний способ более компактный — если нужно установить код не для группы кнопок, а только для одной кнопки.

RadioButton — зависимые переключатели

Зависимые переключатели (виджеты класса `RadioButton`) используются внутри контейнера `RadioGroup` — группы зависимых переключателей. Зависимый переключатель позволяет выбрать только одну из опций в одной из групп переключателей. В одной из групп может быть активным только один переключатель.

Вы в основном будете пользоваться тремя методами:

- `toggle()` — инвертирует состояние зависимого переключателя;
- `isChecked()` — возвращает значение зависимого переключателя (true — активен);
- `setChecked()` — изменяет значение переключателя в зависимости от переданного параметра.

Для демонстрации работы с зависимыми переключателями добавьте в наш проект с кнопкой и текстовым полем два зависимых переключателя с именами `r1` и `r2`. По нажатию на кнопку значение текстового поля будет изменено в зависимости от выбранного переключателя. Если ни один из переключателей не выбран, тогда значение `TextView` изменено не будет. Разметка проекта приведена в листинге 4.12.

Листинг 4.12. Файл разметки проекта (зависимые переключатели)

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
<RadioButton
        android:text="Radiobox 1"
        android:id="@+id/r1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</RadioButton>
<RadioButton android:text="Radiobox 2"
        android:id="@+id/r2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</RadioButton>
<TextView
    android:id="@+id/txt1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textSize="20pt"
    android:textStyle="bold"
    android:text="Привет"
    />
```

```
<Button
        android:text="OK"
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
</Button>
</RadioGroup>
```

Как видите, все достаточно просто. Готовое приложение показано на рис. 4.19 (приложение запущено) и 4.20 (выбрано значение `Radiobox 2` и нажата кнопка).

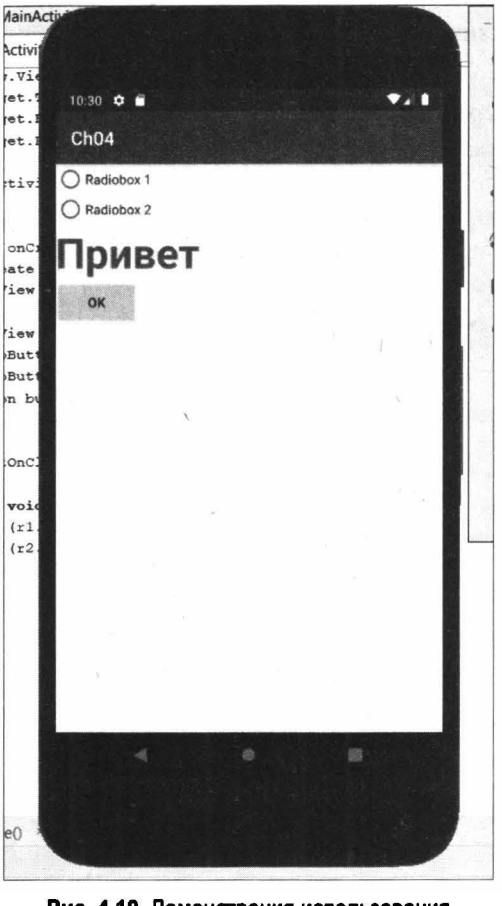


Рис. 4.19. Демонстрация использования зависимых переключателей: приложение запущено

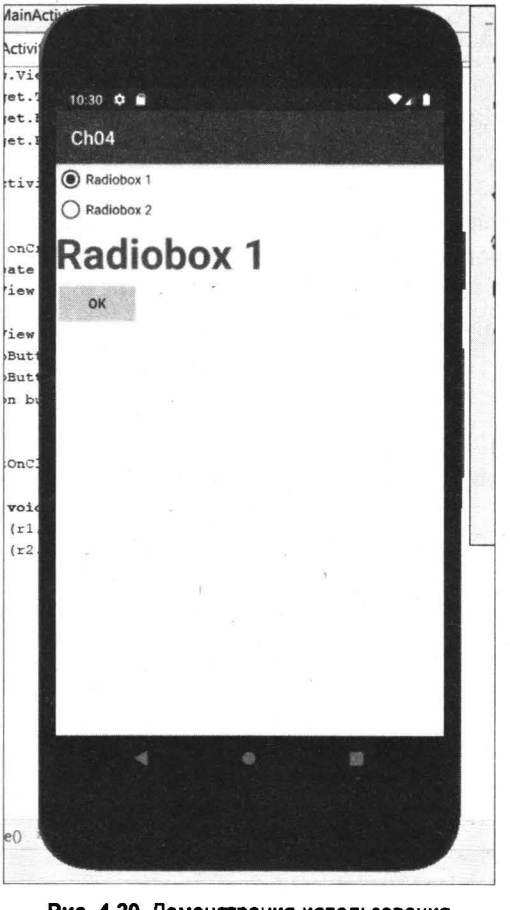


Рис. 4.20. Демонстрация использования зависимых переключателей: выбрано значение `Radiobox 2` и нажата кнопка

CheckBox — независимые переключатели

Переключатели `CheckBox` не привязываются к какому-либо контейнеру (вроде `RadioGroup`), их значение не зависит от состояния других переключателей, поэтому они и называются независимыми.

Для работы с CheckBox вы можете использовать те же методы (isChecked(), toggle(), setChecked()), что и в случае с RadioButton. В файле разметки независимые переключатели определяются так:

```
<CheckBox
  android:id="@+id/checkbox"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Yes"/>
```

Найти CheckBox в Java-коде можно так:

```
final CheckBox check = (CheckBox)findViewById(R.id.checkbox);
```

Далее можно проверить, включен ли переключатель:

```
if (check.isChecked()) txt1.setText("OK");
```

ToggleButton — кнопка включено/выключено

Кнопка ToggleButton может находиться в одном из двух положений: включено или выключено. Когда кнопка нажата, горит зеленый индикатор, как бы встроенный в кнопку, также видно, что кнопка немного вдавлена.

По большому счету от ToggleButton можно было отказаться и использовать обычный виджет CheckBox, однако кнопка ToggleButton выглядит более привлекательно и больше подходит для включения/выключения режимов программы, чем переключатель CheckBox.

У кнопки есть два основных свойства: android:textOff и android:textOn. Первое устанавливает текст кнопки, когда она выключена, а второе — когда включена. В программном коде этим свойствам соответствуют методы setTextOff() и setTextOn().

С помощью метода setChecked(boolean checked) вы можете программно изменить состояние кнопки. При изменении состояния генерируется событие onCheckedChanged().

Добавьте в наш проект кнопку ToggleButton и удалите все, что нам не нужно. Пусть в проекте останутся только текстовое поле TextView, кнопка Button (она нам еще пригодится) и новая кнопка ToggleButton. Разметка проекта представлена в листинге 4.13.

Листинг 4.13. Использование кнопки ToggleButton (файл разметки)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
       android:orientation="vertical"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
    >
```

```
<TextView
        android:id="@+id/txt1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="20pt"
        android:textStyle="bold"
        android:text="Выкл"
    />
<ToggleButton
                android:textOff="Off"
                android:textOn="Вкл"
                android:id="@+id/tb1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
</ToggleButton>
</LinearLayout>
```

При нажатии на кнопку `ToggleButton` в зависимости от состояния кнопки будет изменяться значение текстового поля. Этот эффект достигается благодаря обработчику события `OnCheckedChange`. Код Java представлен в листинге 4.14.

Листинг 4.14. Пример использования кнопки `ToggleButton`

```
package com.example.ch04;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;
import android.widget.CompoundButton;
import android.widget.CompoundButton.OnCheckedChangeListener;
import android.widget.ToggleButton;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        final TextView txt1 = (TextView)findViewById(R.id.txt1);
        final ToggleButton toggle = (ToggleButton)findViewById(R.id.tb1);

        toggle.setOnCheckedChangeListener(new OnCheckedChangeListener() {

            public void onCheckedChanged(
                    CompoundButton buttonView, boolean isChecked) {
```

```
                if (isChecked)
                    txt1.setText("Вкл");
                else
                    txt1.setText("Выкл");
            }
        });
    }
}
```

Результат внесенных в код изменений показан на рис. 4.21.

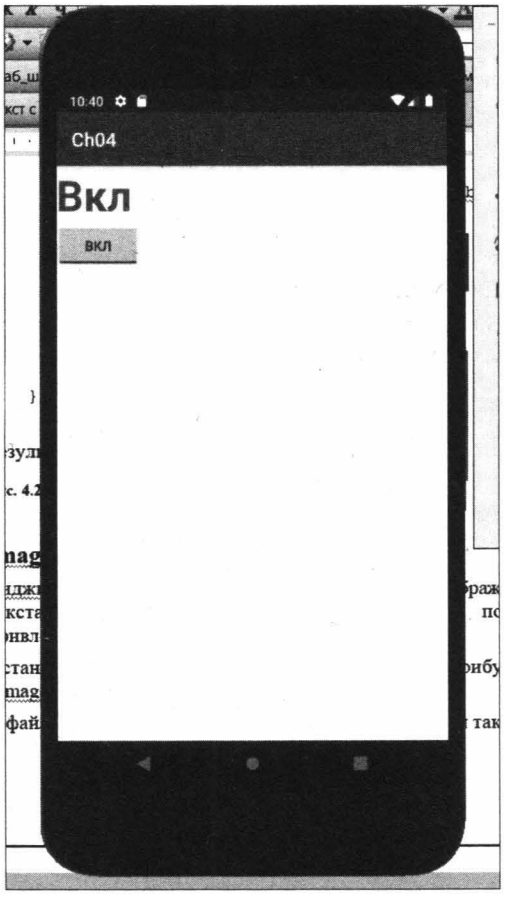


Рис. 4.21. Так выглядит кнопка `ToggleButton` в работе

ImageButton — кнопка с изображением

Виджет `ImageButton` — что-то среднее между изображением и кнопкой. Вместо текста у этой кнопки будет изображение, что позволяет создавать более привлекательные кнопки.

Установить изображение кнопки можно или атрибутом `android:src` элемента `<ImageButton>` или методом `setImageResource(int)`.

В файле разметки элемент <ImageButton> описывается так:

```
<ImageButton
android:id="@+id/button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/icon"/>
```

Работа с такой кнопкой осуществляется так же, как и с обычной кнопкой. Первым делом нужно подключить необходимый пакет, объявить переменную и найти сам элемент в файле разметки:

```
import android.widget.ImageButton;
...
ImageButton button;
...
button = (ImageButton)findViewById(R.id.button);
button.setImageResource(R.drawable.im);
```

Последние два оператора находят кнопку ImageButton и устанавливают для нее изображение с именем play, которое находится в папке res\drawable или res\drawable-* (имя зависит от разрешения экрана — для более новых платформ Android). Лучше всего использовать файлы в формате PNG. В нашем случае (ресурс R.drawable.im) в каталоге res\drawable должен присутствовать файл im.png.

Обработчик нажатия кнопки устанавливается так же, как и для обычной кнопки Button.

4.2.3. Индикатор *ProgressBar*

Сейчас мы рассмотрим весьма интересный виджет — ProgressBar, представляющий собой индикатор (шкалу) процесса.

Начнем с разметки (листинг 4.15). Наша деятельность будет содержать два поля ввода, кнопку и, конечно же, ProgressBar. Редактор разметки для нашего приложения показан на рис. 4.22.

Листинг 4.15. Разметка ProgressBar

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

<TextView android:id="@+id/TextView02"
       android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Максимум"></TextView>
```

```
<EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content"
android:text="100"
android:id="@+id/maximum"></EditText>

<TextView android:id="@+id/TextView01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Шаг"></TextView>

<EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content"
android:text="5"
android:id="@+id/increment"></EditText>

<Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
android:text="Пуск"
android:id="@+id/startbtn"></Button>

</LinearLayout>
```

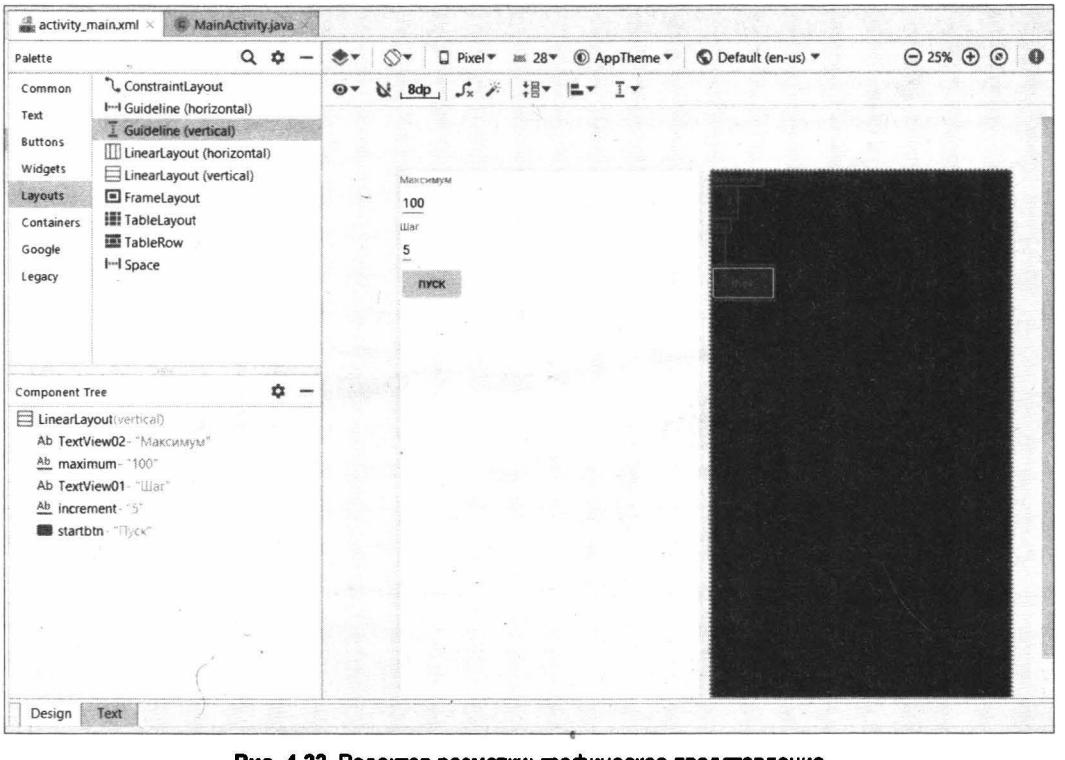


Рис. 4.22. Редактор разметки: графическое представление

Прежде чем приступать к написанию Java-кода, нужно разобраться, как будет работать наше приложение. Максимальное значение для `ProgressBar` устанавливается в поле `Max Value`, шаг (значение, на которое будет увеличено значение `ProgressBar` каждые полсекунды или 500 мс) задается значением, указанным в поле `Increment by`. При нажатии на кнопку `Start` появится диалоговое окно, в котором и будет отображен наш индикатор `ProgressBar`. Увеличение значения `ProgressBar` будет производиться в потоке каждые 500 мс. Понимаю, что с диалоговыми окнами мы еще пока не работали, но они будут рассмотрены уже в следующей главе.

Полный исходный Java-код представлен в листинге 4.16. Внимательно читайте комментарии, чтобы разобраться что и к чему. На этот раз я не выделял никакие строки, поскольку выделять пришлось бы бóльшую часть Java-кода. Также удалены обработчики меню, которые были в исходном проекте, — они только занимали место и не делали ничего полезного.

Листинг 4.16. Полный Java-код приложения (ProgressBar)

```
package com.example.ch04;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.widget.Button;
import android.widget.EditText;
import android.app.ProgressDialog;
import android.view.View;
import android.view.View.OnClickListener;

public class MainActivity extends AppCompatActivity {

    int increment;
    ProgressDialog dialog;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button startbtn = (Button) findViewById(R.id.startbtn);
        startbtn.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {

                // получаем шаг инкремента из текстового поля
                EditText et = (EditText) findViewById(R.id.increment);
                // конвертируем строку в число
                increment = Integer.parseInt(et.getText().toString());

                // создаем новый диалог
                dialog = new ProgressDialog(MainActivity.this);
```

```
                dialog.setCancelable(true);
                dialog.setMessage("Загрузка...");
                // шкала должна быть горизонтальной
                dialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
                // значение шкалы по умолчанию — 0
                dialog.setProgress(0);

                // получаем максимальное значение
                EditText max = (EditText) findViewById(R.id.maximum);
                // конвертируем строку в число
                int maximum = Integer.parseInt(max.getText().toString());
                // устанавливаем максимальное значение
                dialog.setMax(maximum);
                // отображаем диалог
                dialog.show();

                // создаем поток для обновления шкалы
                Thread background = new Thread(new Runnable() {
                    public void run() {
                        try {
                            // увеличиваем значение шкалы каждые 500 мс,
                            // пока не будет достигнуто максимальное значение
                            while (dialog.getProgress() <= dialog.getMax()) {
                                // ждем 500 мс
                                Thread.sleep(500);

                                // активируем обработчик обновления
                                progressHandler.sendMessage(progressHandler.
                                                            obtainMessage());
                            }
                        } catch (java.lang.InterruptedException e) {

                        }
                    }
                });
                // запускаем фоновый поток
                background.start();

            }
            // обработчик для фонового обновления
            Handler progressHandler = new Handler() {
                public void handleMessage(Message msg) {
                    // увеличиваем значение шкалы
                    dialog.incrementProgressBy(increment);
                }
            };
        });
    }
}
```

Запустите приложение, установите параметры (рис. 4.23) и нажмите кнопку **ПУСК** — вы увидите диалоговое окно и работающий в нем индикатор `ProgressBar` (рис. 4.24).

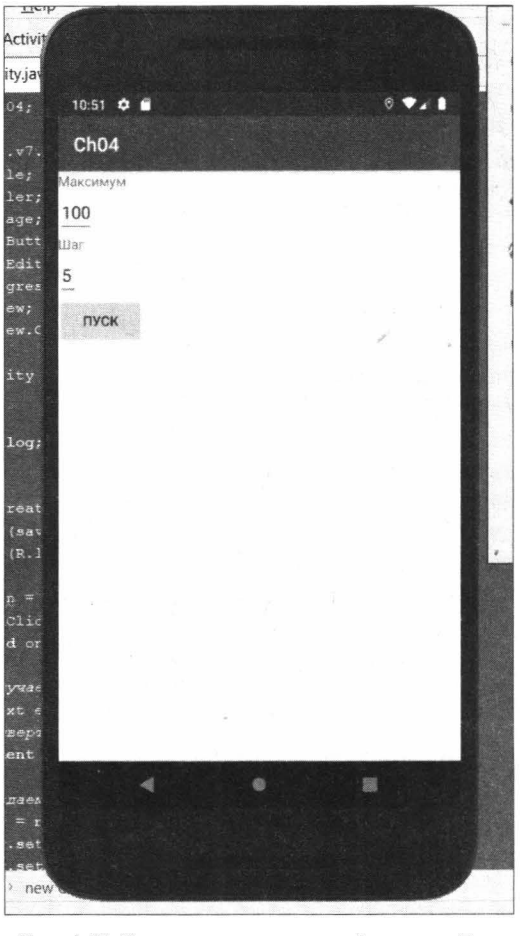


Рис. 4.23. Установка параметров `ProgressBar`

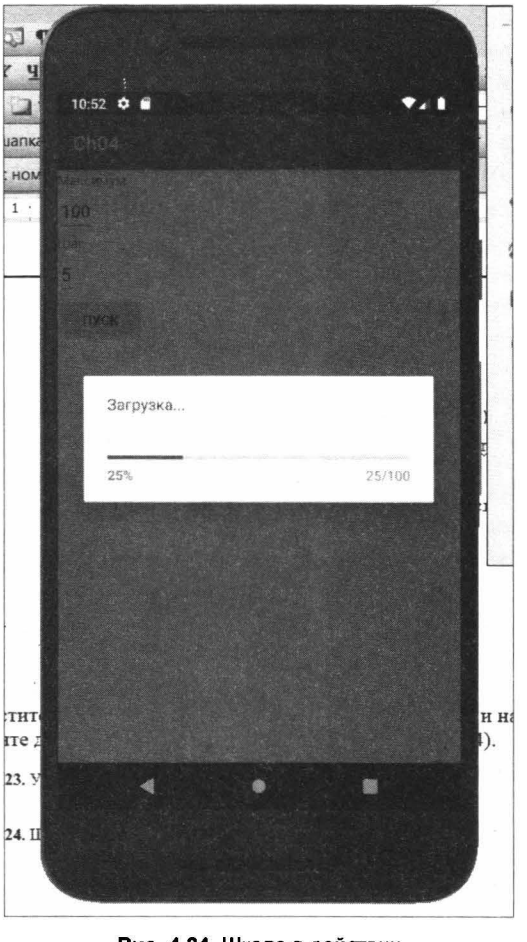


Рис. 4.24. Шкала в действии

4.2.4. Средства отображения графики

Для отображения графики служит виджет `ImageView`, являющийся базовым элементом для графического наполнения.

Для загрузки изображения в классе `ImageView` существует несколько методов:

- `setImageResource(int resId)` — загружает изображение из ресурса;
- `setImageURI(Uri uri)` — загружает изображение по его URI;
- `setImageBitmap(Bitmap bitmap)` — загружает растровое изображение.

Для изменения размера изображения используются следующие методы:

- `setMaxHeight()` — устанавливает максимальную высоту;
- `setMaxWidth()` — устанавливает максимальную ширину.

Если вы хотите загрузить изображение из файла разметки, то используйте атрибут `android:src`. При добавлении `ImageView` с помощью редактора разметки откроется окно, позволяющее выбрать изображение из списка ресурсов проекта или системных ресурсов.

В файле разметки виджет `ImageView` определяется так:

```
<ImageView android:layout_height="wrap_content" android:id="@+id/imageView1"
android:src="@drawable/icon"
android:layout_width="wrap_content">
</ImageView>
```

После этого в Java-коде работа с виджетом производится так:

```
final ImageView image = (ImageView)findViewById(R.id.imageView1);
// загружаем изображение из ресурса
image.setImageResource(R.drawable.icon);
```

* * *

В этой главе мы познакомились с основными виджетами. А об остальных виджетах вы можете прочитать в руководстве разработчика Android: **http://developer.android.com/reference/android/package-summary.html**.

В следующей же главе мы поговорим об уведомлениях, диалоговых окнах и меню.

ГЛАВА 5

Уведомления, диалоговые окна и меню

5.1. Уведомления

5.1.1. Простое всплывающее уведомление

Приложения могут отображать два типа уведомлений: краткие всплывающие сообщения (Toast Notification) и постоянные напоминания (Status Bar Notification). Первые отображаются на экране мобильного устройства какое-то время и не требуют внимания пользователя. Как правило, это не критические информационные сообщения. Вторые постоянно отображаются в строке состояния и требуют реакции пользователя.

Например, приложение требует подключения к вашему серверу. Если соединение успешно установлено, можно отобразить краткое уведомление, а вот если подключиться не получилось, тогда отображается постоянное уведомление, чтобы пользователь сразу мог понять, почему приложение не работает.

Чтобы отобразить всплывающее сообщение, используйте класс `Toast` и его методы `makeText` (создает текст уведомления) и `show` (отображает уведомление):

```
import android.widget.Toast;
import android.content.Context;
...
Context context = getApplicationContext();
Toast toast = Toast.makeText(context, "Это уведомление",
                             Toast.LENGTH_LONG);
toast.show();
```

Первый параметр метода `makeText()` — это контекст приложения, который можно получить с помощью вызова `getApplicationContext()`. Второй параметр — текст уведомления. Третий — задает продолжительность отображения уведомления:

- `LENGTH_SHORT` — небольшая продолжительность (1–2 секунды) отображения текстового уведомления;
- `LENGTH_LONG` — показывает уведомление в течение более длительного периода времени (примерно 4 секунды). Предпочтительнее использовать «длинные» уве-

домления, поскольку пользователь может просто не успеть прочитать текст, выводимый программой.

По умолчанию всплывающее уведомление появится в нижней части экрана. Чтобы отобразить уведомление в другом месте, можно воспользоваться методом `setGravity()`, который нужно вызвать до метода `show()`:

```
toast.setGravity(Gravity.CENTER, 0, 0);
```

Первый параметр задает размещение в пределах большего контейнера — например, `GRAVITY.CENTER`, `GRAVITY.TOP` и т. д. Второй параметр — это смещение по оси *X*, третий — смещение по оси *Y*.

В нашем примере уведомление будет отображено по центру окна.

Теперь немного практики. Создайте новый проект (пусть он называется `Test6` — для совместимости с моим кодом), разметку можете не изменять, а можете вообще удалить все элементы деятельности.

Java-код приложения представлен в листинге 5.1. Это приложение отобразит при запуске уведомление (рис. 5.1).

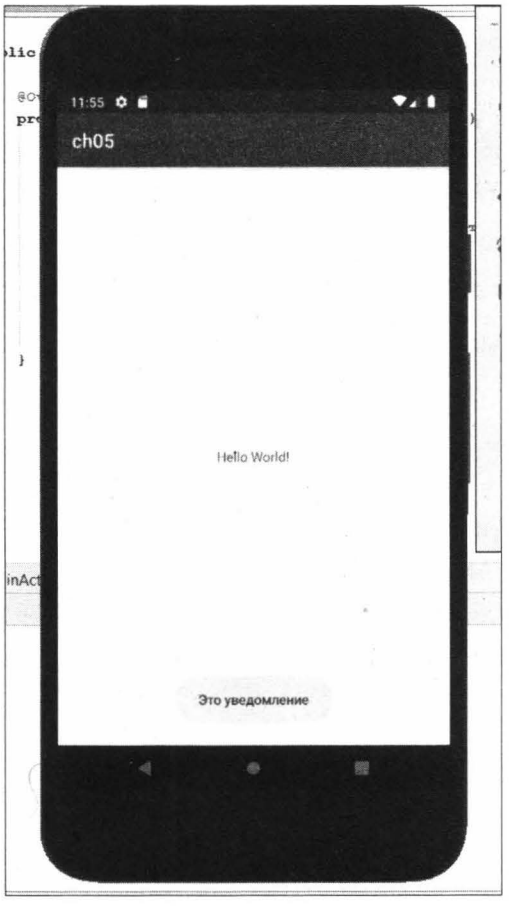


Рис. 5.1. Всплывающее уведомление

Листинг 5.1. Отображение всплывающего уведомления

```
package com.example.ch05;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
import android.content.Context;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Context context = getApplicationContext();
        Toast toast = Toast.makeText(context, "Это уведомление",
                Toast.LENGTH_LONG);
        toast.show();

    }
}
```

5.1.2. Уведомление в строке состояния

Создать уведомление в строке состояния немного сложнее. Начиная с API 16, вместо устаревшего метода `getNotification()` следует использовать метод `build()`. Интересно, что если взглянуть на исходный код Android, то видно, что старый метод вызывает метод `build()`. Видимо, разработчикам не понравилось имя метода, вот его и объявили устаревшим.

Затем нужно сформировать уведомление с помощью специального менеджера. Ссылку на `NotificationManager` можно получить через вызов метода `getSystemService()`, передав ему в качестве параметра строковую константу `NOTIFICATION_SERVICE`, определенную в классе `Context`.

Выводится уведомление с помощью метода `notify()` — это своеобразный аналог метода `show()` у параметра `Toast` из предыдущего примера (см. листинг 5.1). Взглянем на код с комментариями:

```
Notification.Builder builder = new Notification.Builder(context);

builder.setContentIntent(contentIntent)
                // маленькое изображение
        .setSmallIcon(R.drawable.small)
                // большое изображение
        .setLargeIcon(BitmapFactory.decodeResource(res, R.drawable.big))
```

```
        // Только для Android 4.x, в 5.x и 6.x не сработает
        .setTicker("Файл был зашифрован!")
        .setWhen(System.currentTimeMillis())
        .setAutoCancel(true)
       // Заголовок уведомления
        .setContentTitle("My program")
       // Текст уведомления для Android 5.x, 6.x и более новых
        .setContentText("Файл был зашифрован!");

Notification notification = builder.build();

NotificationManager notificationManager = (NotificationManager) context
        .getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFY_ID, notification);
```

Как можно видеть, для Android 4.*x* надо использовать метод `setTicker()`, который не сработает в Android 5–7, где вместо него нужно будет задействовать метод `setContentText()`. На рис. 5.2 приведено уведомление в Android 7.0 (API 24).

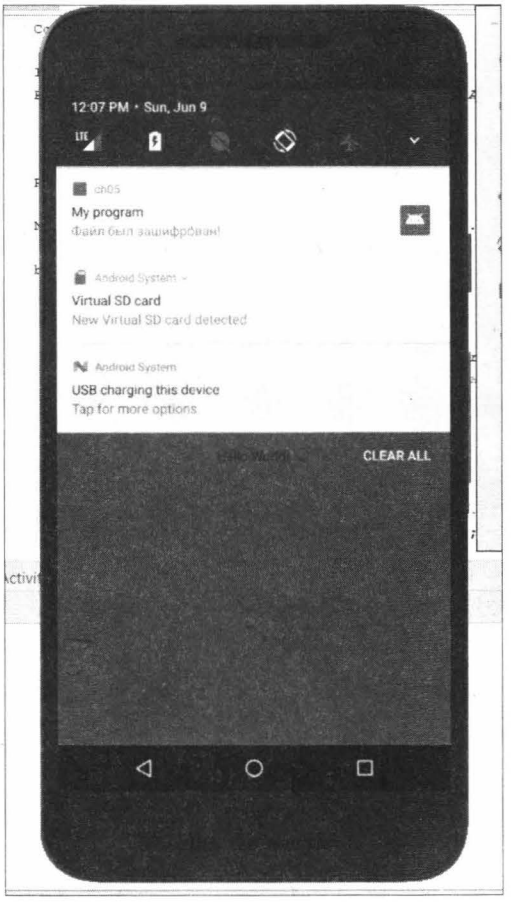


Рис. 5.2. Наше уведомление в эмуляторе

Полный код приложения, отображающего уведомление в строке состояния, приведен в листинге 5.2. Обратите внимание на подключаемые пакеты и переменные, определяемые до основного кода отображения уведомления. Также заметьте, что в каталог res\drawable нужно поместить два файла: small.png и big.png — это значки вашего приложения. Впрочем, вы можете просто скопировать их из папок res\mipmap-mdpi (маленькое) и res\mipmap-xxhdpi (большое).

При запуске приложения в строке состояния вы увидите значок уведомления и его текст.

Листинг 5.2. Уведомление в строке состояния

```
package com.example.ch05;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Toast;
import android.content.Context;

import android.app.Notification;
import android.app.NotificationManager;
import android.content.Intent;
import android.app.PendingIntent;
import android.graphics.BitmapFactory;
import android.content.res.Resources;

public class MainActivity extends AppCompatActivity {
    // ID уведомления
    private static final int NOTIFY_ID = 101;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Context context = getApplicationContext();

        Intent notificationIntent = new Intent(context, MainActivity.class);
        PendingIntent contentIntent = PendingIntent.getActivity(context,
                0, notificationIntent,
                PendingIntent.FLAG_CANCEL_CURRENT);

        Resources res = context.getResources();

        Notification.Builder builder = new Notification.Builder(context);
```

```
        builder.setContentIntent(contentIntent)
                // маленькое изображение
                .setSmallIcon(R.drawable.small)
                // большое изображение
                .setLargeIcon(BitmapFactory.decodeResource(res, R.drawable.big))
                // Только для Android 4.x, в 5.x и 6.x не сработает
                .setTicker("Файл был зашифрован!")
                .setWhen(System.currentTimeMillis())
                .setAutoCancel(true)
                // Заголовок уведомления
                .setContentTitle("My program")
                // Текст уведомления для Android 5.x, 6.x
                .setContentText("Файл был зашифрован!");

        Notification notification = builder.build();

        NotificationManager notificationManager = (NotificationManager) context
                .getSystemService(Context.NOTIFICATION_SERVICE);
        notificationManager.notify(NOTIFY_ID, notification);

    }

// Автоматически сгенерированный средой код удален

}
```

* * *

Теперь подробно рассмотрим изменения, которые произошли в новых API. При подготовке книг я всегда исхожу из того, что приводимый в них код должен работать на большом количестве устройств самых разных версий. Именно поэтому код этот не всегда самый современный. Однако, если использовать самый современный API, код перестанет работать в старых версиях Android, что не всегда хорошо.

Итак, уведомления для относительно новых версий Android (до API 26) лучше создавать с помощью объекта `NotificationCompat.Builder`:

```
NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this,
                                                                CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle(textTitle)
        .setContentText(textContent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

Конструктору вы должны передать маленький значок, получить который можно методом `setSmallIcon()`, заголовок, устанавливаемый методом `setContentTitle()`, текст уведомления, задаваемый методом `setContentText()` и приоритет, устанавливаемый методом `setPriority()`. Приоритет определяет, насколько «навязчивым» должно быть уведомление в Android 7.1 (API 25) и ниже. В Android 8.0 (начиная

с API 26) и выше вы должны использовать каналы уведомлений, которые будут рассмотрены далее.

> ***Примечание***
> Напомню уровни API: 24 — 7.0, 25 — 7.1, 26 — 8.0, 27 — 8.1, 28 — 9.0

5.1.3. Каналы уведомлений в Android 9.0

В Android 8.0 (API 26+) вы должны не просто отображать уведомления — вам следует зарегистрировать *канал уведомлений* вашего приложения, передав экземпляр объекта `NotificationChannel` методу `createNotificationChannel`. Однако этот код будет работать только на API 26 и выше.

```
private void createNotificationChannel() {
    // Создаем NotificationChannel, работает на API 26+
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        CharSequence name = getString(R.string.channel_name);
        String description = getString(R.string.channel_description);
        int importance = NotificationManager.IMPORTANCE_DEFAULT;
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, name,
                                                               importance);
        channel.setDescription(description);
        // Регистрируем канал в системе, после этого нельзя
       // изменять настройки канала (важность - importance,
       // описание и т. д.)
        NotificationManager notificationManager =
                                    getSystemService(NotificationManager.class);
        notificationManager.createNotificationChannel(channel);
    }
}
```

Поскольку канал вы должны зарегистрировать до того, как начнете отправлять уведомления, вам надо выполнить этот код как можно раньше после запуска вашего приложения.

5.1.4. Определение действия уведомления

Обычно, если пользователь нажимает на уведомление, то запускается ваше приложение. Это общепринятая практика. Если нужно просто вывести уведомление и не требуется, чтобы при нажатии на него приложение запускалось, используйте вызов намерения без параметров:

```
Intent intent = new Intent();
```

Рассмотрим другой пример, где при нажатии на уведомлении открывается ваш сайт:

```
Context context = getApplicationContext();
Intent notificationIntent = new Intent(Intent.ACTION_VIEW,
        Uri.parse("https://dkws.org.ua/"));
```

```
PendingIntent pendingIntent = PendingIntent.getActivity(context, 0,
        notificationIntent, PendingIntent.FLAG_CANCEL_CURRENT);

Notification.Builder builder = new Notification.Builder(context)
        .setContentTitle("Посетите сайт")
        .setContentText("https://dkws.org.ua/")
        .setTicker("Info").setWhen(System.currentTimeMillis())
        .setContentIntent(pendingIntent)
        .setDefaults(Notification.DEFAULT_SOUND).setAutoCancel(true)
        .setSmallIcon(R.drawable.ic_launcher);

NotificationManager notificationManager = (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
notificationManager.notify(NOTIFY_ID, builder.build());
```

Как можно видеть, здесь мы в `Intent()` задаем адрес сайта, который нужно посетить. Адрес сайта в уведомлении никак не влияет на действие. Поэтому в уведомлении можно вывести короткий адрес сайта — например, `dkws.org.ua`, а в `Intent()` задать полный адрес — `https://dkws.org.ua/`.

5.1.5. Кнопки действия

Иногда простого нажатия на уведомление недостаточно, и нужно добавить несколько кнопок действия. Для создания кнопки действия передайте в метод `addAction()` параметр `PendingIntent`. Код выглядит примерно так:

```
Intent snoozeIntent = new Intent(this, MyBroadcastReceiver.class);
snoozeIntent.setAction(ACTION_SNOOZE);
snoozeIntent.putExtra(EXTRA_NOTIFICATION_ID, 0);
PendingIntent snoozePendingIntent =
        PendingIntent.getBroadcast(this, 0, snoozeIntent, 0);

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this,
CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("Мое уведомление")
        .setContentText("Привет!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setContentIntent(pendingIntent)
        .addAction(R.drawable.ic_snooze, getString(R.string.snooze),
                snoozePendingIntent);
```

В метод `addAction()` вы должны передать также изображение для вашей кнопки и текст надписи на кнопке.

Дополнительную информацию (в том числе способы организации ввода из области уведомлений и построения индикатора процесса в области уведомлений) вы можете найти по адресу: **https://developer.android.com/training/notify-user/build-notification**.

5.1.6. Удаление собственных уведомлений

Если вы ошибочно отправили уведомление, тогда удалить его можно так:

```
notificationManager.cancel(NOTIFY_ID);
```

5.1.7. Звуковая, световая и вибросигнализация

При выдаче уведомления можно добавить звуковой сигнал, вибросигнал и мерцание светодиодов телефона:

- `Notification.DEFAULT_LIGHTS` — мерцание светодиодами;
- `Notification.DEFAULT_SOUND` — звуковой сигнал;
- `Notification.DEFAULT_VIBRATE` — вибрация.

Тип уведомления задается так:

```
notification.defaults = Notification.DEFAULT_SOUND |
        Notification.DEFAULT_VIBRATE;
```

Как можно видеть, здесь будут использоваться и звук, и вибрация. Параметры звука и вибрации принимаются по умолчанию. В большинстве случаев этого достаточно, однако параметры уведомления допускается «кастомизировать». Задать звук уведомления можно так:

```
notification.sound = Uri.parse("file:///sdcard/warning.mp3");
```

Но это только в том случае, если вы знаете точный путь к звуковому файлу. Вы можете также добавить звуковой файл в качестве ресурса и использовать его в своем проекте.

Для использования в приложении вибрации нужно в файл манифеста добавить строку:

```
<uses-permission android:name="android.permission.VIBRATE"/>
```

В настоящее время эмулятор Android не умеет воспроизводить звуковое и световое уведомление (про вибрацию, думаю, и так понятно). Поэтому, если приведенный код не работает в эмуляторе, — не расстраивайтесь, просто запустите созданный APK-файл на реальном устройстве.

5.1.8. Вывод длинного текста

Если вы попробуете установить длинный текст методом `setContentText()`, то заметите, что он отображен не полностью. Для отображения длинного текста нужно использовать метод `BigTextStyle().bigText()`. Вот пример вывода длинного текста:

```
public void sendBigTextStyleNotification(View view) {
        String bigText = "Lorem Ipsum"
                        + " is simply dummy text of the printing and typesetting
                          industry. Lorem Ipsum has been the industry's standard
                          dummy text ever since the 1500s";
```

```
        Intent intent = new Intent(this, SecondActivity.class);
        PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, 0);

        Notification.Builder builder = new Notification.Builder(this)
              .setTicker("Lorem Ipsum!")
              .setContentTitle("Длинное уведомление")
              .setContentText(
              " is simply dummy text of the printing and typesetting industry.")
              .setSmallIcon(R.drawable.ic_launcher)
              .addAction(R.drawable.ic_launcher, "Run activity",
                        pIntent).setAutoCancel(true);

        Notification notification = new Notification.BigTextStyle(builder)
              .bigText(bigText).build();

        NotificationManager notificationManager = (NotificationManager)
                                          getSystemService(NOTIFICATION_SERVICE);
        notificationManager.notify(1, notification);
}
```

5.2. Диалоговые окна

В *главе 4* мы познакомились с диалоговым окном `ProgressDialog`, отображающим индикатор `ProgressBar`, поэтому здесь мы этот тип диалоговых окон рассматривать не станем. Однако кроме `ProgressDialog` в Android предусмотрены диалоговые окна следующих типов:

- `AlertDialog` — диалоговое окно с кнопками;
- `DatePickerDialog` — диалоговое окно выбора даты;
- `TimePickerDialog` — диалоговое окно выбора времени.

5.2.1. Диалоговое окно *AlertDialog*

Самый частый вариант применения диалогового окна `AlertDialog` — это классическое диалоговое окно вопроса с кнопками **Yes** и **No**. Вот сейчас мы и займемся его разработкой. При запуске наше приложение отобразит диалоговое окно, при нажатии кнопки **No** в котором не будет произведено никаких действий (будет вызван метод `cancel()`). А вот действие при нажатии на кнопку **Yes** запрограммируете вы сами — но позже, когда научитесь устанавливать интернет-соединения. Взгляните, как будет выглядеть наше диалоговое окно (рис. 5.2).

Код приложения, отображающего это диалоговое окно, представлен в листинге 5.3.

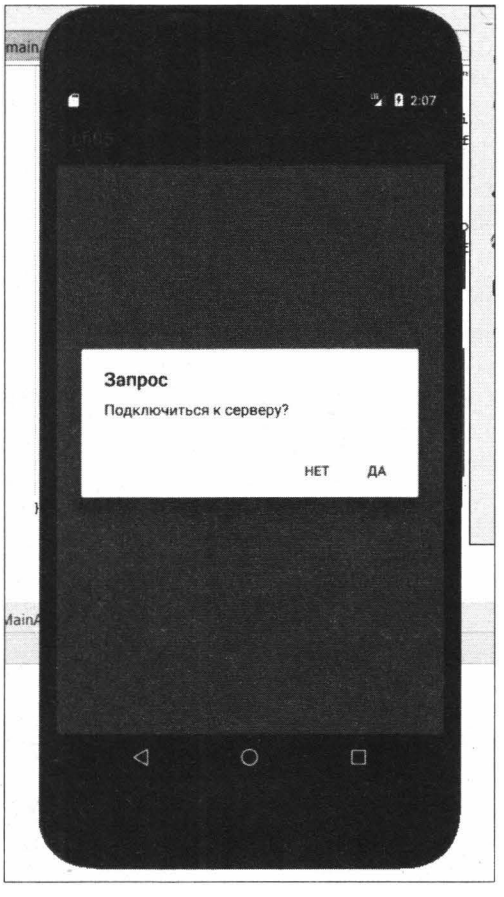


Рис. 5.3. Диалоговое окно типа `AlertDialog`

Листинг 5.3. Пример диалогового окна типа `AlertDialog`

```
package com.example.ch05;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.content.Context;
import android.app.Dialog;
import android.content.DialogInterface;
import android.app.AlertDialog;


public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        AlertDialog.Builder alt_bld = new AlertDialog.Builder(this);
        // Сообщение диалога
        alt_bld.setMessage("Подключиться к серверу?")
                .setCancelable(false)
                .setPositiveButton("Да", new DialogInterface.OnClickListener()
{
                    public void onClick(DialogInterface dialog, int id) {
                        // Действие для кнопки Yes
                    }
                })
                .setNegativeButton("Нет", new DialogInterface.OnClickListener()
{
                    public void onClick(DialogInterface dialog, int id) {
                        //  Действие для кнопки No
                        dialog.cancel();
                    }
                });
        AlertDialog alert = alt_bld.create();
        // Title for AlertDialog
        alert.setTitle("Запрос");
        // Icon for AlertDialog
        alert.show();

    }
}
```

5.2.2. *DatePickerDialog*: диалоговое окно выбора даты

В Android существуют встроенные диалоговые окна выбора даты и времени, следовательно, вам не придется «изобретать велосипед» для создания таковых. В предыдущем издании этой книги было показано, как использовать оба эти диалоговых окна, однако API Android с тех пор изменился, и теперь диалоговые окна даты и времени вызываются иначе. Так, метод `showDialog()` считается устаревшим, и для отображения диалоговых окон нужно теперь использовать метод `show()`.

Сейчас мы создадим приложение, отображающее диалоговое окно выбора даты. Разметка проекта предусматривает надпись, в которую будет выводиться дата, и кнопки **Выбор даты** и **Готово**. По нажатию кнопки **Выбор даты** открывается диалоговое окно выбора даты, по нажатию в этом окне кнопки **Готово** выбранная дата отображается в надписи.

Прежде всего определим для нашего приложения некоторые константы (листинг 5.4) и занесем их в файл strings.xml.

Листинг 5.4. Строковые константы (файл res\values\strings.xml)

```
<resources>
    <string name="app_name">Ch05</string>
```

```
    <string name="choose_date">Выбор даты</string>
    <string name="ready">Готово</string>
</resources>
```

Разметка нашего приложения приведена в листинге 5.5.

Листинг 5.5. Разметка приложения (файл activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="16/09/1983"
        android:id="@+id/tv"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Выбор даты"
        android:id="@+id/button"
        android:layout_below="@+id/tv"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="116dp"
        android:textSize="22sp"
        android:onClick="onClick"/>
</android.support.constraint.ConstraintLayout>
```

Теперь в дереве проекта (крайняя левая панель Android Studio) щелкните правой кнопкой мыши на строке java/<название пакета>, выберите команду **New | Java Class** (рис. 5.4) и введите название класса: `DatePicker`. Код класса приведен в листинге 5.6.

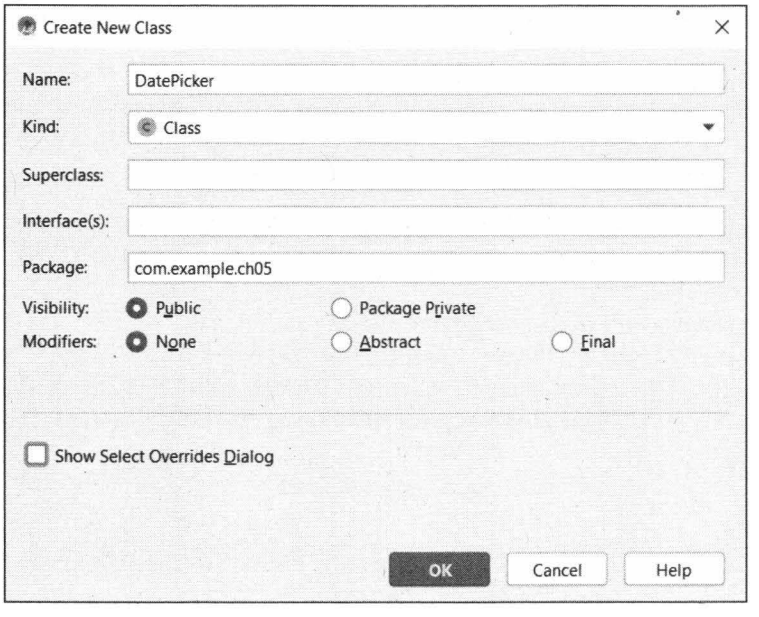


Рис. 5.4. Создание нового класса

Листинг 5.6. Класс `DatePicker` (файл DatePicker.java)

```
package com.example.ch05;

import android.app.AlertDialog;
import android.app.DatePickerDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.widget.Button;
import android.widget.TextView;

import java.util.Calendar;

public class DatePicker extends DialogFragment
        implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {

        // определяем текущую дату
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);
```

```
        // создаем DatePickerDialog и возвращаем его
        Dialog picker = new DatePickerDialog(getActivity(), this,
                year, month, day);
        picker.setTitle(getResources().getString(R.string.choose_date));

        return picker;
    }
    @Override
    public void onStart() {
        super.onStart();
        // добавляем пользовательский текст для кнопки
        Button nButton =  ((AlertDialog) getDialog())
                .getButton(DialogInterface.BUTTON_POSITIVE);
        nButton.setText(getResources().getString(R.string.ready));

    }

    @Override
    public void onDateSet(android.widget.DatePicker datePicker, int year,
                          int month, int day) {

        TextView tv = (TextView) getActivity().findViewById(R.id.tv);
        tv.setText(day + "-" + month + "-" + year);
    }
}
```

Осталось только написать код для главной деятельности (листинг 5.7). Там все просто — мы определяем обработчик для нажатия кнопки. Само диалоговое окно выбора даты вызывается методом `show()`:

```
DialogFragment dateDialog = new DatePicker();
dateDialog.show(getSupportFragmentManager(), "datePicker");
```

Листинг 5.7. Файл MainActivity.java

```
package com.example.ch05;

import android.support.v7.app.AppCompatActivity;
import android.support.v4.app.DialogFragment;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
```

```
    public void onClick(View v) {
        DialogFragment dateDialog = new DatePicker();
        dateDialog.show(getSupportFragmentManager(), "datePicker");
    }

}
```

Теперь посмотрим на наше приложение в действии. Сначала отображается установленное для надписи значение по умолчанию и кнопка **Выбор даты** (рис. 5.5).

По нажатию кнопки **Выбор даты** отображается наше диалоговое окно (рис. 5.6).

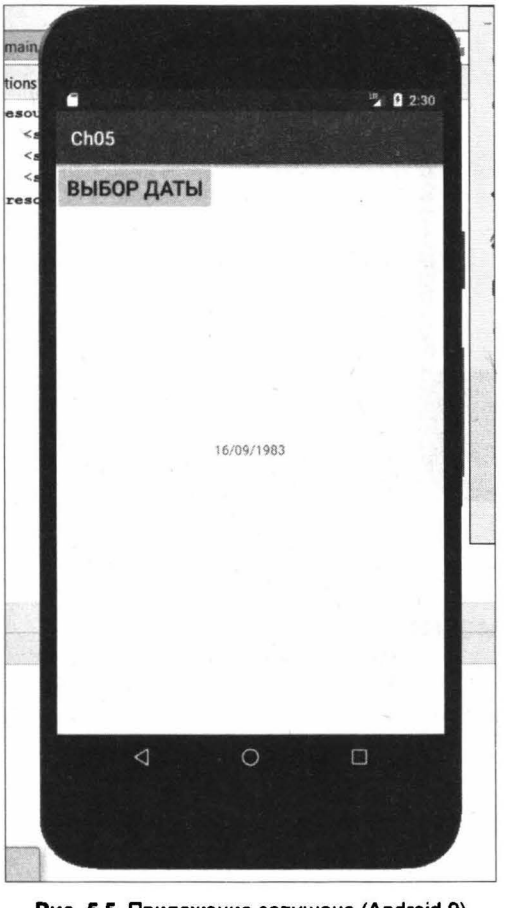


Рис. 5.5. Приложение запущено (Android 9)

Рис. 5.6. Диалоговое окно выбора даты

По нажатию в диалоговом окне кнопки **Готово** выбранная дата устанавливается в качестве значения для надписи (рис. 5.7).

Но с нашим диалоговым окном есть одна проблема. Посмотрите на рис. 5.6 — выбрана дата 9 июня 2019 года, а на рис. 5.7 отображается дата 9-5-2019. Почему-то нумерация месяцев начинается здесь с нуля, поэтому этот факт нужно учитывать и при установке даты увеличивать переменную `month` на единицу.

Рис. 5.7. Дата выбрана

5.2.3. *TimePickerDialog*: диалоговое окно выбора времени

Диалоговое окно `TimePickerDialog` работает аналогично диалоговому окну выбора даты. Прежде всего, как обычно, определим файл со строковыми константами: res\values\strings.xml (листинг 5.8).

Листинг 5.8. Строковые константы (файл res\values\strings.xml)

```
<resources>
    <string name="app_name">TimePicker</string>
    <string name="choose_time">Установите время</string>
    <string name="ready">Готово</string>
    <string name="hours">Часы </string>
    <string name="minutes">Минуты </string>
</resources>
```

Затем создаем класс `TimePickFragment` подобно тому, как мы ранее создавали класс `DatePicker` (листинг 5.9).

Листинг 5.9. Класс `TimePickFragment` (файл TimePickFragment.java)

```
package com.example.ch05time;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.TimePickerDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.widget.Button;
import android.widget.TextView;
import android.widget.TimePicker;

import java.util.Calendar;

public class TimePickFragment extends DialogFragment
        implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // устанавливаем текущее время для TimePicker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // создаем TimePickerDialog и возвращаем его
        Dialog picker = new TimePickerDialog(getActivity(), this, hour, minute,
                                                                    true);
        picker.setTitle(getResources().getString(R.string.choose_time));

        return picker;
    }

    @Override
    public void onStart() {
        super.onStart();
        // добавляем пользовательский текст для кнопки
        Button nButton =  ((AlertDialog)
                    getDialog()).getButton(DialogInterface.BUTTON_POSITIVE);
        nButton.setText(getResources().getString(R.string.ready));

    }
    @Override
    public void onTimeSet(TimePicker view, int hours, int minute) {
        // выводим выбранное время
        TextView tv = (TextView) getActivity().findViewById(R.id.tv);
```

```
        tv.setText(getResources().getString(R.string.hours) + hours
                + getResources().getString(R.string.minutes) + minute);
    }

}
```

Разметка (файл activity_main.xml) будет почти такой же (листинг 5.10).

Листинг 5.10. Разметка приложения (файл activity_main.xml)

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>

    <TextView
        android:text="@string/app_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="22sp"
        android:id="@+id/tv" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Установите время"
        android:id="@+id/button"
        android:layout_below="@+id/tv"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="116dp"
        android:textSize="22sp"
        android:onClick="onClick"/>

</RelativeLayout>
```

Осталось только определить код MainActivity.java (листинг 5.11).

Листинг 5.11. Файл MainActivity.java

```
package com.example.ch05time;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v4.app.DialogFragment;
import android.view.View;
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View v) {
        DialogFragment newFragment = new TimePickFragment();
        newFragment.show(getSupportFragmentManager(), "timePicker");
    }
}
```

В отличие от диалогового окна выбора даты, выбранное время не требует никаких поправок, и надпись отобразит именно то, что вы выбрали (рис. 5.8).

Рис. 5.8. Диалоговое окно выбора времени

5.3. Меню

Уведомления и диалоговые окна — это хорошо, но редко какое приложение обходится без меню.

Меню являются стандартными компонентами пользовательского интерфейса в приложениях многих типов. Для обеспечения привычной и единообразной технологии работы с приложением следует представлять действия пользователя и другие варианты выбора в своих операциях с помощью API-интерфейсов класса `Menu`.

В устройствах, работающих под управлением Android, начиная с версии Android 3.0 (уровень API 11), наличие отдельной кнопки **Меню** больше не требуется. С учетом этого изменения приложения для Android более не должны зависеть от традиционной панели меню из 6 пунктов. Вместо нее в них должна присутствовать строка действий с часто используемыми действиями пользователя.

Операционная система Android предлагает три вида меню:

- меню параметров (Options Menu) — это основное меню приложения, здесь следует размещать пункты, затрагивающее приложение в целом: **Настройка**, **Создать**, **Поиск** и пр.
- контекстное меню (Context Menu) — это меню появляется при долгом касании (2 или более секунды) сенсорного экрана;
- всплывающее меню или подменю (Submenu) — оно привязывается к конкретному пункту меню (меню параметров или контекстного меню). Пункты подменю не поддерживают вложенное меню.

5.3.1. Определение меню в XML-файле

С учетом того, что меню более не ограничено шестью пунктами и может содержать гораздо больше команд, проще определять команды меню в XML-файле, который находится в папке \res\menu\. Использовать XML-файлы (а не определять команды меню в Java-коде) рекомендуется по нескольким причинам:

- в XML проще визуализировать структуру меню;
- это позволяет отделить контент для меню от кода, определяющего работу приложения;
- это позволяет создавать альтернативные варианты меню для разных версий платформы, размеров экрана и других конфигураций путем использования структуры ресурсов приложения.

Вернемся к XML-файлу. Он должен содержать следующие элементы:

- `<menu>` — определяет класс `Menu`, являющийся контейнером для пунктов меню. Элемент `<menu>` должен быть корневым узлом файла, в котором может находиться один или несколько элементов `<item>` и `<group>`;
- `<item>` — представляет один пункт меню. Элемент может содержать вложенный элемент `<menu>` для создания подменю;

- `<group>` — невидимый контейнер для элементов `<item>`. Позволяет разделять пункты меню на категории и назначать им одинаковые свойства — например, видимость, активное состояние.

В листинге 5.12 приведен пример файла main_menu.xml, описывающего меню приложения.

Листинг 5.12. Меню приложения (файл main_menu.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

Элемент `<item>` поддерживает несколько атрибутов, с помощью которых можно определить внешний вид и поведение пункта меню. Пункты приведенного в листинге 5.12 меню имеют следующие атрибуты:

- `android:id` — идентификатор ресурса, который является уникальным для этого пункта, что позволяет приложению распознавать пункт, когда его выбирает пользователь;
- `android:icon` — ссылка на графический элемент, который будет использоваться в качестве значка пункта меню;
- `android:title` — ссылка на строку, которая будет использоваться в качестве названия пункта меню;
- `android:showAsAction` — указывает, когда и как этот пункт должен отображаться в строке действий.

Это самые важные атрибуты, но при желании в Android SDK вы можете ознакомиться и с другими атрибутами.

К пункту любого меню (кроме вложенного) можно добавить вложенное меню или подменю. Для этого нужно добавить элемент `<menu>` в качестве дочернего элемента `<item>` (листинг 5.13).

Листинг 5.13. Определение подменю

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
```

```
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

5.3.2. Создание основного меню (меню параметров)

Как уже отмечалось, в основном меню размещаются команды, затрагивающие приложение в целом. Место, где будет отображено это меню, зависит только от версии Android, и сами вы не можете повлиять на его расположение.

Объявлять пункты меню параметров можно либо из подкласса `Activity`, либо из подкласса `Fragment`. Если и ваша операция, и фрагменты объявляют пункты меню параметров, в пользовательском интерфейсе они объединяются. При этом сначала отображаются пункты операции, а за ними следуют пункты фрагментов в том порядке, в котором каждый фрагмент добавляется в операцию. При необходимости можно изменить порядок следования пунктов меню с помощью атрибута `android:orderInCategory`, указываемого в каждом элементе `<item>`, который требуется переместить.

Чтобы указать меню параметров для операции, следует переопределить метод `onCreateOptionsMenu()`. В этом методе можно загрузить в класс `Menu` собственный ресурс меню, заблаговременно определенный в XML:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_menu, menu);
    return true;
}
```

Пункты меню также можно добавлять с помощью метода `add()`, а получать их с помощью метода `findItem()` для пересмотра свойств этих пунктов с помощью API-интерфейсов `MenuItem`. Именно метод `add()` использовался в предыдущем издании этой книги:

```
public static final int IDM_NEW = 101;
       public static final int IDM_OPEN = 102;
       public static final int IDM_SAVE = 103;
       public static final int IDM_EXIT = 104;
...
  @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Создаем пункты меню
        menu.add(Menu.NONE, IDM_NEW, Menu.NONE, "New");
```

```
        menu.add(Menu.NONE, IDM_OPEN, Menu.NONE, "Open");
        menu.add(Menu.NONE, IDM_SAVE, Menu.NONE, "Save");
        menu.add(Menu.NONE, IDM_EXIT, Menu.NONE, "Exit");

        return(super.onCreateOptionsMenu(menu));
    }
```

Если пунктов меню немного, вы можете использовать метод `add()`. А во всех остальных случаях удобнее использовать XML-файл.

Теперь поговорим об обработке нажатий. Наши пункты меню уже отображаются, но пока ничего не делают, поскольку обработчики событий еще не заданы. Когда пользователь выбирает пункт меню параметров, система вызывает метод `onOptionsItemSelected()`. Этот метод передает выбранный класс `MenuItem`. Идентифицировать пункт меню можно, вызвав метод `getItemId()`, который возвращает уникальный идентификатор пункта меню (определенный атрибутом `android:id` из ресурса меню или целым числом, переданным методу `add()`). Этот идентификатор можно сопоставить с известными пунктами меню, чтобы выполнить соответствующее действие. Рассмотрим пример:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Обрабатываем выбранное действие
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Если вы использовали метод `add()`, то распознать, какая команда была выбрана пользователем, можно так:

```
        switch (item.getItemId()) {
            case IDM_NEW: t = "New selected"; break;
            case IDM_OPEN: t = "Open selected"; break;
            case IDM_SAVE: t = "Save selected"; break;
            case IDM_EXIT: t = "Exit selected"; break;
            default: return false;
        }
```

Это также делается в методе `onOptionsItemsSelected()`.

Начиная с Android 3.0, появилась возможность с помощью атрибута `android:onClick` определять в XML поведение для пунктов меню при нажатии на них. Значением

этого атрибута должно быть имя метода, определенное операцией с помощью меню. Этот метод должен быть общедоступным и принимать один параметр `MenuItem`, — когда система вызывает этот метод, она передает ему выбранный пункт меню.

5.3.3. Создание контекстного меню

В контекстное меню помещают команды, затрагивающие определенный элемент пользовательского интерфейса. Контекстное меню можно создать для любого представления, но чаще всего они используются для элементов из представлений `ListView`, `GridView` или других групп представлений, в которых пользователь может выполнять действия непосредственно с каждым элементом.

Существуют два способа предоставления возможности контекстных действий:

- в плавающем контекстном меню. Меню отображается в виде плавающего списка пунктов меню (вроде диалогового окна), когда пользователь длительно нажимает на экран (нажимает и удерживает нажатым) в представлении, которое объявляет поддержку контекстного меню. Пользователи могут каждый раз выполнять контекстное действие только с одним элементом;
- в режиме контекстных действий. Этот режим является системной реализацией `ActionMode`, отображающей строку контекстных действий вверху экрана с пунктами действий, которые затрагивают выбранные элементы. Когда этот режим активен, пользователи могут одновременно выполнять действие с несколькими элементами (если это допускается приложением).

Начнем с создания плавающего контекстного меню. Прежде всего нужно зарегистрировать класс `View`, с которым с помощью метода `registerForContextMenu()` надо связать контекстное меню, передав ему `View`. Если операция использует представления `ListView` или `GridView` и требуется, чтобы каждый элемент предоставлял одинаковое контекстное меню, зарегистрируйте все элементы для контекстного меню, передав `ListView` или `GridView` методу `registerForContextMenu()`.

Далее нужно реализовать метод `onCreateContextMenu()` в `Activity` или `Fragment`.

Когда зарегистрированное представление примет событие длительного нажатия, система вызовет ваш метод `onCreateContextMenu()`. Именно здесь определяются пункты меню. Делается это обычно путем загрузки ресурса меню:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

`MenuInflater` позволяет загружать контекстное меню из ресурса меню.

Остается только реализовать метод `onContextItemSelected()` для определения обработчиков меню:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

Теперь поговорим о режиме контекстных действий. Он представляет собой системную реализацию класса `ActionMode`, которая ориентирует пользователя на выполнение контекстных действий при его взаимодействии с приложением. Когда пользователь, используя этот режим, выбирает элемент, вверху экрана открывается строка контекстных действий, содержащая действия, которые пользователь может выполнить с выбранными в настоящий момент элементами. В этом режиме пользователь может выбирать несколько элементов (если это допускается приложением), снимать выделения с элементов и продолжать навигацию в операции (в тех пределах, в которых это поддерживается приложением). Режим контекстных действий отключается, а строка контекстных действий исчезает, когда пользователь снимет выделение со всех элементов, нажмет кнопку **Back** или выберет действие **Done**, расположенное слева.

Чтобы создать такое контекстное меню, нужно реализовать интерфейс `ActionMode.Callback`. В его методах обратного вызова вы можете указать действия для строки контекстных действий, реагировать на нажатия пунктов действий и обрабатывать другие события жизненного цикла для режима действий. Рассмотрим пример:

```
private ActionMode.Callback mActionModeCallback = new ActionMode.Callback() {

    // Вызывается при создании режима действий, вызывается startActionMode()
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Получаем элементы меню из XML
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Вызывается каждый раз при показе меню. Всегда вызывается после
    // onCreateActionMode, но может быть вызван несколько раз
    @Override
```

```
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // возвращаем false, если ничего не выбрано
    }

    // Вызывается, когда пользователь выбрал команду контекстного меню
    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish();
                return true;
            default:
                return false;
        }
    }

    // Вызывается при выходе из режима контекстных
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};
```

Для включения режима контекстных действий, когда это необходимо, — например, в ответ на длительное нажатие в представлении `View`**, вызывайте** `startActionMode()`:

```
someView.setOnLongClickListener(new View.OnLongClickListener() {
    // Called when the user long-clicks on someView
    public boolean onLongClick(View view) {
        if (mActionMode != null) {
            return false;
        }

        // Start the CAB using the ActionMode.Callback defined above
        mActionMode = getActivity().startActionMode(mActionModeCallback);
        view.setSelected(true);
        return true;
    }
});
```

5.3.4. Создание всплывающего меню

Всплывающее меню представляет собой модальное меню, привязанное к представлению `View`. Оно отображается ниже представления, к которому привязано, если там есть место, либо поверх него.

Если для определения меню используется XML, то алгоритм его показа следующий:

- создайте экземпляр класса `PopupMenu` с помощью его конструктора, принимающий текущие `Context` и `View` приложения, к которым должно быть привязано меню;
- с помощью `MenuInflater` загрузите свой ресурс меню в объект `Menu`, возвращенный методом `PopupMenu.getMenu()`. На API уровня 14 и выше вместо этого можно использовать `PopupMenu.inflate()`;
- вызовите метод `PopupMenu.show()`.

Рассмотрим код определения кнопки с атрибутом `android:onClick` — она будет показывать всплывающее меню:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

После этого покажем меню так:

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

В API уровня 14 и выше можно объединить две строки, которые загружают меню, с помощью `PopupMenu.inflate()`.

Меню закрывается, когда пользователь выбирает один из пунктов или касается экрана за пределами области меню. Прослушивать событие закрытия меню можно с помощью `PopupMenu.OnDismissListener`.

Для выполнения действия, когда пользователь выбирает пункт меню, необходимо реализовать интерфейс `PopupMenu.OnMenuItemClickListener` и зарегистрировать его в своем `PopupMenu`, вызвав метод `setOnMenuItemclickListener()`. Когда пользователь выбирает пункт меню, система выполняет обратный вызов `onMenuItemClick()` в вашем интерфейсе.

Пример кода:

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}
```

```
@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

* * *

На этом рассмотрение тем этой главы может считаться завершенным. А в следующей главе мы поговорим о двумерной графике.

ГЛАВА 6

Двумерная графика

В этой главе мы поговорим о создании статической графики путем рисования в объекте **View** из разметки или же рисования непосредственно на *канве* (canvas, холст). Анимация будет рассмотрена в *главе 14*.

6.1. Класс *Drawable*

Для рисования на формах и изображениях используется графическая библиотека `android.graphics.drawable`. Класс `Drawable` определяет различные виды графики — например, `BitmapDrawable`, `ShapeDrawable`, `LayerDrawable` и др.

Существуют два способа определения и инициализации объекта `Drawable`. Первый заключается в использовании объектов из каталога res\drawable, а второй — в создании XML-файла со свойствами объекта `Drawable`.

В Android-приложениях вы можете использовать изображения следующих форматов:

- PNG — рекомендуемый формат;
- JPEG — поддерживаемый формат, но лучше использовать PNG;
- BMP — поддерживается, но использовать не рекомендуется из-за большого размера файлов в этом формате;
- GIF — формат поддерживается, но его использование настоятельно не рекомендуется.

Ресурсы изображений, помещенные в каталог res\drawable, во время компиляции программы оптимизируются утилитой `aapt`. Если вам нужно использовать растровые изображения без оптимизации, поместите их в каталог res\raw — при компиляции файлы из этого каталога оптимизированы не будут.

Рассмотрим подробнее процесс добавления ресурса в проект. Предположим, что нам нужно добавить в проект два файла: foto1.jpg и foto2.jpg. Поскольку эти имена вы потом будете использовать в коде, нужно, чтобы они соответствовали правилу именования идентификаторов, т. е. начинались с символа, не содержали пробелов

и т. п. Таким образом, допустимые имена: foto1.jpg, foto2.jpg, а недопустимые: 1.jpg, 2.jpg.

Подготовьте три варианта каждого файла: с высоким разрешением, со средним и с низким. Значение разрешения зависит от выбранной платформы и от самого мобильного устройства.

Файлы с высоким разрешением нужно поместить в каталог res\drawable-hdpi, файлы с низким — в каталог res\drawable-ldpi, со средним — в каталог res\drawable-mdpi. После этого перезапустите, сверните и заново разверните элемент app в области навигатора, и вы увидите добавленные файлы.

По умолчанию ресурсы изображений загружаются в папку res\drawable. Для упрощения примера я загружу изображения в эту папку и не буду создавать их различные версии (рис. 6.1).

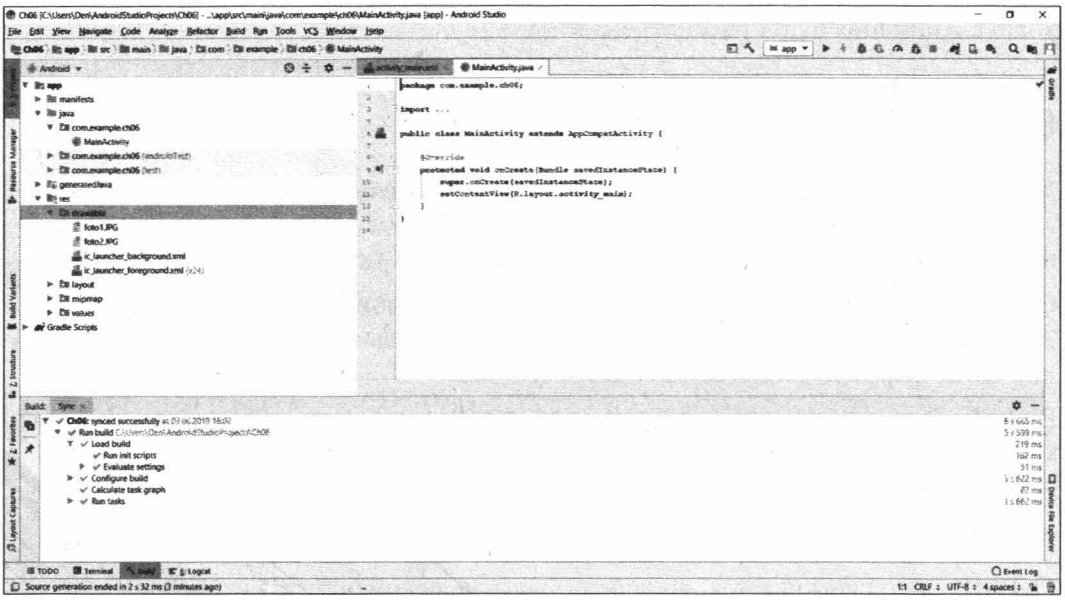


Рис. 6.1. Два изображения добавлены в проект

После этого добавьте элемент `ImageView`. При его добавлении с помощью графического редактора разметки появится окно, в котором нужно выбрать изображение, которое требуется отобразить в `ImageView` (рис. 6.2). Выбранное изображение сразу будет отображено в редакторе разметки (рис. 6.3), а в файл activity_main.xml добавлен код, приведенный в листинге 6.1.

Если же вы создавали проект не с пустой (empty) активностью, то у вас появится и второй файл — content_main.xml. В нем содержится разметка контента, поэтому если вы не хотите потерять меню и другие элементы, формируемые средой по умолчанию в файле activity_main.xml, вносите изменения в файл content_main.xml.

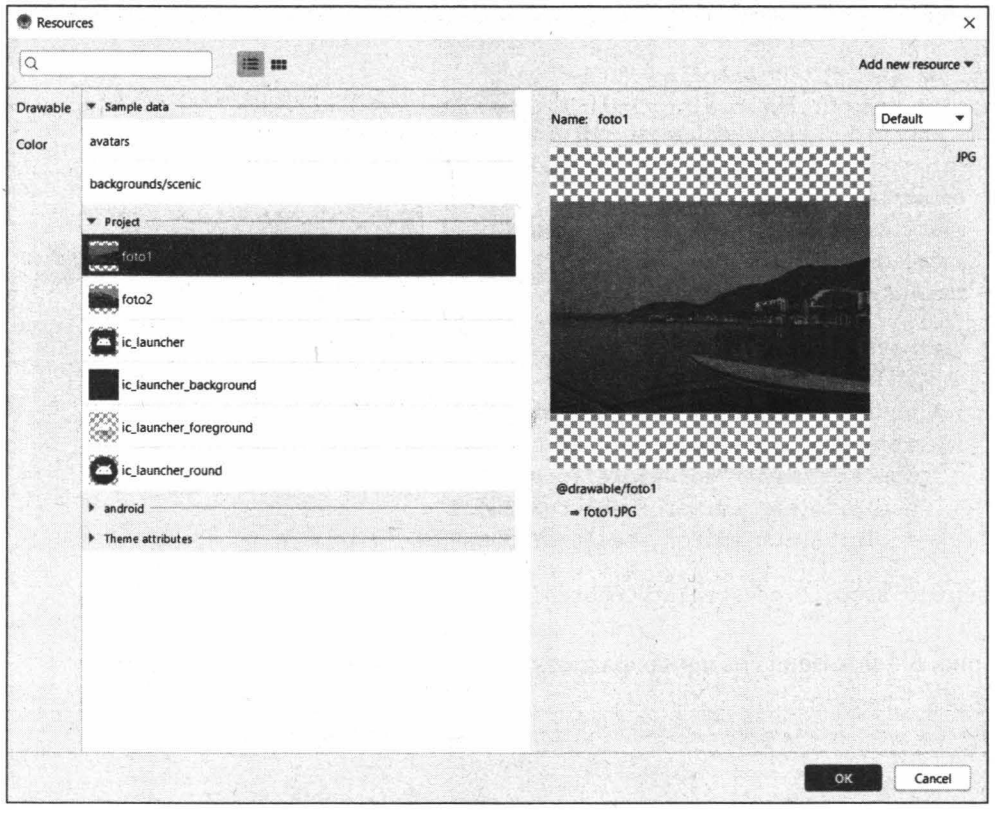


Рис. 6.2. Выбираем изображение

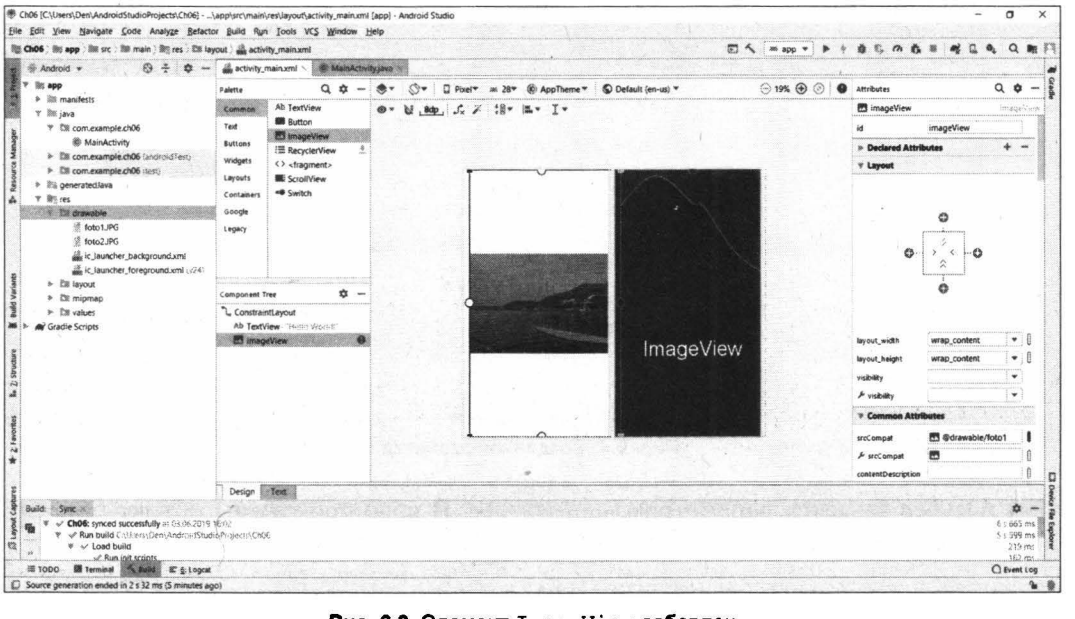


Рис. 6.3. Элемент `ImageView` добавлен

Листинг 6.1. Пример описания элемента `ImageView`

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
    tools:context=".MainActivity">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:srcCompat="@drawable/foto1"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="0dp" />

</android.support.constraint.ConstraintLayout>
```

На рис. 6.4 показана созданная разметка и ее предварительный просмотр.

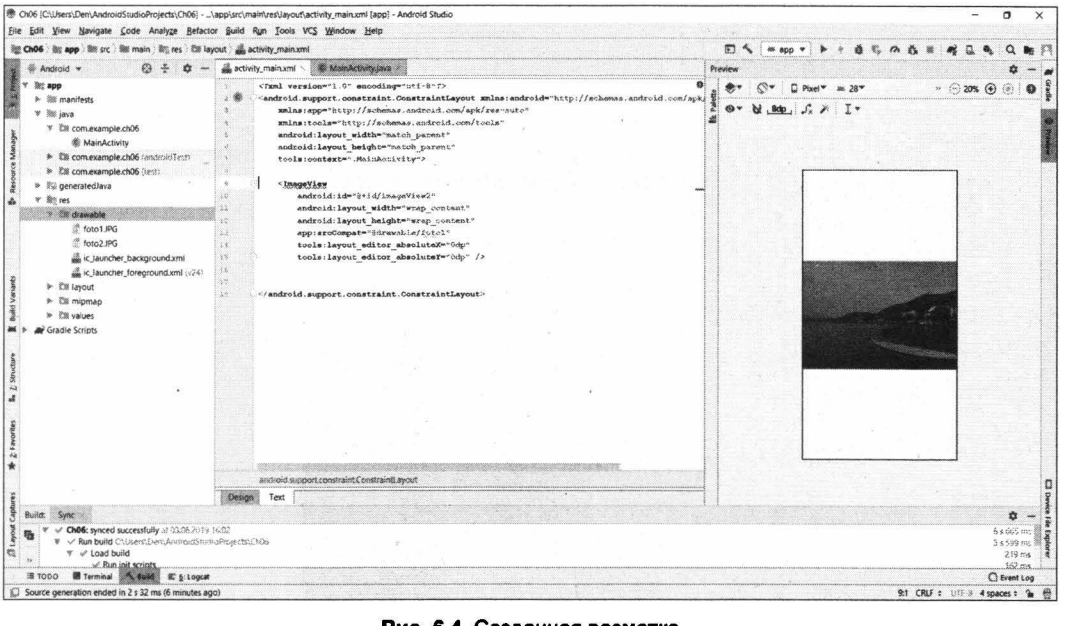


Рис. 6.4. Созданная разметка

Имя ресурса задается параметром `android:src`. В коде программы объект `Drawable` инициализируется так:

```
Resources R = mContext.getResources();
Drawable exitImage = R.getDrawable(R.drawable.exit_image);
```

Представим, что вы создали два разных объекта `Drawable` для одного и того же ресурса. Если вы потом измените свойство одного из объектов `Drawable`, это же свойство будет автоматически изменено и для второго объекта `Drawable`, поскольку они используют один и тот же ресурс.

6.2. Класс *TransitionDrawable*: переход между изображениями

В предыдущем разделе мы не зря добавили изображения в каталог ресурсов — сейчас мы рассмотрим класс `TransitionDrawable`, который используется для создания переходов между изображениями.

Код, написанный для этого случая в предыдущем издании книги, полностью устарел. Так, начиная с API 21, уже нельзя задействовать метод `getDrawable()`, который мы использовали ранее для получения файла переходов, и этот пример пришлось переделать полностью.

Для нашего приложения файл res\layout\main.xml уже готов, осталось создать только в каталоге res\drawable файл transition.xml. В этом файле определяется эффект перехода между изображениями и описываются изображения, которые будут меняться во время перехода (листинг 6.2).

Листинг 6.2. Файл @drawable/transition.xml

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android="http://schemas.android.com/apk/res/android">
<item android:drawable="@drawable/foto1"></item>
<item android:drawable="@drawable/foto2"></item>
</transition>
```

Организация самого перехода возлагается на Java-код. Но до Java-кода пока далеко, ведь нам нужно создать еще и разметку (листинг 6.3). Созданная разметка показана на рис. 6.5.

Листинг 6.3. Разметка приложения

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
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"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainActivity"
    tools:showIn="@layout/activity_main">
```

```
    <Button
        android:id="@+id/start_transition_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Переход" />

    <ImageView
        android:id="@+id/transition_image"
        android:layout_width="wrap_content"
        android:layout_height="303dp"
        android:src="@drawable/transition"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="93dp" />

</android.support.constraint.ConstraintLayout>
```

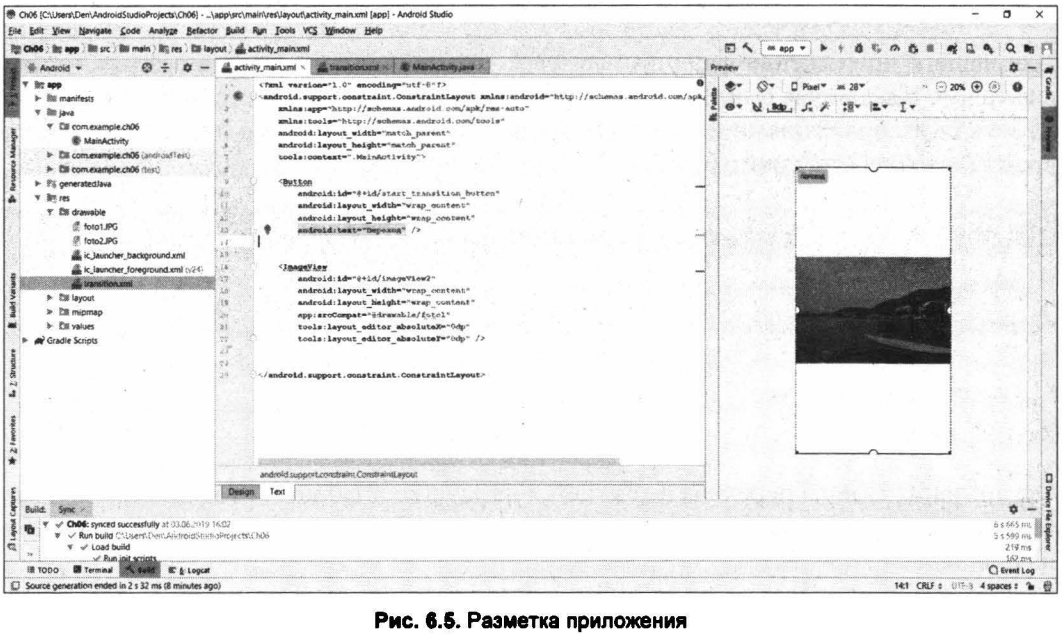


Рис. 6.5. Разметка приложения

Как видите, разметка очень проста. В ней есть кнопка **Переход** и элемент `ImageView` — в него мы будем загружать изображения. Обратите внимание, что в качестве источника изображения мы задаем путь не такой: **@drawable\foto1**, а такой: **@drawable\transition**, т. е. сразу указываем файл с переходами.

Теперь рассмотрим программный код:

```
// Находим наш imageView
ImageView imageView = findViewById(R.id.transition_image);
final TransitionDrawable transitionDrawableransitionDrawable =
                                                    (TransitionDrawable) imageView
        .getDrawable();
```

```
// Находим кнопку Start Transition и устанавливаем для нее обработчик
// нажатия (OnClickListener)
Button startTransition = findViewById(R.id.start_transition_button);
startTransition.setOnClickListener(new Button.OnClickListener() {
    @Override
    public void onClick(View view) {
        transitionDrawableransitionDrawable.startTransition(1000);
    }
});
```

При желании можно сделать кнопку **Назад**, обеспечивающую показ изображений в обратном порядке. Код разметки тогда будет выглядеть так:

```
<Button
    android:id="@+id/reverse_transition_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Назад"
    tools:layout_editor_absoluteX="0dp"
    tools:layout_editor_absoluteY="373dp" />
```

А программный код — так:

```
Button reverseTransition = findViewById(R.id.reverse_transition_button);
reverseTransition.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View view) {
            transitionDrawableransitionDrawable.reverseTransition(1000);
        }
    });
```

Программный код нужно вставить в обработчик `OnCreate`, находящийся в файле MainActivity.java (полный код обработчика приведен в листинге 6.4). По сути, это и есть полный код приложения, поскольку весь остальной код будет сгенерирован средой Android Studio автоматически, и нет смысла приводить его в книге — он будет таким же, как у вас.

Листинг 6.4. Полный код обработчика

```
@Override
protected void onCreate(Bundle savedInstanceState) {
// Код далее сгенерирован автоматически
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);

    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
```

```
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
                                                        Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });
// Конец автоматически сгенерированного кода

// Код приложения
        ImageView imageView = findViewById(R.id.transition_image);
        final TransitionDrawable transitionDrawableransitionDrawable =
                                                (TransitionDrawable) imageView
                .getDrawable();

        Button startTransition = findViewById(R.id.start_transition_button);
        startTransition.setOnClickListener(new Button.OnClickListener() {
            @Override
            public void onClick(View view) {
                transitionDrawableransitionDrawable.startTransition(1000);
            }
        });

    }
```

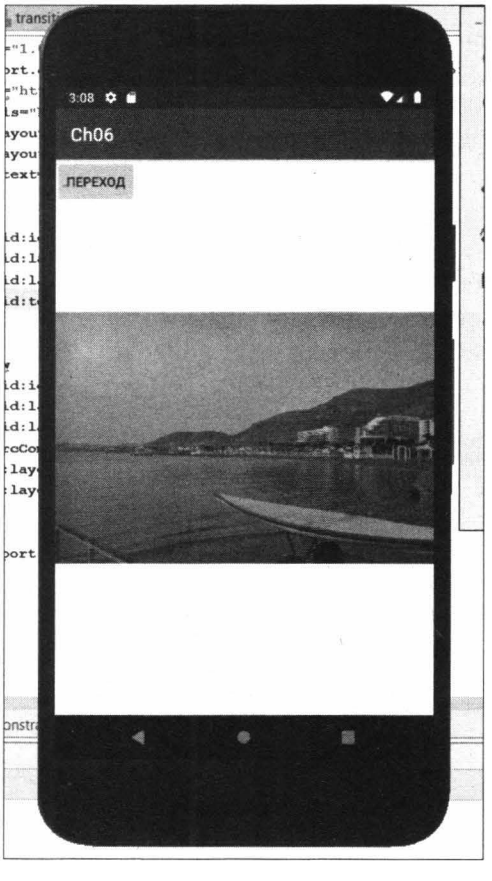


Рис. 6.6. Приложение в эмуляторе

Запустите приложение (рис. 6.6) и щелкните на кнопке **Переход** — в `ImageView` будет загружено другое изображение. Нажмите на кнопку еще раз — будет опять загружено первое изображение, и т. д.

6.3. Класс *ShapeDrawable*

Класс `ShapeDrawable` используется для рисования двумерных фигур: линий, овалов, прямоугольников и т. п. Создание таких графических примитивов обеспечивается следующим набором производных классов:

- `PathShape` — множественный контур;
- `RectShape` — прямоугольник;
- `ArcShape` — дуга;
- `OvalShape` — овал;
- `RoundRectShape` — прямоугольник с закругленными углами.

Класс `ShapeDrawable` является расширением класса `Drawable`, поэтому вы можете использовать его так же, как и класс `Drawable`. Например, можно установить прозрачность методом `setAlpha()` или цветовой фильтр методом `setColorFilter()`.

В конструкторе `ShapeDrawable` рисуемый по умолчанию примитив определяется как `RectShape` (прямоугольник). Но для объекта `ShapeDrawable` нужно установить границы — если этого не сделать, примитив не будет нарисован. Требуется установить и цвет границы, иначе фигура будет черной, и вы ее не увидите.

Класс `RectShape` можно использовать также для рисования горизонтальных и вертикальных линий. Для этого задайте ширину (или высоту) прямоугольника в 1–2 пиксела:

```
// создаем фигуру (прямоугольник)
ShapeDrawable shape = new ShapeDrawable(new RectShape());
// устанавливаем высоту 1 пиксел, ширину - 200 пикселов
shape.setIntrinsicHeight(1);
shape.setIntrinsicWidth(200);
// устанавливаем цвет - синий
shape.getPaint().setColor(Color.BLUE);
```

С той же целью можно использовать и овал:

```
// создаем фигуру (овал)
ShapeDrawable shape = new ShapeDrawable(new OvalShape());
// устанавливаем высоту 1 пиксел, ширину - 200 пикселов
shape.setIntrinsicHeight(1);
shape.setIntrinsicWidth(200);
// устанавливаем цвет - красный
shape.getPaint().setColor(Color.RED);
```

Нарисовать прямоугольник с закругленными углами (RoundRectShape) сложнее. Конструктор RoundRectShape выглядит так:

```
RoundRectShape(float[] outerRad, RectF inset, float[] innerRad)
```

Рассмотрим параметры этого конструктора:

- outerRad — массив из восьми значений радиуса закругления углов внешнего прямоугольника. Первые два значения соответствуют верхнему левому углу, вторые — верхнему правому, третьи — нижнему правому, четвертые — нижнему левому;
- inset — объект класса RectF, определяет расстояние от внутреннего прямоугольника до каждой стороны внешнего прямоугольника. Конструктор объекта RectF принимает четыре параметра: пары X и Y координат левого верхнего и правого нижнего углов внутреннего прямоугольника;
- innerRad — массив из восьми значений радиуса закругления углов внутреннего прямоугольника. Пары значений такие же, как в случае с outerRad, а если на внутреннем прямоугольнике закруглений не предусмотрено, передается null.

Пример:

```
float[] oR = new float[] { 5, 5, 5, 5, 5, 5, 5, 5 };
float[] iR = new float[] { 5, 5, 5, 5, 5, 5, 5, 5 };
RectF rf = new RectF(8, 8, 8, 8);

// создаем фигуру (RoundRect)
ShapeDrawable shape = new ShapeDrawable(new RoundRectShape(oR, rf, iR));
// устанавливаем высоту и ширину
shape.setIntrinsicHeight(150);
shape.setIntrinsicWidth(200);
// устанавливаем цвет - красный
shape.getPaint().setColor(Color.RED);
```

Класс Path используется для создания множественного контура (пригодится для построения графов и других фигур сложной формы), состоящего из линий, квадратичных и кубических кривых. Для установки точек используется метод moveTo(), а для создания линий — метод lineTo(), параметры метода — это координаты конечной точки линии, а координаты начальной точки задаются методом moveTo().

Вот пример использования класса Path:

```
Path p = new Path();
p.moveTo(60, 0);
p.lineTo(100,150);
p.lineTo(200,250);
ShapeDrawable shape = new ShapeDrawable(new PathSpahe(p, 100, 100));
shape.setIntrinsicHeight(200);
shape.setIntrinsicWidth(250);
shape.getPaint().setColor(Color.GREEN);
```

Класс `ArcShape()` используется для создания дуги. Его конструктор принимает два параметра типа `float`. Первый из них задает угол начала прорисовки дуги в градусах, а второй — угловой размер дуги, тоже в градусах:

```
ArcShape (float startAngle, float sweepAngle);
```

* * *

В следующей главе мы поговорим о работе с мультимедиа, а к рисованию еще вернемся в *главе 14*, когда будем рассматривать анимацию.

ГЛАВА 7

Мультимедиа

7.1. Форматы мультимедиа, поддерживаемые ОС Android

В предыдущей главе была рассмотрена поддержка графических форматов, и вы узнали, что Android поддерживает популярные форматы JPG, GIF и PNG. В этой главе мы поговорим о поддержке аудио- и видеоформатов. Android поддерживает практически все популярные мультимедиаформаты: MP3, MP4, FLAC, AAC, OGG и ряд даже не очень популярных. Поддерживаемые аудио/видеоформаты представлены в табл. 7.1.

Таблица 7.1. *Поддерживаемые аудио/видеоформаты*

Тип мультимедиа	Сжатие	Поддерживаемые действия	Форматы
Видео	Почти без потерь	Воспроизведение	H.264
	С минимальными потерями	Запись/воспроизведение	H.263, MPEG-4 SP
Аудио (музыка)	Без сжатия	Запись/воспроизведение	PCM
	Без сжатия	Воспроизведение	WAVE
	Без потерь	Не поддерживается	FLAC
	С минимальными потерями	Воспроизведение	MP3, MP4, OGG, AAC, HE-AACv1, HE-AACv2
	Midi	Воспроизведение	MID, RXT, XMF, OTA, IMY, RTTTL
Аудио (речь)	С минимальными потерями	Запись/воспроизведение	AMR-NB
	С минимальными потерями	Воспроизведение	AMR-WB

Если в вашем приложении планируется запись аудио или видео, то в файл манифеста нужно добавить соответствующие разрешения:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.RECORD_VIDEO"/>
```

> ***Примечание***
>
> Ранее было сказано, что в Android 6 некоторые возможности — например, номер уровня API — «переехали» из файла манифеста в Gradle Scripts. Так вот, разрешений это не коснулось, и они все еще определяются в файле манифеста.

7.2. Работа со звуком

Существуют два различных способа записи и воспроизведения аудироликов:

- `MediaPlayer/MediaRecorder` — классы, обеспечивающие стандартный метод работы со звуком. Звук при этом либо хранится в файле, либо может быть представлен как потоковые данные. Для обработки аудио создается отдельный поток;
- `AudioTrack/AudioRecorder` — прямой (raw) доступ к аудиоданным. Полезен при манипуляциях со звуком в памяти, записи звука в буфер при воспроизведении или в любых других случаях, не требующих наличия потока или файла. Отдельный поток при обработке звука не создается.

7.2.1. Используем *MediaPlayer*

Рассмотрим, как можно воспроизвести звук с помощью класса `MediaPlayer`. Подключается он так:

```
import android.media.MediaPlayer;
```

Затем создать экземпляр `MediaPlayer`:

```
MediaPlayer music = new MediaPlayer();
```

После этого указываем источник звука (в нашем случае это будет RAW-ресурс) и загружаем звуковой файл:

```
music = MediaPlayer.create(this, R.raw.music1);
music.setDataSource(path);
music.prepare();
```

Выполнив все подготовительные работы, запускаем воспроизведение методом `start()`:

```
music.start();
```

Метод `pause()` приостанавливает воспроизведение, продолжить воспроизведение можно снова методом `start()`:

```
music.pause();
music.start();
```

Остановить воспроизведение и освободить ресурсы можно методами `stop()` и `release()` соответственно:

```
music.stop();          // останавливаем воспроизведение
music.release();       // освобождаем ресурсы
```

Класс `MediaPlayer` поддерживает воспроизведение не только с локального устройства. Можно воспроизводить музыку, которая находится на удаленном сервере, например:

```
final String DATA_HTTP = "http://example.com/music.mp3";
final String DATA_STREAM = "http://online.example.com:8101/rr_128";

mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(DATA_HTTP);
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setOnPreparedListener(this);
mediaPlayer.prepareAsync();
```

Аналогично можно запустить воспроизведение потокового аудио:

```
mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(DATA_STREAM);
mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
mediaPlayer.setOnPreparedListener(this);
mediaPlayer.prepareAsync();
```

7.2.2. Использование *MediaRecorder*: запись звука

Прежде, чем приступить к записи звука, нужно определиться с его источником (свойство `MediaRecorder.AudioSource`):

- `MIC` — встроенный микрофон;
- `VOICE_UPLINK` — исходящий голосовой поток при телефонном звонке (то, что вы говорите);
- `VOICE_DOWNLINK` — входящий голосовой поток при телефонном звонке (то, что говорит ваш собеседник);
- `VOICE_CALL` — запись телефонного звонка;
- `CAMCORDER` — микрофон, связанный с камерой, если таковой доступен;
- `VOICE_RECOGNITION` — микрофон, используемый для распознавания голоса, если таковой доступен.

После выбора источника звука нужно задать формат записываемого звука (свойство `MediaRecorder.OutputFormat`):

- `THREE_GPP` — формат 3GPP;
- `MPEG_4` — формат MPEG4;
- `AMR_NB` — формат AMR_NB, лучше всего подходит для речи.

Рассмотрим последовательность действий для записи звука:

1. Создание экземпляра `MediaRecorder`.
2. Указание источника звука — например, микрофона.
3. Установка результирующего формата и сжатия звука.
4. Установка пути к файлу, в котором будет сохранен звук.
5. Подготовка и запуск записи.
6. Остановка записи.

Далее приведен код для каждого из этих этапов:

```
// 1
MediaRecorder record = new MediaRecorder();
// 2
record.setAudioSource(MediaRecorder.AudioSource.MIC);
// 3
record.setOutputFormat(MediaRecorder.OutputFormat.AMR_NB);
record.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
// 4
record.setOutputFile(path);
// 5
record.prepare();
record.start();
// 6
record.stop();
```

Для поддержки `MediaRecorder` нужно подключить следующий пакет:

```
import android.media.MediaRecorder;
```

7.2.3. Использование *AudioRecord* и *AudioTrack*

Возможностей `MediaPlayer/Media.Recorder` в большинстве случаев должно хватить. Но для манипуляции прямыми (raw) аудиоданными, полученными, например, непосредственно из микрофона, нужно использовать классы `AudioRecord` и `AudioTrack`. Первый класс служит для записи звука, второй — для его воспроизведения.

Для непосредственной манипуляции со звуком вам нужно подключить следующие пакеты:

```
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
```

Чтобы приложение могло записывать данные с помощью `AudioRecord`, надо в файле манифеста объявить соответствующее разрешение:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Теперь рассмотрим процесс записи с помощью AudioRecord. Прежде всего нужно создать экземпляр AudioRecord и установить источник звука:

```
short[] Audio1 = new short[10000];
AudioRecord audioRecord = new AudioRecord(
MediaRecorder.AudioSource.MIC, 11025,
AudioFormat.CHANNEL_IN_MONO,
AudioFormat.ENCODING_PCM_16BIT, 10000);
```

Наша конфигурация подходит для записи голоса со встроенного микрофона в буфер myAudio. Мы записываем 11 025 образцов в секунду, а размер буфера — 10 000 образцов, следовательно, длительность записи — менее секунды. Для более длительной записи нужно увеличить размер буфера.

- Первый параметр — это источник звука. Вы можете использовать свойство MediaRecorder.AudioSource для указания источника звука.
- Второй параметр (11 025) — это частота звука (в Гц). Такая частота подходит только для записи голоса, для CD-качества необходима частота 44 100.
- Мы записываем монозвук, для записи стерео измените третий параметр — его значение должно выглядеть так: AudioFormat.CHANNEL_IN_STEREO.
- Кодирование звука задается четвертым параметром. Здесь мы можем использовать либо 16-битное кодирование (что мы и делаем), либо 8-битное (AudioFormat.ENCODING_PCM_8BIT).
- Последний параметр — это размер буфера в байтах, в который будет производиться запись. Другими словами — это общий размер выделенной памяти. Для правильного задания этого параметра лучше было бы воспользоваться методом getMinBufferSize(), но для упрощения примера мы указали значение в байтах.

Далее нужно начать запись:

```
audioRecord.startRecording();
```

Поскольку у нас прямой доступ к микрофону, то просто указать файл, в который нужно поместить прочитанный звук, нельзя. Нужно еще вручную считать этот звук с микрофона. Для этого используется метод read():

```
audioRecord.read(Audio1, 0, 10000);
```

Остановить запись можно методом stop():

```
audioRecord.stop();
```

Теперь поговорим о воспроизведении звука средствами AudioTrack. Конструктору объекта AudioTrack нужно передать:

- тип потока: AudioManager.STREAM_MUSIC (микрофон) или STREAM_VOICE_CALL (голосовой звонок). Другие варианты используются реже;
- частоту в герцах — значения такие же, как и для записи звука;
- конфигурацию канала: AudioFormat.CHANNEL_OUT_STEREO или AudioFormat.CHANNEL_OUT_MONO. Можно также использовать значение CHANNEL_OUT_5POINT1 для звука 5.1;

- тип кодирования звука — значения такие же, как и для записи;
- размер буфера в байтах;
- режим буфера: `AudioTrack.MODE_STATIC` (подходит для небольших звуков, которые полностью помещаются в памяти) или `AudioTrack.MODE_STREAM` (для потокового звука).

Вот пример инициализации объекта `AudioTrack`:

```
AudioTrack audioTrack = new AudioTrack(
AudioManager.STREAM_MUSIC, 11025,
AudioFormat.CHANNEL_OUT_MONO,
AudioFormat.ENCODING_PCM_16BIT, 4096,
AudioTrack.MODE_STREAM);
```

Далее нужно начать воспроизведение методом `play()`. Так как мы все делаем вручную, то должны вручную же записать звуковые данные на устройство воспроизведения. Это делается методом `write()`. Первый параметр этого метода — буфер со звуковыми данными, а третий параметр — размер буфера. Второй параметр — смещение относительно начала буфера. Если смещение равно 0, то воспроизведение будет начато с начала буфера. Остановить воспроизведение можно методом `stop()`:

```
audioTrack.play();
audioTrack.write(myAudio, 0, 10000);
audioTrack.stop();
```

А теперь самое интересное — практический пример. Если в случае с `MediaPlayer` и `MediaRecorder` вы могли разобраться сами, то классы `AudioRecorder` и `AudioTrack` требуют наглядного примера.

Сейчас мы напишем небольшое приложение, главное окно которого будет содержать две кнопки: **Запись** и **Воспр.** По нажатию кнопки **Запись** будет инициирована запись с микрофона на протяжении 5 секунд. По нажатию кнопки **Воспр.** будет воспроизведен записанный звук.

Файл разметки нашего приложения представлен в листинге 7.1.

Листинг 7.1. Файл разметки приложения `AudioDemo`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
<TextView android:id="@+id/status"
        android:text="Ready" android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

```
<Button android:id="@+id/record"
        android:text="Запись"
        android:textSize="20sp" android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
<Button android:id="@+id/play"
        android:text="Воспр." android:textSize="20sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

Созданная согласно листингу 7.1 разметка показана на рис. 7.1, а код приложения `AudioDemo` представлен в листинге 7.2.

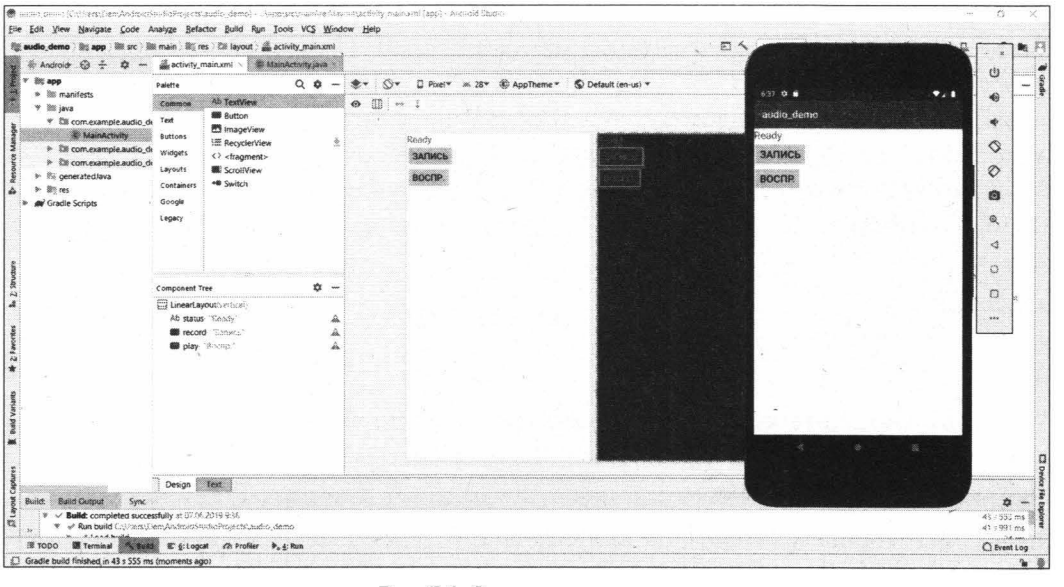


Рис. 7.1. Созданная разметка

Листинг 7.2. Код приложения `AudioDemo`

```
package com.samples.audio_demo;

import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.Button;
import android.widget.TextView;
```

```
// Подключаемые пакеты
import android.media.AudioFormat;
import android.media.AudioManager;
import android.media.AudioRecord;
import android.media.AudioTrack;
import android.media.MediaRecorder;
import android.os.Handler;
import android.util.Log;

public class MainActivity extends AppCompatActivity {

    private TextView statusText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        FloatingActionButton fab = (FloatingActionButton)
findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view, "Replace with your own action",
                                                        Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });

        // Ищем надпись и кнопки в разметке
        statusText = (TextView) findViewById(R.id.status);
        Button actionButton = (Button) findViewById(R.id.record);

// Собственный обработчик для кнопки записи
        actionButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View view) {
                record_thread();
            }
        });

    }

    String text_string;
    final Handler mHandler = new Handler();
    final Runnable mUpdateResults = new Runnable() {
```

```
    public void run() {
        updateResultsInUi(text_string);
    }
};

private void updateResultsInUi(String update_txt) {
    statusText.setText(update_txt);
}

// Поток записи
private void record_thread() {
    Thread thread = new Thread(new Runnable() {
        public void run() {
            text_string = "Записываем...";
            mHandler.post(mUpdateResults);
            record();
            text_string = "Все!";
            mHandler.post(mUpdateResults);
        }
    });
    thread.start();
}

// Параметры записи звука
private int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
int frequency = 11025;      //Гц
// Размер буфера записи
int bufferSize = 50*AudioTrack.getMinBufferSize(frequency,
        AudioFormat.CHANNEL_OUT_MONO, audioEncoding);

// Объект AudioRecord для записи звука
public AudioRecord audioRecord = new AudioRecord(
        MediaRecorder.AudioSource.MIC,
        frequency, AudioFormat.CHANNEL_IN_MONO,
        audioEncoding, bufferSize);

// Объект AudioTrack с теми же параметрами
public AudioTrack audioTrack = new AudioTrack(
        AudioManager.STREAM_MUSIC, frequency,
        AudioFormat.CHANNEL_OUT_MONO,
        audioEncoding, 4096,
        AudioTrack.MODE_STREAM);

// Буфер для звука
short[] buffer = new short[bufferSize];
```

```
    // Функция записи звука
    public void record() {
        try {
            // Начинаем запись
            audioRecord.startRecording();
// Читаем звук в буфер
            audioRecord.read(buffer, 0, bufferSize);
// Останавливаем запись
            audioRecord.stop();
        } catch (Throwable t) {
            Log.e("AudioDemo","Ошибка!");
        }
    }

    // Воспроизведение звука
    public void run() {
        int i=0;
        while(i<bufferSize) {
            audioTrack.write(buffer, i++, 1);
        }
        return;
    }

    // Действие при паузе
    @Override
    protected void onPause() {
        if(audioTrack!=null) {
            if(audioTrack.getPlayState()==AudioTrack.PLAYSTATE_PLAYING) {
                audioTrack.pause();
            }
        }
        super.onPause();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
```

```
        // noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

7.3. Работаем с видео

Для записи и воспроизведения видео используются уже знакомые нам классы `MediaRecorder` и `MediaPlayer`. Для записи видео нужно добавить в файл манифеста следующую строку:

```
<uses-permission android:name="android.permission.RECORD_VIDEO" />
```

Источником видео (свойство `MediaRecorder.VideoSource`) может быть только встроенная камера — `MediaRecorder.VideoSource.CAMERA`.

Формат видео задается свойством `OutputFormat`: `THREE_GPP` (формат 3GPP, в последнее время активно используется для записи мобильного видео) или `MPEG_4` (популярный формат MPEG4).

Кодек видео можно выбрать с помощью свойства `MediaRecorder.VideoEncoder`: `H264` (кодек H.264), `H263` (кодек H.263), `MPEG_4_SP` (кодек MPEG4).

Рассмотрим последовательность действий при записи видео. Как обычно, сначала мы создаем объект класса `MediaRecorder`:

```
MediaRecorder Video = new MediaRecorder();
```

Указываем источник видео (встроенная камера):

```
Video.setVideoSource(MediaRecorder.VideoSource.CAMERA);
```

Затем устанавливаем формат файла и кодек:

```
Video.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
Video.setAudioEncoder(MediaRecorder.AudioEncoder.H263);
```

Путь к файлу, в котором будет сохранено видео, задается так:

```
Video.setOutputFile(path);
```

Осталось только начать запись:

```
Video.prepare();
Video.start();
```

Остановить запись можно методом `stop()`.

Рассмотрим теперь воспроизведение видео:

```
// Создаем новый объект
MediaPlayer player = new MediaPlayer();
```

```
// Загружаем видео из ресурса
player = MediaPlayer.create(this, R.raw.my_video1);
// Аналогично можно загрузить видео из файла с помощью
// следующих двух операторов
// player.setDataSource(path);
// player.prepare();
// Запускаем воспроизведение видео
player.start();
// Останавливаем воспроизведение и освобождаем ресурсы:
player.stop();
player.release();
```

Примечание

Дополнительный пример приложения, использующего `MediaRecorder`, можно получить по адресу: https://github.com/googlesamples/android-MediaRecorder. Для сборки этого проекта уровень SDK должен быть 23. Хотя минимальный уровень API для самого приложения — 14.

* * *

В следующей главе мы поговорим о способах хранения данных на Android-устройствах. А в *главе 13* будет рассмотрена база данных SQLite.

ГЛАВА 8

Доступ к данным

8.1. Методы доступа к данным

Нужно отметить, что с момента выхода последнего издания книги API существенно поменялся, и эту главу пришлось переписать заново.

В Android-приложениях предлагаются три метода доступа к данным: непосредственный (direct), предпочтения и база данных. В этой главе будут рассмотрены первые два способа: мы создадим простейший редактор заметок — приложение, позволяющее сохранить заметки в файл, а потом прочитать их, а также познакомимся с приложением, работающим с предпочтениями. Доступ к базе данных будет рассмотрен в *главе 13*.

Приложениям часто бывает нужно сохранять небольшие фрагменты данных — например, имя пользователя, дату его рождения, какие-то настройки и т. п. Теоретически можно было бы создать некий текстовый файл со всеми этими данными, но тогда вам пришлось бы писать логику обработки этого файла, а это лишнее — зачем «изобретать колесо», если в Android уже все необходимое предусмотрено? В этих целях нужно использовать *предпочтения* (настройки). Во всех остальных случаях надо применять или базу данных (более сложные приложения), или непосредственный доступ к файлам (приложения попроще).

8.2. Работа с файловой системой

Использование настроек (предпочтений) очень удобно, но когда нам нужно работать с большими массивами данных — например, обращаться к графическим файлам, придется работать с файловой системой.

Как известно, Android построен на основе Linux. Этот факт находит свое отражение в работе с файлами. В качестве разделителя в именах файлах используется слеш «/», а не обратный слеш «\», как в Windows. Названия файлов являются регистрозависимыми, т. е. file.txt — это не то же самое, что File.TXT.

Приложение Android хранит свои данные в каталоге /data/data/<название_пакета>/ и, как правило, относительно этого каталога будет идти работа с файлами.

Для работы с файлами абстрактный класс `android.content.Context` определяет ряд методов:

- `deleteFile(String name)` — удаляет определенный файл;
- `fileList()` — получает все файлы, которые содержатся в подкаталоге /files в каталоге приложения;
- `getCacheDir()` — получает ссылку на подкаталог cache в каталоге приложения;
- `getDir(String dirName, int mode)` — получает ссылку на подкаталог в каталоге приложения, если такого подкаталога нет, то он создается;
- `getExternalCacheDir()` — получает ссылку на папку /cache внешней файловой системы устройства;
- `getExternalFilesDir()` — получает ссылку на каталог /files внешней файловой системы устройства;
- `getFileStreamPath(String filename)` — возвращает абсолютный путь к файлу в файловой системе;
- `openFileInput(String filename)` — открывает файл для чтения;
- `openFileOutput (String name, int mode)` — открывает файл для записи.

Все файлы, которые создаются и редактируются в приложении, как правило, хранятся в подкаталоге /files в каталоге приложения (рис. 8.1).

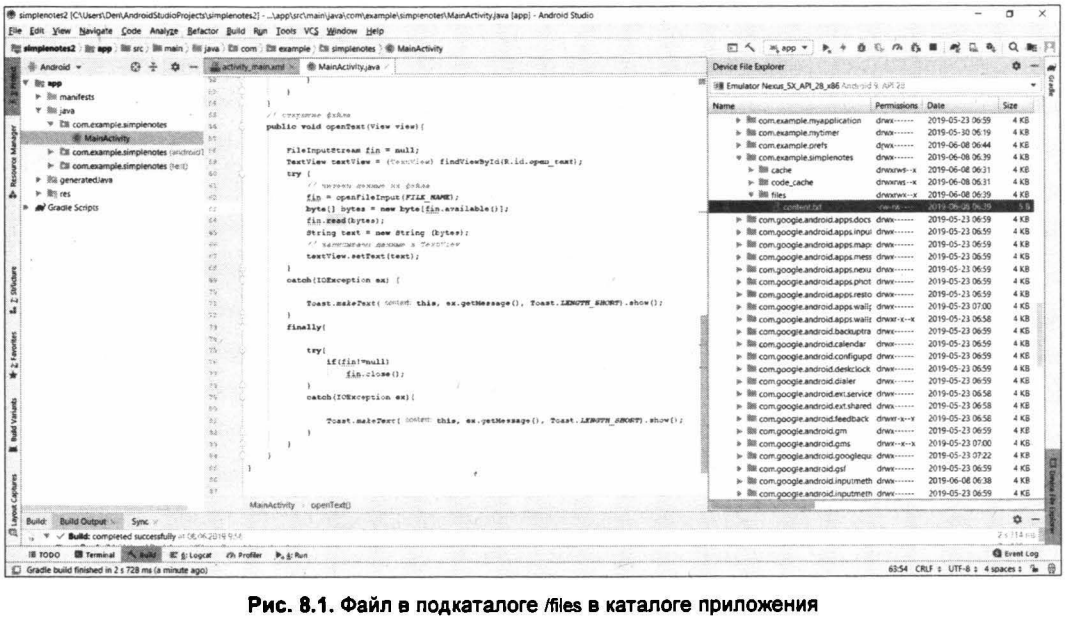


Рис. 8.1. Файл в подкаталоге /files в каталоге приложения

Для непосредственного чтения и записи файлов применяются также стандартные классы Java из пакета java.io.

Итак, применим в приложении функционал чтения/записи файлов. Пусть у нас будет следующая примитивная разметка `layout` (листинг 8.1).

Листинг 8.1. Разметка приложения

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <EditText
        android:id="@+id/save_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_margin="16dp"
        android:layout_gravity="center"
        android:onClick="saveText"
        android:text="Сохранить"/>

    <TextView
        android:layout_marginTop="80dp"
        android:id="@+id/open_text"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:gravity="start"
        android:layout_weight="4"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_margin="16dp"
        android:layout_weight="1"
        android:layout_gravity="center"
        android:onClick="openText"
        android:text="Открыть"/>

</LinearLayout>
```

Обратите внимание, как мы используем атрибут `onClick`, чтобы сразу задать обработчик нажатия кнопки.

Поле `EditText` предназначено для ввода текста, а объект `TextView` — для вывода ранее сохраненного текста. Для сохранения и восстановления текста добавлены две кнопки.

Теперь в коде активности нужно определить код обработчиков кнопок: один обработчик будет сохранять текст, который ввел пользователь в `EditText`, а второй —

выводить в TextView ранее сохраненный текст из файла. Полный код приложения (файл MainActivity.java) приведен в листинге 8.2.

Листинг 8.2. Полный код приложения (файл MainActivity.java)

```
package com.example.simplenotes;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class MainActivity extends AppCompatActivity {

    private final static String FILE_NAME = "content.txt";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // сохранение файла
    public void saveText(View view){

        FileOutputStream fos = null;
        try {
            // Получаем содержимое EditText
            EditText textBox = (EditText) findViewById(R.id.save_text);
            String text = textBox.getText().toString();

            // Подготавливаем файл для вывода
            fos = openFileOutput(FILE_NAME, MODE_PRIVATE);
            // Записываем данные в файл
            fos.write(text.getBytes());
            // Выводим уведомление
            Toast.makeText(this, "Файл сохранен", Toast.LENGTH_SHORT).show();
        }
        catch(IOException ex) {
            // В случае исключения ввода/вывода выводим информацию
            // для отладки
            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
```

```
        finally{
            try{
                if(fos!=null)
                    fos.close();
            }
            catch(IOException ex){

                Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    }
    // открытие файла
    public void openText(View view){

        FileInputStream fin = null;
        TextView textView = (TextView) findViewById(R.id.open_text);
        try {
            // читаем данные из файла
            fin = openFileInput(FILE_NAME);
            byte[] bytes = new byte[fin.available()];
            fin.read(bytes);
            String text = new String (bytes);
            // записываем данные в TextView
            textView.setText(text);
        }
        catch(IOException ex) {

            Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
        }
        finally{

            try{
                if(fin!=null)
                    fin.close();
            }
            catch(IOException ex){

                Toast.makeText(this, ex.getMessage(), Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

Система позволяет создавать файлы с двумя разными режимами:

- MODE_PRIVATE — файлы могут быть доступны только владельцу приложения (режим по умолчанию);
- MODE_APPEND — данные могут быть добавлены в конец файла.

Как можно видеть из листинга 8.2, по нажатию на кнопку сохранения создается поток вывода в режиме `MODE_PRIVATE`:

```
FileOutputStream fos = openFileOutput(FILE_NAME, MODE_PRIVATE)
```

и введенный текст сохраняется в файл data/data/название_пакета/files/content.txt (см. рис. 8.1).

Поэтому если файл content.txt по указанному пути уже существует, то он будет перезаписан. Если же нам нужно было дописать данные в файл, тогда надо было бы использовать режим `MODE_APPEND`:

```
FileOutputStream fos = openFileOutput(FILE_NAME, MODE_APPEND);
```

Для чтения файла применяется поток ввода `FileInputStream`:

```
FileInputStream fin = openFileInput(FILE_NAME);
```

Подробнее про использование потоков ввода/вывода можно прочитать в официальном руководстве **https://developer.android.com/training/data-storage/files**. А мы пока посмотрим, какое приложение у нас получилось.

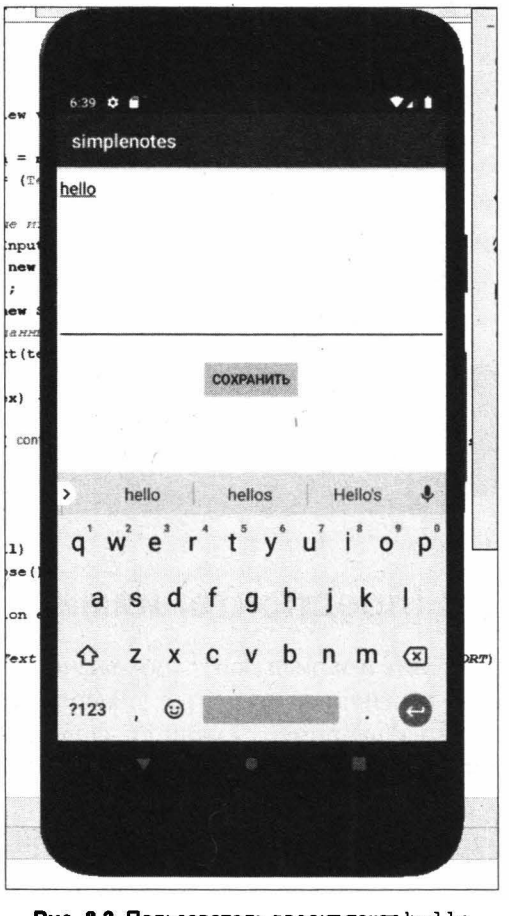


Рис. 8.2. Пользователь вводит текст `hello`

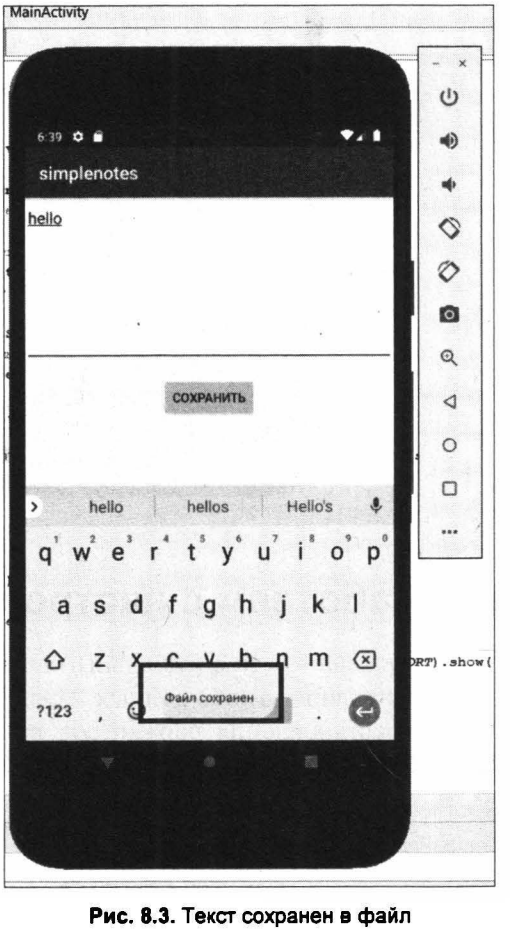


Рис. 8.3. Текст сохранен в файл

На рис. 8.2 показано, что пользователь вводит в `EditText` текст `hello`, а на рис. 8.3 — что текст сохранен (выведено соответствующее уведомление).

На рис. 8.4 показано, что пользователь нажал кнопку **Открыть**, и приложение отобразило сохраненный ранее в файле content.txt текст (напомним, что путь к этому файлу показан на рис. 8.1).

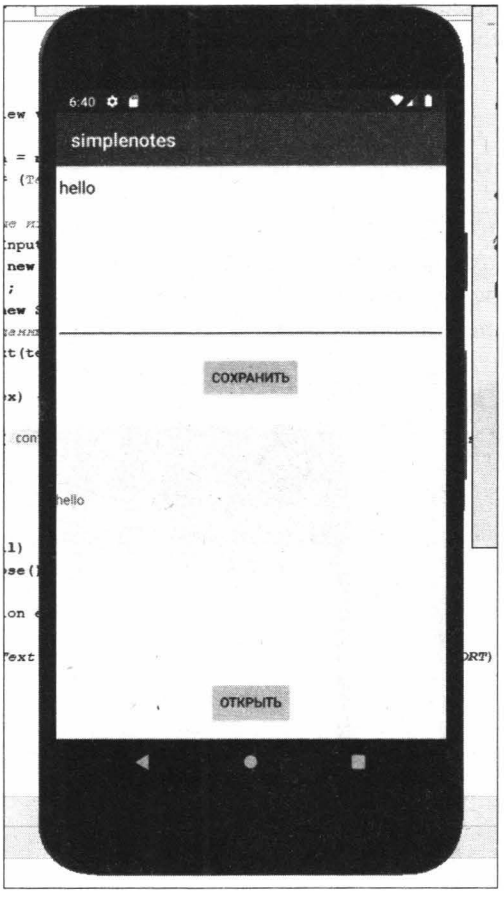


Рис. 8.4. Приложение прочитало текст из файла

8.3. Работаем с настройками (предпочтениями)

Предпочтения — это специальный механизм чтения и записи пар «ключ-значение» для примитивных типов данных. Предпочтения обычно используются для сохранения и восстановления параметров приложения — например, имени графической темы (скина), текста приветствия, названия шрифта и т. п.

В качестве их значений могут выступать данные следующих типов: `Boolean`, `Float`, `Integer`, `Long`, `String`, набор строк.

Предпочтения (настройки) бывают общими для всех активностей в приложении, но также могут быть и настройки непосредственно для отдельных активностей.

Предпочтения хранятся в XML-файлах в незашифрованном виде в локальном хранилище. Они невидимы, поэтому для простого пользователя недоступны.

При работе с предпочтениями следует учитывать следующие моменты. Поскольку они хранятся в незашифрованном виде, то не рекомендуется сохранять в них чувствительные данные типа паролей или номеров банковских карт. Кроме того, они представляют данные, ассоциированные с приложением, и через панель управления приложениями пользователь может удалить эти данные вместе с приложением.

Существуют разные типы доступа к предпочтениям:

- `MODE_PRIVATE` — предпочтение доступно только создавшему его приложению;
- `MODE_WORLD_READABLE` — предпочтение доступно всем приложениям для чтения;
- `MODE_WORLD_WRITEABLE` — предпочтение доступно всем приложениям для записи.

Для работы с разделяемыми предпочтениями в классе `Activity` (точнее, в его базовом классе `Context`) имеется метод `getSharedPreferences()`:

```
import android.content.SharedPreferences;

//.........................

SharedPreferences settings = getSharedPreferences("PreferencesName",
                                                   MODE_PRIVATE);
```

Первый параметр метода указывает на название предпочтения. В показанном случае это название: `"PreferencesName"`. Если предпочтений с подобным названием нет, то они создаются при вызове этого метода. Второй параметр указывает на режим доступа. В нашем случае режим описан константой `MODE_PRIVATE`.

Класс `android.content.SharedPreferences` предоставляет ряд методов для управления предпочтениями:

- `contains(String key)` — возвращает `true`, если в настройках сохранено значение с ключом `key`;
- `getAll()` — возвращает все сохраненные в настройках значения;
- `getBoolean (String key, boolean defValue)` — возвращает из настроек значение типа `Boolean`, которое имеет ключ `key`. Если элемента с таким ключом не окажется, то возвращается значение `defValue`, передаваемое вторым параметром;
- `getFloat(String key, float defValue)` — возвращает значение типа `float` с ключом `key`. Если элемента с таким ключом не окажется, то возвращается значение `defValue`;
- `getInt(String key, int defValue)` — возвращает значение типа `int` с ключом `key`;
- `getLong(String key, long defValue)` — возвращает значение типа `long` с ключом `key`;
- `getString(String key, String defValue)` — возвращает строковое значение с ключом `key`;
- `getStringSet(String key, Set<String> defValues)` — возвращает массив строк с ключом `key`;

- `edit()` — возвращает объект `SharedPreferences.Editor`, который используется для редактирования настроек.

Для управления настройками используется объект класса `SharedPreferences.Editor`, возвращаемый методом `edit()`. Он определяет следующие методы:

- `clear()` — удаляет все настройки;
- `remove(String key)` — удаляет из настроек значение с ключом `key`;
- `putBoolean(String key, boolean value)` — добавляет в настройки значение типа `boolean` с ключом `key`;
- `putFloat(String key, float value)` — добавляет в настройки значение типа `float` с ключом `key`;
- `putInt(String key, int value)` — добавляет в настройки значение `int` с ключом `key`;
- `putLong(String key, long value)` — добавляет в настройки значение типа `long` с ключом `key`;
- `putString(String key, String value)` — добавляет в настройки строку с ключом `key`;
- `putStringSet(String key, Set<String> values)` — добавляет в настройки строковый массив;
- `commit()` — подтверждает все изменения в настройках;
- `apply()` — так же, как и метод `commit()`, подтверждает все изменения в настройках, однако измененный объект `SharedPreferences` вначале сохраняется во временной памяти и лишь затем в результате асинхронной операции записывается на мобильное устройство.

Теперь рассмотрим практический пример. Создайте новое приложение с пустой активностью (**Empty Activity**). В файле activity_main.xml определим интерфейс пользователя (листинг 8.3).

Листинг 8.3. Интерфейс приложения (файл activity_main.xml)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
        android:id="@+id/nameBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Введите имя"/>
    <Button
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Сохранить"
        android:onClick="saveName"/>

    <TextView
        android:id="@+id/nameView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Получить имя"
        android:onClick="getName"/>
</LinearLayout>
```

На экране присутствуют две кнопки: для сохранения и для вывода ранее сохраненного значения, а также поле для ввода и текстовое поле для вывода сохраненной настройки (рис. 8.5).

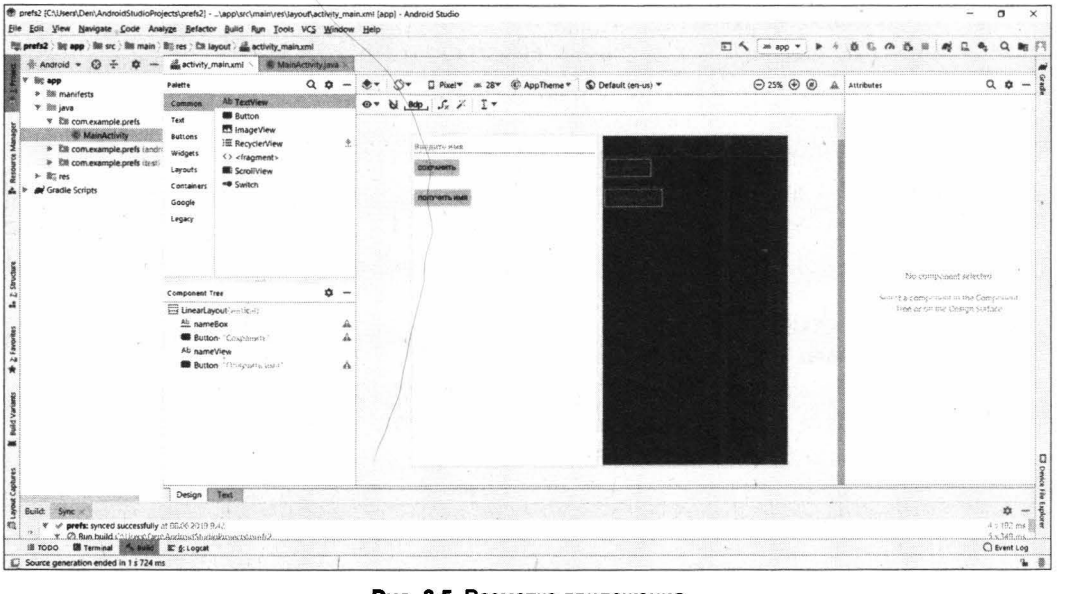


Рис. 8.5. Разметка приложения

Теперь займемся кодом приложения, приведенном в файле MainActivity.java (листинг 8.4).

Листинг 8.4. Полный код приложения (файл MainActivity.java)

```
package com.example.den.prefs;

import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    private static final String PREFS_FILE = "Account";
    private static final String PREF_NAME = "Name";
    SharedPreferences settings;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);
    }

    public void saveName(View view) {
        // получаем введенное имя
        EditText nameBox = (EditText) findViewById(R.id.nameBox);
        String name = nameBox.getText().toString();
        // сохраняем его в настройках
        SharedPreferences.Editor prefEditor = settings.edit();
        prefEditor.putString(PREF_NAME, name);
        prefEditor.apply();
    }

    public void getName(View view) {
        // получаем сохраненное имя
        TextView nameView = (TextView) findViewById(R.id.nameView);
        String name = settings.getString(PREF_NAME,"n/a");
        nameView.setText(name);
    }
}
```

Запустите приложение. Если предпочтения (настройки) нет, приложение отобразит значение по умолчанию — **n/a** (рис. 8.6).

Теперь введите имя (см. рис. 8.6), нажмите кнопку **Сохранить**, а затем кнопку **Получить имя**. Вы должны увидеть введенное ранее имя пользователя (рис. 8.7).

Часто возникает задача автоматически сохранять вводимые данные при выходе пользователя из activity. Для этого мы можем переопределить метод `onPause` (листинг 8.5).

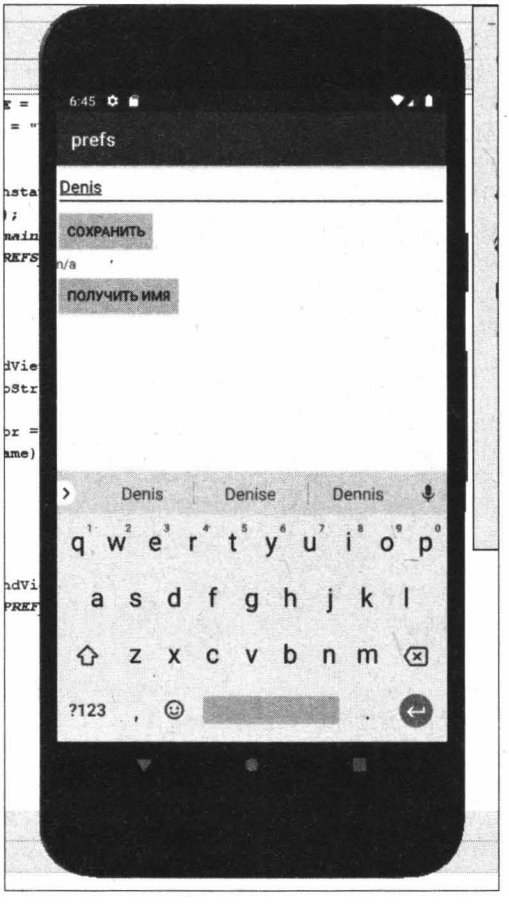


Рис. 8.6. Значение еще не задано

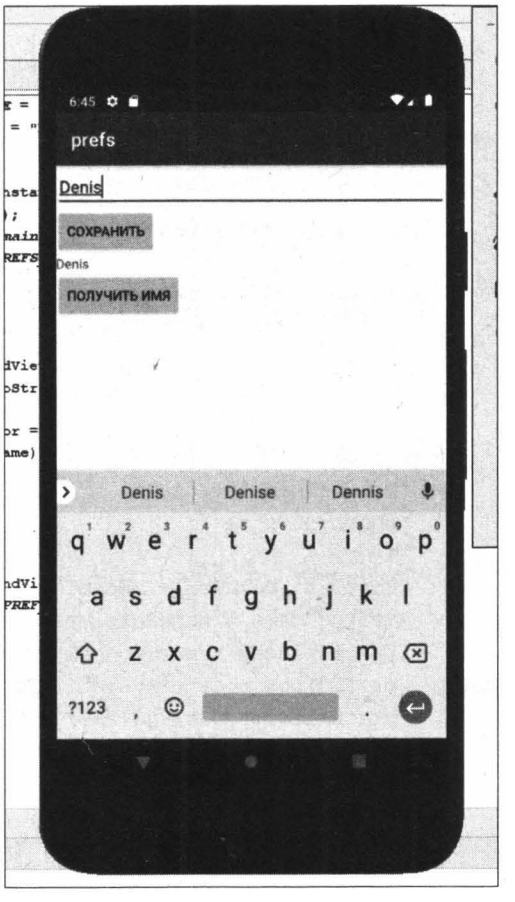


Рис. 8.7. Настройки загружены

Листинг 8.5. Автоматическое сохранение настроек

```
package com.example.den.prefs;

import android.content.SharedPreferences;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    private static final String PREFS_FILE = "Account";
    private static final String PREF_NAME = "Name";
    EditText nameBox;
    SharedPreferences settings;
```

```
    SharedPreferences.Editor prefEditor;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        nameBox = (EditText) findViewById(R.id.nameBox);
        settings = getSharedPreferences(PREFS_FILE, MODE_PRIVATE);

        // получаем настройки
        String name = settings.getString(PREF_NAME,"");
        nameBox.setText(name);
    }

    @Override
    protected void onPause(){
        super.onPause();

        EditText nameBox = (EditText) findViewById(R.id.nameBox);
        String name = nameBox.getText().toString();
        // сохраняем в настройках
        prefEditor = settings.edit();
        prefEditor.putString(PREF_NAME, nameBox.getText().toString());
        prefEditor.apply();
    }

    public void saveName(View view) {

    }

    public void getName(View view) {

    }
}
```

* * *

На этом мы завершаем разговор о базовом программировании для Android и переходим к следующей части книги, в которой будем разбираться с построением сложных Android-приложений.

ЧАСТЬ III

Построение сложного приложения

ГЛАВА 9

Межпроцессное взаимодействие Android-приложений

9.1. Деятельности и намерения

Любое приложение может содержать следующие компоненты:

- `Activity` — деятельность;
- `Service` — служба;
- `BroadcastReceiver` — приемник широковещательных намерений;
- `ContentProvider` — контент-провайдер.

Как минимум один из этих компонентов приложение содержать должно. В самых сложных случаях в приложении будут присутствовать все четыре компонента. Операционная система (и другие Android-приложения при соответствующих разрешениях) может вызывать необходимый ей компонент приложения.

Какие компоненты должны быть именно в вашем приложении, зависит от самого приложения, т. е. от того, что оно должно делать. Так, если нужно организовать интерфейс пользователя (например, получить от пользователя ввод), тогда в вашем приложении обязательно должна присутствовать *деятельность* (`Activity`). При разработке фонового процесса — например, для воспроизведения музыки или фонового обновления деятельности, — вам пригодится *служба* (`Service`). Для обращения к ресурсам телефона — например, для доступа к списку контактов, — вам понадобится *контент-провайдер* (`ContentProvider`). А для получения широковещательных сообщений (с целью реакции на какие-то события телефона) задействуется *приемник широковещательных намерений* (`BroadcastReceiver`).

Намерение (`Intent`) — это действие. Компоненты приложения вызываются с помощью намерений. Так, с помощью намерения вы можете вызвать браузер:

```
Intent Mybrowser = Intent(Intent.ACTION_VIEW);
Mybrowser.setData(Uri.parse("http://bhv.ru"));
startActivity(Mybrowser);
```

Аналогично вы можете вызвать компонент своего собственного приложения. Пусть у вас есть приложение, в котором имеются две (или более) деятельности (два или

более окна). Вызвать вспомогательное окно можно только с помощью намерения.

Представим, что вы разрабатываете приложение, содержащее меню. В меню имеется кнопка **Start**, которая запускает другую деятельность в главном окне. Обработчик нажатия кнопки **Start** будет таким:

```
import android.content.Intent;
...
Intent startButton = new Intent(this, StartButton.class);
startActivity(startButton);
```

В таком проекте должен присутствовать класс `StartButton`. «Болванка» для этого класса приведена в листинге 9.1. Наш класс ничего не делает — он просто находит в файле разметки деятельности кнопку `the_end` и устанавливает для нее обработчик — функцию `finish()`. Напомню: чтобы добавить класс `StartButton` в проект, нужно щелкнуть на проекте (в области **Package Explorer**) правой кнопкой мыши и выбрать команду **New | Java Class**.

Листинг 9.1. Файл StartButton.java

```
package com.example.den.startgame;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class StartButton extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // файл разметки называется app.xml, находится в папке layout проекта
    setContentView(R.layout.app);
    // обработка кнопки завершения программы
    Button endButton = (Button) findViewById(R.id.the_end);
      endButton.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
       finish();
      }
   });
  }
}
```

Чтобы создать файл разметки для деятельности `StartButton`, нужно щелкнуть в дереве проекта (крайняя левая панель Android Studio) на строке res\layout и выбрать команду **New | XML | Layout XML File** (рис. 9.1). В открывшемся окне (рис. 9.2) введите только название файла: app, более ничего туда вписывать не нужно, поскольку мы создадим этот файл вручную (листинг 9.2).

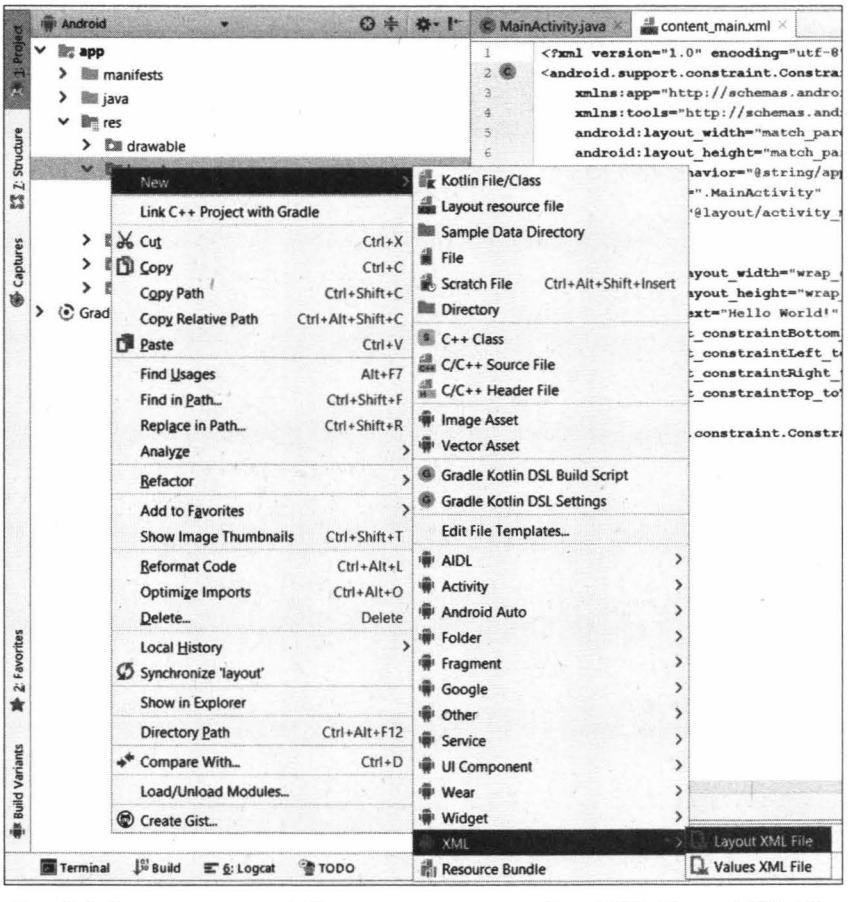


Рис. 9.1. Создание нового файла разметки: команда New | XML | Layout XML File

Листинг 9.2. Файл разметки для деятельности `StartButton` (файл res\layout\app.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button android:id="@+id/the_end"
        android:layout_width="100dip"
        android:layout_height="100dip"
        android:text="@string/the_end"
        android:layout_centerInParent="true"
        />
</LinearLayout>
```

В файл строковых констант strings.xml нужно добавить константу `the_end` (рис. 9.3), а в файле манифеста зарегистрировать нашу новую деятельность и установить для нее действия: `VIEW` и `DEFAULT` (листинг 9.3).

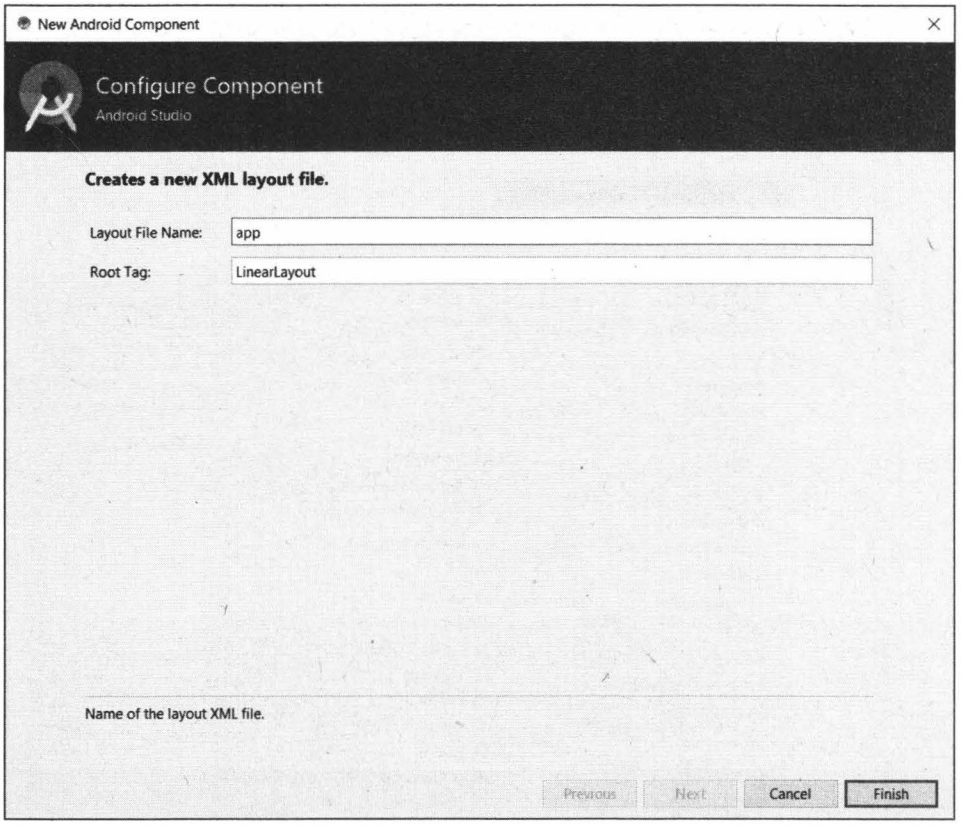


Рис. 9.2. Окно создания нового файла разметки

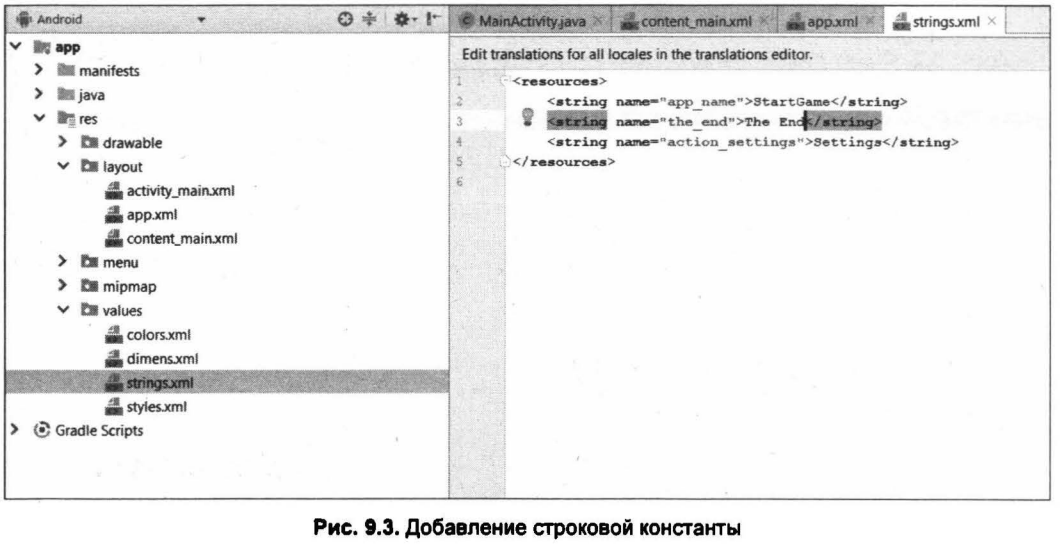


Рис. 9.3. Добавление строковой константы

Листинг 9.3. Файл манифеста для приложения с несколькими деятельностями

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.den.startgame">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".StartButton"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

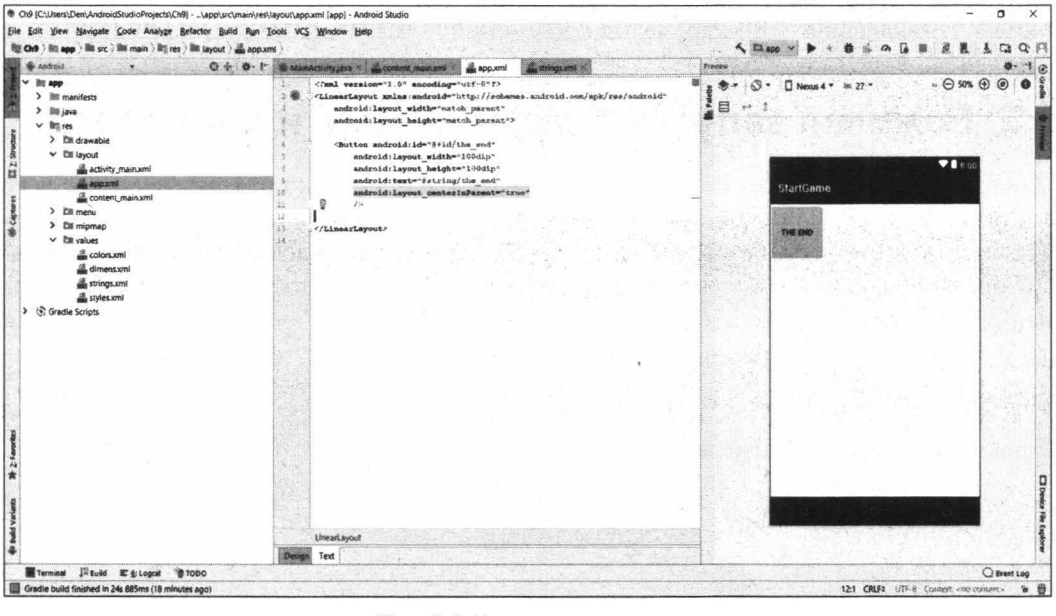


Рис. 9.4. Созданная разметка

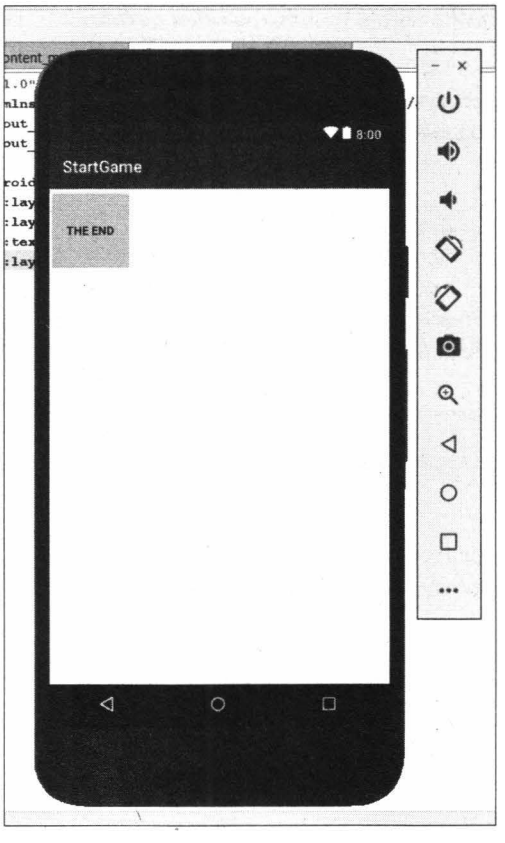


Рис. 9.5. Наше приложение отобразило вторую активность с кнопкой **THE END**

Созданная в соответствии с листингом 9.2 разметка показана на рис. 9.4, а приложение, отображающее вторую активность с кнопкой **THE END**, — на рис. 9.5.

9.2. Режимы запуска *singleInstance* и *singleTask*

Приложение может быть запущено пользователем несколько раз, в результате чего в памяти окажутся запущенными несколько экземпляров одной и той же деятельности, что приведет к ее перерасходу. Чтобы избежать подобной ситуации, разработчик может контролировать поведение каждой деятельности с помощью файла манифеста. Для каждого элемента `activity`, обладающего `intent`-фильтрами `MAIN` и `LAUNCHER`, добавьте следующий параметр:

```
android:launchMode="singleInstance"
```

Должно получиться примерно так:

```
<activity android:name=".Menu"
android:launchMode="singleInstance"
android:label="@string/app_name">
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
```

```
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
   </activity>
```

В результате будет запущен только один экземпляр деятельности.

Но вы можете задать еще более жесткое ограничение. По умолчанию каждая дочерняя деятельность запускается как отдельная задача. Чтобы как одна задача запускались все деятельности, используется следующий режим запуска:

```
android:launchMode="singleTask"
```

Полный код при этом выглядит так:

```
<activity android:name=".Menu"
android:launchMode="singleTask"
android:label="@string/app_name">
<intent-filter>
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

9.3. Сохранение и восстановление состояния деятельности

Ранее мы рассмотрели способ хранения настроек приложения — предпочтения. А сейчас мы познакомимся со способом сохранения и восстановления состояния деятельности. Так, перед завершением работы деятельности вызывается функция `onSaveInstanceState()`, сохраняющая состояние деятельности. При повторном создании деятельности вызывается функция `onRestoreInstanceState()`. Эта функция используется для восстановления состояния деятельности.

В листинге 9.4 показано, как реализовать функции `onSaveInstanceState()` и `onRestoreInstanceState()`.

Листинг 9.4. Сохранение и восстановление состояния деятельности

```
String my_data = "My data";
float[] my_array = {1.0, 1.2, 1.3};
// Сохранение состояния деятельности
@Override
protected void onSaveInstanceState(Bundle outState) {
super.onSaveInstanceState(outState);
// Сохраняем связанную информацию
outState.putString("data", my_data);
outState.putFloatArray("array", my_array);
}
```

```
// Восстановление состояния деятельности
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
super.onRestoreInstanceState(savedInstanceState);
// Восстанавливаем информацию
my_data = savedInstanceState.getString("data");
my_array = savedInstanceState.getFloatArray("array");
}
```

9.4. Передача данных между деятельностями

Существует несколько способов передачи данных между деятельностями. Первый заключается в использовании публичных членов класса. Выглядит это примерно так:

```
public int flag;
...
MainActivity.flag = 0;
```

В этом способе нет ничего нового, и если вы знаете Java (искренне надеюсь, что это так), то я уверен, вы уже использовали этот способ. Но хочется чего-то, ориентированного на Android.

Со вторым способом — предпочтениями — вы также уже знакомы. Осталось рассмотреть еще два способа: экстра-данные и использование базы данных. Сейчас мы узнаем, как задействовать экстра-данные, а работа с базой данных будет рассмотрена в *главе 13*.

Ранее мы рассмотрели пример, когда одна деятельность вызывает другую:

```
Intent startButton = new Intent(this, StartButton.class);
startActivity(startButton);
```

Представим, что нам нужно передать вызываемой деятельности какие-либо данные. Для этого используется метод `putExtra()`:

```
int flag = 0;
int coins = 100;
...
Intent startButton = new Intent(this, StartButton.class);
startGame.putExtra("flag", flag);
startGame.putExtra("coins", coins);
startActivity(startButton);
```

Первый параметр метода `putExtra()` — это название экстра-переменной, по которому можно обратиться к ней с целью получения ее значения. Второй параметр — это переменная, значение которой передается.

Для получения значения экстра-переменной в коде класса `StartButton`, содержащегося в файле StartButton.java, вызываются методы `getIntExtra()` или `getStringExtra()`

из класса `Intent`, который удобно использовать для получения строковых переменных:

```
// Функция getIntent() возвращает намерение, запустившее эту деятельность
Intent i = getIntent();
// Получаем экстра-переменную с именем flag,
// если таковая не найдена, возвращаем значение по умолчанию — 0
flag = i.getIntExtra("flag", 0);
// Получаем экстра-переменную с именем coins, значение по
// умолчанию не задано
coins = i.getStringExtra("coins");
```

Дополнительные примеры использования метода `getStringExtra()` можно получить по адресу:

https://www.codota.com/code/java/methods/android.content.Intent/getStringExtra.

* * *

В следующей главе мы поговорим о весьма сложных материях: службах, потоках и т. п.

ГЛАВА 10

Потоки, службы и широковещательные приемники

10.1. Потоки

Потоки позволяют выполнять несколько задач одновременно, что дает возможность более эффективно использовать системные ресурсы. Потоки удобно создавать для фонового выполнения какой-либо части программы. Самый простой пример многопоточной программы — это музыкальный проигрыватель. Такая программа запускается и отображает список файлов и кнопки управления воспроизведением. Вы нажимаете кнопку **Play**, и запускается отдельный поток — поток воспроизведения композиции. Но ведь программа должна обрабатывать и нажатия других кнопок — например, кнопок **Stop** и **Pause**. Иначе бы после нажатия кнопки **Play** не было бы возможности остановить, приостановить воспроизведение или переключиться на другую композицию. Благодаря потокам такая возможность у нас есть.

10.1.1. Запуск потока

Представим, что мы создаем тот же музыкальный проигрыватель. В листинге 10.1 приведен обработчик кнопки **Play**, вызывающий функцию `play()` для воспроизведения музыки. Запуск этой функции происходит без создания нового потока.

Листинг 10.1. Версия обработчика кнопки *Play* (без потоков)

```
Button playButton = (Button) findViewById(R.id.play);
playButton.setOnClickListener(new View.OnClickListener() {
       public void onClick(View view){
              play();
       }
});
```

Как видите, ничего сверхъестественного нет — просто вызывается функция `play()`.

Теперь реализуем то же самое, но с запуском функции `play()` в отдельном потоке. Прежде всего нужно создать сам поток:

```
Thread myThread = new Thread(
// здесь будет описание объекта Runnable
);
```

Затем описать объект `Runnable` — это делается в конструкторе потока:

```
new Runnable() {
        public void run() {
                play();
        }
}
```

Как можно видеть, функция `play()` вызывается внутри потока.

Наконец, мы запускаем поток:

```
myThread.start();
```

Полный код создания потока выглядит так:

```
Thread myThread = new Thread(
        new Runnable() {
                public void run() {
                        play();
                }
        }
);
myThread.start();
```

Что будет делать функция `play()`? Все, что нужно сделать в потоке, — т. е. код этой функции зависит от создаваемой программы. В нашем случае будет создан объект класса `MediaPlayer` и вызван метод `start()` для воспроизведения музыки. Примерно так:

```
MediaPlayer media = new MediaPlayer();
media = MediaPlayer.create(this, R.raw.music1);
media.start();
```

«Усыпить» на время поток можно методом `sleep()`:

```
// засыпаем на 1 секунду
myThread.sleep(1000);
```

10.1.2. Установка приоритета потока

Для установки приоритета процесса используется метод `setPriority()`, который нужно вызвать до метода `start()`. Значение приоритета может лежать в диапазоне от `Thread.MIN_PRIORITY (1)` до `Thread.MAX_PRIORITY (10)`:

```
myThread.setPriority(10);
myThread.start();
```

10.1.3. Отмена выполнения потока

Мы должны позаботиться и об останове потока. Использовать метод `stop()` не рекомендуется, поскольку он оставляет приложение в неопределенном состоянии. Правильнее поток `myThread` завершать так (измените идентификаторы на идентификаторы вашего потока):

```
if(myThread != null) {
        Thread dummy = myThread;
        myThread = null;
        dummy.interrupt();
}
```

Существует и другой способ. Он заключается в том, что все запускаемые потоки вы объявляете демонами. В этом случае все запущенные потоки будут автоматически завершены при завершении основного потока приложения. На мой взгляд, это самый удобный вариант:

```
myThread.setDaemon(true);
myThread.start();
```

10.1.4. Обработчики *Runnable*-объектов: класс *Handler*

При сложных вычислениях может понадобиться очередь `Runnable`-объектов. Помещая такой объект в очередь, вы можете задать время его запуска, — например, спустя какое-то количество миллисекунд после помещения в очередь, или же указать точное время запуска объекта.

Для демонстрации обработчика потока мы разработаем программу, запускающую фоновый процесс, который будет каждые 200 мс получать текущее время и обновлять текстовую надпись в главной деятельности приложения. Кроме того, мы организуем кнопку **Start**, которую вы сможете нажимать. По нажатию кнопки **Start** будет выполняться какое-либо действие, в нашем случае — просто изменение надписи на кнопке. Так, после первого нажатия надпись `"Старт"` заменяется числом — количеством нажатий кнопки. В то же время в фоновом режиме наш поток будет обновлять текстовую надпись.

Создайте новый проект. Код разметки приложения приведен в листинге 10.2.

Листинг 10.2. Разметка приложения

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView android:id="@+id/text"
        android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:text=""
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Старт"
        android:id="@+id/start"
        android:layout_gravity="center_horizontal" />

</LinearLayout>
```

Код приложения приведен в листинге 10.3, а само приложение показано на рис. 10.1.

Рис. 10.1. Приложение в действии

Листинг 10.3. Код приложения

```
package com.example.mytimer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.os.Handler;
import android.os.SystemClock;
import android.view.View;
```

```
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private int buttonPressCount = 0;      // Счетчик нажатий кнопки
    TextView buttonLabel;
    private long sTime = 0L;               // Счетчик времени
    private TextView TimeLabel;
    // Обработчик потока: обновляет сведения о времени
    private Handler Handler1 = new Handler();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        if (sTime == 0L) {
            sTime = SystemClock.uptimeMillis();
            Handler1.removeCallbacks(TimeUpdater);
              // Добавляем Runnable-объект TimerUpdater в очередь
              // сообщений, объект будет запущен после задержки в 150 мс.
            Handler1.postDelayed(TimeUpdater, 150);
        }
        TimeLabel = (TextView) findViewById(R.id.text);
        buttonLabel = (TextView) findViewById(R.id.start);
        Button startButton = (Button) findViewById(R.id.start);
        startButton.setOnClickListener(new View.OnClickListener() {
            // Обработчик нажатия кнопки
            public void onClick(View view) {
                buttonLabel.setText(++buttonPressCount);
            }
        });

    }

    private Runnable TimeUpdater = new Runnable() {
        public void run() {
            // Вычисляем время
            final long start = sTime;
            long millis = SystemClock.uptimeMillis() - start;
            int seconds = (int) (millis / 1000);
            int min = seconds / 60;
            seconds = seconds % 60;
              // Выводим время
            TimeLabel.setText("" + min + ":"
                    + String.format("%02d",seconds));
              // Задержка в 300 мс
            Handler1.postDelayed(this, 300);
        } // void run()
    };
```

```
    @Override
    protected void onPause() {
        // Удаляем Runnable-объект
        Handler1.removeCallbacks(TimeUpdater);
        super.onPause();
    }
    @Override
    protected void onResume() {
        super.onResume();
        // Добавляем Runnable-объект
        Handler1.postDelayed(TimeUpdater, 150);
    }

}
```

Кроме метода `postDelayed()` вы можете использовать метод `postAtTime()`:

```
postAtTime(Runnable r, long uptimeMillis)
```

В этом случае объект `r` добавляется в очередь сообщений, запуск объекта производится во время, заданное вторым параметром (в миллисекундах).

Самый простой способ помещения объекта в очередь — метод `post()`, когда указывается только помещаемый объект, но не указывается время выполнения объекта:

```
post(Runnable r)
```

Подробно об обработчиках потока вы можете прочитать в руководстве разработчика:

http://developer.android.com/reference/android/os/Handler.html

10.2. Службы

Служба (сервис, от англ. service) — компонент Android-приложения, запускаемый в фоновом режиме без всякого интерфейса пользователя. Служба может быть запущена или остановлена любым компонентом приложения. Пока служба запущена, любой компонент может воспользоваться предоставляемыми службой возможностями. Приведем несколько примеров использования служб.

Первая деятельность запускает службу, загружающую изображения на веб-сайт. Вторая деятельность подключается к этой службе с целью узнать, сколько файлов уже загружено, чтобы отобразить эту информацию пользователю.

Аналогично можно использовать сервисы при копировании файлов с одной SD-карты на другую — деятельность может подключаться к службе копирования файла, чтобы узнать, сколько байтов скопировано. В случае с музыкальным проигрывателем деятельность может подключаться к сервису, чтобы узнать позицию воспроизводимого трека.

Сервисы представляют собой особую организацию приложения. В отличие от активности (activity) они не требуют наличия визуального интерфейса. Сервисы позволяют выполнять долговременные задачи без вмешательства пользователя.

Все сервисы наследуются от класса `Service` и проходят следующие этапы жизненного цикла:

- метод `onCreate()` — вызывается при создании сервиса;
- метод `onStartCommand()` — вызывается при получении сервисом команды, отправленной с помощью метода `startService()`;
- метод `onBind()` — вызывается при закреплении клиента за сервисом с помощью метода `bindService()`;
- метод `onDestroy()` — вызывается по завершении работы сервиса.

Службы отлично подходят для выполнения постоянных или регулярных операций и для обработки различных событий. Службы запускаются, останавливаются и контролируются разными компонентами приложения, в том числе другими службами, деятельностями (активностями), а также приемниками широковещательных намерений. Если вам нужно выполнять задачи, которые не зависят от взаимодействия с пользователем, то службы — это лучший выбор.

У запущенных служб приоритет выше, чем у невидимых деятельностей, поэтому менее вероятно, что служба будет при распределении ресурсов завершена преждевременно. Даже если такое произойдет, то, в отличие от деятельности, ваша служба автоматически перезапустится, как только окажется достаточно доступных ресурсов.

Если служба взаимодействует с пользователем, то нужно повысить ее приоритет до уровня деятельностей, которые работают на переднем плане. Это гарантирует, что служба завершится только в крайнем случае, но при этом ваше приложение может немного подтормаживать, что испортит от него впечатление. Жизненный цикл службы представлен на рис. 10.2.

Иногда вместо создания собственной службы проще использовать уже имеющиеся системные службы (табл. 10.1). Все зависит от того, какую задачу вам нужно решить.

***Таблица 10.1.** Системные службы*

Служба	Описание
`Account Service`	Управляет пользовательскими учетными записями
`Activity Service`	Управляет деятельностями
`Alarm Service`	Отправляет разовые или периодические оповещения
`Clipboard Service`	Используется для управления буфером обмена
`Connectivity Service`	Управляет сетевыми соединениями
`Download Service`	Управляет загрузками
`Input Method Service`	Управляет текстовым вводом

Таблица 10.1 (окончание)

Служба	Описание
Layout Inflater Service	Управляет компоновкой экрана
Location Service	Занимается отслеживанием координат
NFC Service	Управляет NFC
Notification Service	Управляет уведомлениями
Power Service	Управляет энергопотреблением
Search Service	Управляет поиском
Sensor Service	Используется для доступа к датчикам
Telephony Service	Управляет телефонными функциями
Vibrator Service	Управляет доступом к виброзвонку
Wallpaper Service	Управляет обоями домашнего экрана
WiFi Service	Управляет соединениями Wi-Fi

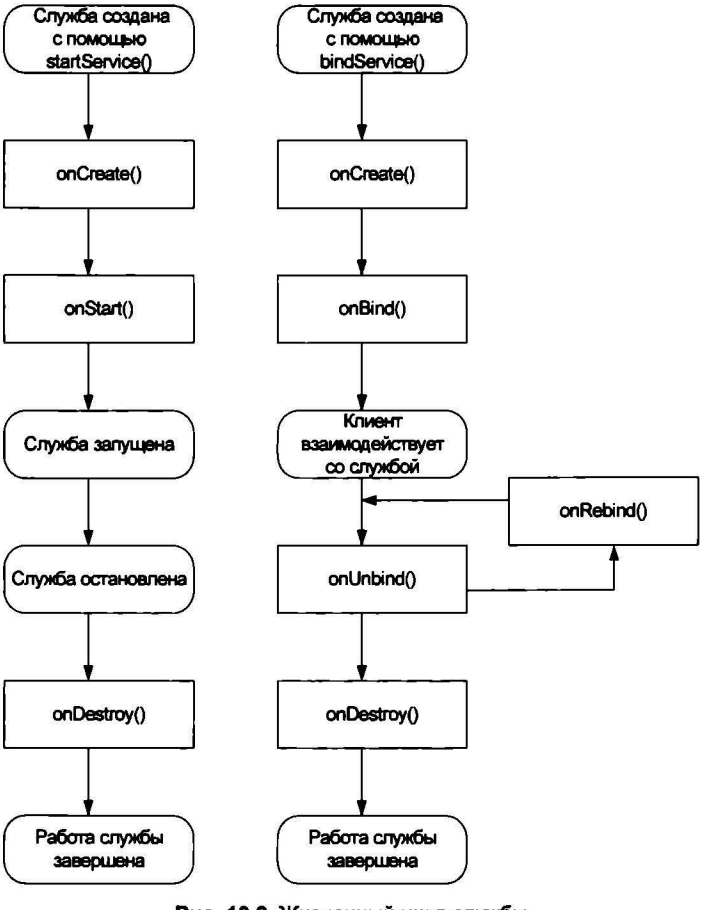


Рис. 10.2. Жизненный цикл службы

Рассмотрим общий алгоритм создания службы. Первым делом нужно создать класс, расширяющий класс `android.app.Service`. В Android Studio для этого нужно щелкнуть правой кнопкой мыши на проекте и выбрать команду **New | Java Class** (рис. 10.3).

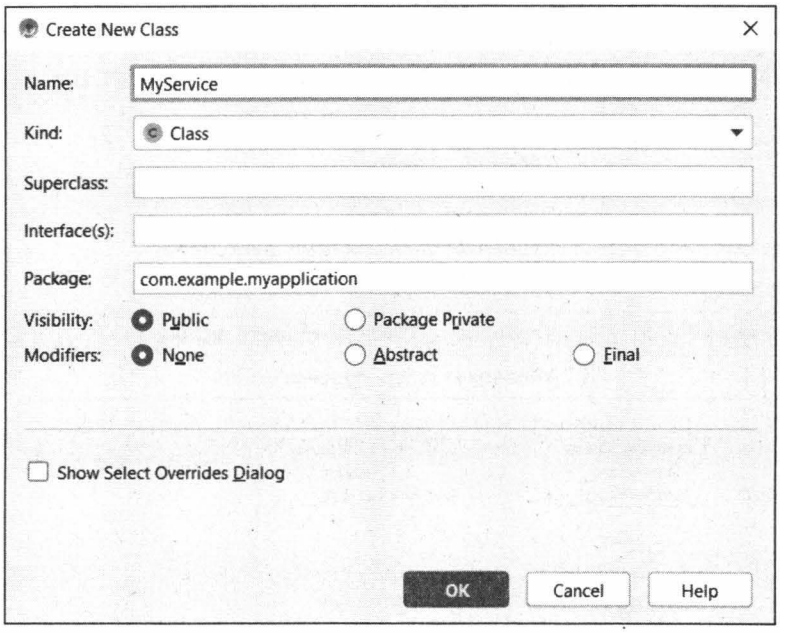


Рис. 10.3. Создание класса службы

По умолчанию будет создана заготовка класса сервиса, представленная в листинге 10.4, *а*.

Листинг 10.4, *а*. Заготовка класса сервиса

```
package com.example.den.myfirstservice;

public class MyService {

}
```

Изменим полученную «болванку», как показано в листинге 10.4, *б*.

Листинг 10.4, *б*. Полноценная заготовка нашего класса

```
package com.example.den.myfirstservice;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
```

```
/**
 * Создано 30.05.2019
 */
public class MyService extends Service  {
    @Override
    public IBinder onBind(Intent arg0) {
        // TODO Auto-generated method stub
        return null;
    }

}
```

В файле манифеста приложения вам нужно описать сервис:

```
<service android:name=".MyService"></service>
```

Элемент `<service>` добавляется в элемент `<application>`. Пример файла манифеста представлен в листинге 10.5.

Листинг 10.5. Пример файла манифеста для приложения с компонентом `Service`

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.den.myfirstservice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <service android:name=".MyService"></service>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Теперь нужно переопределить методы `onCreate()` и `onDestroy()`. Для этого щелкните на имени класса в исходном коде и выберите команду **Code | Override Methods** (рис. 10.4). Конечно, переопределить методы можно и без этого окна, но нужно же продемонстрировать, что таковое в Android Studio имеется.

Рис. 10.4. Окно переопределения методов

Метод `onBind()` нужно переопределять в том случае, если новый компонент привязывается к этой службе после его создания.

Запустите службу функцией `startService()`. Остановить сервис можно функцией `stopService()`. Представим, что у нас есть деятельность `MainActivity` с кнопками **Start** и **Stop**, при этом кнопка **Start** запускает сервис `MyService`, описанный в классе `MyService` (файл **MyService.java**). Обработчик нажатия этой кнопки будет выглядеть так:

```
startButton.setOnClickListener(new View.OnClickListener() {
 public void onClick(View view){
  startService(new Intent(MainActivity.this,
   MyService.class));
 }
});
```

Обработчик кнопки **Stop** будет таким:

```
stopButton.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v){
  stopService(new Intent(MainActivity.this,
   MyService.class));
 }
});
```

В листинге 10.6 приведена расширенная заготовка сервиса. Наш сервис при запуске и останове выводит соответствующее уведомление.

Листинг 10.6. Расширенная заготовка сервиса

```
package com.example.den.myfirstservice;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MyService extends Service  {
    @Override
    public IBinder onBind(Intent arg0) {
        // TODO Auto-generated method stub
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
        Toast.makeText(this,"Служба запущена...",
                Toast.LENGTH_LONG).show();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Служба остановлена...",
                Toast.LENGTH_LONG).show();
    }

}
```

Мы только что создали службу, которая ничего не делает. Пусть она послужит основой для ваших собственных сервисов. А сейчас давайте рассмотрим создание сервиса, который бы воспроизводил MP3-файл.

Создайте новое приложение, поместите в каталог res\raw какой-либо MP3-файл на ваше усмотрение. Пусть он называется music.mp3. Добавьте новый класс и назовите его `MediaService`. Код этого класса показан в листинге 10.7.

Листинг 10.7. Код класса `MediaService`

```
package com.example.den.soundserviceapp;

import android.app.Service;
import android.content.Intent;
```

```
import android.os.IBinder;
import android.media.MediaPlayer;

public class MediaService extends Service {
    MediaPlayer ambientMediaPlayer;
    @Override
    public IBinder onBind(Intent intent) {

        throw new UnsupportedOperationException("Не реализовано");
    }
    @Override
    public void onCreate(){
        ambientMediaPlayer=MediaPlayer.create(this, R.raw.zymotyx);
        ambientMediaPlayer.setLooping(true);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        ambientMediaPlayer.start();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        ambientMediaPlayer.stop();
    }
}
```

Добавьте сервис `MediaService` в файл манифеста:

```
<service
    android:name=".MediaService"
    android:enabled="true"
    android:exported="true" >
</service>
```

Регистрация сервиса производится в узле `application` с помощью добавления элемента `<service>`. В нем определяется атрибут `android:name`, который хранит название класса сервиса. Кроме того, он может принимать еще ряд атрибутов:

- `android:enabled` — если имеет значение `"true"`, то сервис может создаваться системой. Значение по умолчанию — `"true"`;
- `android:exported` — указывает, могут ли другие компоненты приложения обращаться к сервису. Если имеет значение `"true"`, то могут, если имеет значение `"false"`, то нет;
- `android:icon` — значок сервиса, представляет собой ссылку на ресурс `drawable`;
- `android:isolatedProcess` — если имеет значение `"true"`, то сервис может быть запущен как специальный процесс, изолированный от остальной системы;
- `android:label` — название сервиса, которое отображается пользователю;

- android:permission — набор разрешений, которые должно применять приложение для запуска сервиса;
- android:process — название процесса, в котором запущен сервис. Как правило, имеет то же название, что и пакет приложения.

Вернемся к коду службы. Как мы договорились ранее, для воспроизведения музыкального файла сервис будет использовать компонент MediaPlayer.

В сервисе переопределяются все четыре метода жизненного цикла (разве что лишь метод onBind() по сути не имеет никакой реализации):

- в методе onCreate() инициализируется медиапроигрыватель с помощью музыкального ресурса, который добавлен в папку res\raw;
- в методе onStartCommand() начинается воспроизведение. Метод onStartCommand() может возвращать одно из значений, которое предполагает различное поведение в случае, если процесс сервиса был неожиданно завершен системой:
  - START_STICKY — в этом случае сервис снова возвращается в запущенное состояние, как если бы снова был вызван метод onStartCommand() без передачи в этот метод объекта Intent;
  - START_REDELIVER_INTENT — в этом случае сервис снова возвращается в запущенное состояние, как если бы снова был вызван метод onStartCommand() с передачей в этот метод объекта Intent;
  - START_NOT_STICKY — сервис остается в остановленном положении;
- метод onDestroy() завершает воспроизведение.

Для управления сервисом нужно изменить нашу активность. Сначала добавим в разметку две кнопки управления сервисом: первая кнопка будет запускать сервис, вторая — останавливать (листинг 10.8).

Листинг 10.8. Разметка приложения

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <Button
        android:id="@+id/start"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Start"
        android:onClick="click"/>
    <Button
        android:id="@+id/stop"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Stop"
        android:onClick="click"/>
</LinearLayout>
```

Код активности приведен в листинге 10.9.

Листинг 10.9. Код активности

```
package com.example.den.soundserviceapp;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.content.Intent;
import android.view.View;

public class MainActivity extends ActionBarActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void click(View v) {
        Intent i=new Intent(this, MediaService.class);
        if (v.getId()==R.id.start) {
            startService(i);
        }
        else {
            stopService(i);
        }
    }
}
```

Для запуска сервиса используется объект `Intent`:

```
Intent i=new Intent(this, MediaService.class);
```

Для запуска сервиса в классе `Activity` определен метод `startService()`, в который передается объект `Intent`. Этот метод будет посылать команду сервису и вызывать его метод `onStartCommand()`, а также указывать системе, что сервис должен продолжать работать до тех пор, пока не будет вызван метод `stopService()`.

Метод `stopService()` также определен в классе `Activity` и принимает объект `Intent`. Он останавливает работу сервиса, вызывая его метод `onDestroy()`.

10.3. Широковещательные приемники

Широковещательные приемники прослушивают широковещательные сообщения системы. Примеры таких сообщений:

- низкий заряд батареи;
- нажата кнопка камеры;
- установлено новое приложение.

Помимо системных событий пользователь может создавать свои события — например, когда поток завершил вычисления или когда поток начал работу.

Стандартные широковещательные сообщения следующие:

- `ACTION_TIME_TICK` — изменение времени, генерируется каждую минуту;
- `ACTION_TIME_CHANGED` — установлено новое время;
- `ACTION_TIMEZONE_CHANGED` — установлен новый часовой пояс;
- `ACTION_BOOT_COMPLETED` — загрузка выполнена;
- `ACTION_PACKAGE_ADDED` — установлено новое приложение;
- `ACTION_PACKAGE_CHANGED` — приложение (пакет) было изменено, например, включен/выключен какой-то компонент;
- `ACTION_PACKAGE_REMOVED` — пакет (приложение) был удален;
- `ACTION_PACKAGE_RESTARTED` — приложение было перезапущено;
- `ACTION_PACKAGE_DATA_CLEARED` — очищены данные приложения;
- `ACTION_BATTERY_CHANGED` — информация об изменении заряда батареи;
- `ACTION_POWER_CONNECTED` — подключено внешнее питание;
- `ACTION_POWER_DISCONNECTED` — отключено внешнее питание;
- `ACTION_SHUTDOWN` — началось завершение работы.

Дополнительную информацию о событиях можно получить по адресу: **http://developer.android.com/reference/android/content/Intent.html**.

Помимо системных событий пользователь может создавать свои события — например, когда поток завершил вычисления или когда поток начал работу.

Широковещательный приемник — это объект класса `BroadcastReceiver` или одного из его подклассов. Самый главный метод этого класса — `onReceive()`, который вызывается, когда приемник получает сообщение.

Рассмотрим пример запуска службы на основании получения широковещательного сообщения — нажатия кнопки камеры. Приемник станет прослушивать сообщения, а фильтр намерения будет установлен на действие `Intent.ACTION_CAMERA_BUTTON`, которое соответствует нажатию кнопки камеры.

Зарегистрировать приемник можно функцией `registerReceiver()`, а удалить приемник — функцией `unregisterReceiver()`.

Итак, начнем создавать наше приложение. Код основной деятельности `MainActivity` представлен в файле MainActivity.java (листинг 10.10).

Листинг 10.10. Код основной деятельности (файл MainActivity.java)

```
package com.samples.my_receiver;

import android.app.Activity;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;

public class MainActivity extends Activity {
       // Создаем объект iReceiver класса MyReceiver
       MyReceiver iReceiver = new MyReceiver();
       /** Вызывается, когда деятельность запускается впервые. */
       @Override
       public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.main);
              // Создаем фильтр намерения
              IntentFilter iFilter = new
                     IntentFilter(Intent.ACTION_CAMERA_BUTTON);
              // Добавляем действие в фильтр
              iFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
              // Регистрируем приемник
       registerReceiver(iReceiver, iFilter);
       }
       @Override
       protected void onDestroy() {
       // Уничтожаем приемник по завершении приложения
              unregisterReceiver(iReceiver);
              super.onDestroy();
       }
}
```

Разберемся, что здесь есть что. Первым делом мы объявили объект `iReceiver` класса `MyReceiver` (этот класс будет описан позже). Затем создали фильтр намерения:

```
IntentFilter iFilter = new IntentFilter(Intent.ACTION_CAMERA_BUTTON);
iFilter.addAction(Intent.ACTION_PACKAGE_ADDED);
```

После этого нужно зарегистрировать приемник с помощью `registerReceiver()`:

```
registerReceiver(iReceiver, iFilter);
```

Первый параметр — это сам приемник, второй — фильтр намерений.

Вот теперь надо создать класс `MyReceiver`. Как это сделать с помощью Android Studio, вы уже знаете, поэтому привожу сразу код класса (листинг 10.11).

Листинг 10.11. Код класса `MyReceiver` (файл MyReceiver.java)

```
package com.samples.my_receiver;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

// Класс MyReceiver является расширением класса BroadcastReceiver
public class MyReceiver extends BroadcastReceiver {
       // Переопределяем метод onReceive()
       @Override
       public void onReceive(Context rcvContext, Intent rcvIntent) {
              String action = rcvIntent.getAction();
              // Если действие = Intent.ACTION_CAMERA_BUTTON
              if (action.equals(Intent.ACTION_CAMERA_BUTTON)) {
                     // то запускаем сервис MyService
                     rcvContext.startService(new Intent(rcvContext,
                     MyService.class));
              }
       }
}
```

Приемник анализирует полученное действие. Если нажата кнопка камеры, то будет запущен сервис (служба) `MyService`, описанный в файле MyService.java. Код этого файла (расширенная заготовка сервиса) был представлен в листинге 10.6. Вам нужно изменить только первую строчку — она должна выглядеть так:

```
package com.samples.my_receiver;
```

* * *

В следующей главе мы подробно рассмотрим взаимодействие с аппаратными средствами смартфона/планшета.

ГЛАВА 11

Аппаратные средства смартфона/планшета

Современный смартфон — это настоящий компьютер и даже немного больше, поскольку смартфоны оснащены всевозможными аппаратными средствами: камерой, микрофоном, приемниками ГЛОНАСС/GPS, различными датчиками: акселерометром, датчиком света, датчиком температуры (попросту говоря — термометром), для полного комплекта не хватает только датчика дождя. В этой главе мы поговорим о том, как работать с аппаратными средствами смартфона. И начнем с самого интересного — с датчиков.

11.1. Датчики смартфона

В вашем смартфоне имеется множество самых разных датчиков. Количество датчиков и их типы различны в зависимости от модели устройства. Камера — это тоже датчик. В некоторых телефонах может быть несколько камер, в некоторых — только одна. Однако камера заслуживает отдельного разговора, поэтому сейчас мы поговорим о более традиционных датчиках, а именно:

- об акселерометрах (`TYPE_ACCELEROMETER`), позволяющих определить ускорение мобильного устройства по осям *X*, *Y*, *Z*;
- о датчиках гравитации (`TYPE_GRAVITY`), позволяющих определить направление силы тяжести;
- о датчиках чистого линейного ускорения без учета гравитации (`TYPE_LINEAR_ACCELERATION`). Это виртуальные датчики, использующие показания акселерометра;
- о гироскопах (`TYPE_GYROSCOPE`), возвращающих текущее положение устройства в пространстве в градусах по трем осям;
- о датчиках приближенности (`TYPE_PROXIMITY`), замеряющих расстояние в сантиметрах между устройством и целевым объектом. Каким образом выбирается объект и какие расстояния поддерживаются, зависит от аппаратной реализации этих датчиков, возможно просто возвращение двух значений: близко или далеко. Типичное применение таких датчиков — фиксация расстояния между устройст-

вом и ухом пользователя для автоматического регулирования яркости экрана или выполнения голосовой команды;

- о датчиках магнитного поля (TYPE_MAGNETIC_FIELD), возвращающих текущие показатели магнитного поля в микротеслах по трем осям;
- о датчиках света (TYPE_LIGHT), используемых многими программами, показывающими пользователю важное визуальное изображение. Например, программами навигации датчик определяет, что вокруг стало темно (наступила ночь, или вы заехали в тоннель), и переключает карту в ночной режим, где используются темные цвета, чтобы экран не превратился в «белое пятно» и не отвлекал от дороги;
- о температурных датчиках — термометрах (TYPE_AMBIENT_TEMPERATURE), возвращающих значения в градусах Цельсия. Они заменили сейчас устаревший датчик TYPE_TEMPERATURE;
- о датчиках атмосферного давления — барометрах (TYPE_PRESSURE);
- о датчиках ориентации (TYPE_ORIENTATION), замеряющих повороты, наклоны и вращение устройства;
- о датчиках относительной влажности — гигрометрах (TYPE_RELATIVE_HUMIDITY), возвращающих значения влажности в процентах;
- о датчиках, подсчитывающих количество шагов, — шагомерах (TYPE_STEP_COUNTER).

Значения, возвращаемые упомянутыми здесь датчиками, приведены в табл. 11.1.

Таблица 11.1. Значения, возвращаемые датчиками

Тип датчика	Кол-во значений	Содержание значений	Примечание
TYPE_ACCELEROMETER	3	value[0]: ось X (поперечная) value[1]: ось Y (продольная) value[2]: ось Z (вертикальная)	Ускорение (м/с^2) по трем осям. Константы SensorManager.GRAVITY_*
TYPE_GRAVITY	3	value[0]: ось X (поперечная) value[1]: ось Y (продольная) value[2]: ось Z (вертикальная)	Сила тяжести (м/с^2) по трем осям. Константы SensorManager.GRAVITY_*
TYPE_RELATIVE_HUMIDITY	1	value[0]: относительная влажность	Относительная влажность в процентах (%)
TYPE_LINEAR_ACCELERATION	3	value[0]: ось X (поперечная) value[1]: ось Y (продольная) value[2]: ось Z (вертикальная)	Линейное ускорение (м/с^2) по трем осям без учета силы тяжести

Таблица 11.1 (окончание)

Тип датчика	Кол-во значений	Содержание значений	Примечание
TYPE_GYROSCOPE	3	value[0]: ось X value[1]: ось Y value[2]: ось Z	Скорость вращения (рад/с) по трем осям
TYPE_ROTATION_VECTOR	3 (+ одно необяз.)	values[0]: x*sin(q/2) values[1]: y*sin(q/2) values[2]: z*sin(q/2) values[3]: cos(q/2) (необяз.)	Положение устройства в пространстве. Описывается в виде угла поворота относительно оси в градусах
TYPE_MAGNETIC_FIELD	3	value[0]: ось X (поперечная) value[1]: ось Y (продольная) value[2]: ось Z (вертикальная)	Внешнее магнитное поле (мкТл)
TYPE_LIGHT	1	value[0]: освещенность	Внешняя освещенность (лк). Константы SensorManager.LIGHT_*
TYPE_PRESSURE	1	value[0]: атм.давление	Атмосферное давление (мбар)
TYPE_PROXIMITY	1	value[0]: расстояние	Расстояние до цели
TYPE_AMBIENT_TEMPERATURE	1	value[0]: температура	Температура воздуха в градусах по Цельсию

Здесь приведен далеко не полный список датчиков. Получить список датчиков можно методом `getSensorList()` класса `SensorManager`:

```
SensorManager sensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);

List<Sensor> listSensor = sensorManager.getSensorList(Sensor.TYPE_ALL);

List<String> listSensorType = new ArrayList<>();
for (int i = 0; i < listSensor.size(); i++) {
    listSensorType.add(listSensor.get(i).getName());
}

setListAdapter(new ArrayAdapter<>(this,
        android.R.layout.simple_list_item_1, listSensorType));
        getListView().setTextFilterEnabled(true);
```

Примечание

Дополнительная информация о датчиках доступна по адресу: http://developer.android.com/reference/android/hardware/SensorManager.html.

Для чтения датчиков нужно подключить следующие пакеты:

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

Когда эти пакеты подключены, надо определить объекты класса `SensorManager`:

```
private SensorManager sManager = null;
sManager = (SensorManager)getSystemService(SENSOR_SERVICE);

sManager.registerListener(tempSensorListener,
sManager.getDefaultSensor(Sensor.TYPE_TEMPERATURE),
SensorManager.SENSOR_DELAY_GAME);
...
```

Как можно видеть, здесь рассматривается датчик температуры. Методу `registerListener()` этого датчика передаются три параметра:

- первый — название обработчика датчика температуры. В нашем случае — это `tempListener`, который будет определен чуть далее;
- второй параметр — датчик по умолчанию, в нашем случае — датчик температуры;
- третий параметр задает время обновления показаний датчика. Для наиболее быстрого обновления используйте `SENSOR_DELAY_GAME`, для обычного обновления — `SENSOR_DELAY_NORMAL`.

Слушателя `tempListener` определяет следующий код. В нем мы переопределяем методы `onAccuracyChanged()` и `onSensorChanged()`. Причем, первый метод нам не нужен, поэтому мы определим его как пустой. А второй метод будет устанавливать текст области `info` (элемент `TextView` в разметке) — в ней мы будем показывать температуру.

```
private final SensorEventListener tempListener = new SensorEventListener(){

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {}

@Override
public void onSensorChanged(SensorEvent event) {
if(event.sensor.getType()==Sensor.TYPE_TEMPERATURE){
  info.setText("Temperature: "+event.values[0]);
        }
       }
};
```

11.2. Работаем с камерой

Камера, как уже отмечалось ранее, это еще один тип датчика. Однако с камерой работать несколько сложнее, чем с другими датчиками. Есть два способа доступа к камере: первый заключается в вызове стандартного интерфейса управления камерой, а второй — в использовании класса `Camera`. Все зависит от того, что вы хотите сделать. Если просто отобразить интерфейс управления камерой, чтобы пользователь мог создать фотографию (потом он может выбрать файл с только что созданным изображением и что-то с ним сотворить), тогда вам предпочтителен первый способ. Второй способ подойдет, если вы сразу хотите получить снимок, созданный камерой. Понятно, что второй способ более удобен для пользователя, но более сложен для реализации программистом.

Начнем с первого способа. Для отображения стандартного интерфейса управления камерой служит следующий код:

```
Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
startActivity(intent);
```

Просто, но неудобно. Будет отображен интерфейс управления камерой, и пользователь, конечно, сможет сделать снимок, но в программе должна быть предусмотрена возможность выбора этого снимка. Для этого в нее, например, можно ввести две кнопки: одну — для вызова интерфейса управления камерой, а вторую — для выбора файла. В этом и неудобство, т. к. сначала нужно будет сделать фотографию, а потом указать путь к ней.

Поэтому рассмотрим второй способ. Чтобы использовать класс `Camera`, нужно добавить в файл манифеста соответствующее разрешение:

```
<uses-permission android:name="android.permission.CAMERA" />
```

Для работы с камерой нам понадобятся следующие классы:

- `Camera` — доступ к «железу» (непосредственно к самой камере);
- `Camera.Parameters` — позволяет указать параметры камеры, например размер изображения, его качество и т. п.;
- `SurfaceView` — позволяет выделить поверхность, которая будет использоваться в качестве области предварительного просмотра для камеры.

Создайте новый проект `MyCamera` (имя пакета com.samples.mycamera). Основной файл разметки res\layout\main.xml для нашего проекта приведен в листинге 11.1. Элемент `SurfaceView` будет использоваться в качестве области предварительного просмотра камеры.

Листинг 11.1. Файл разметки main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical">
```

```
<SurfaceView android:id="@+id/surface"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
</SurfaceView>
</LinearLayout>
```

Теперь нужно создать интерфейс управления камерой. Его мы опишем в файле camera_int.xml (листинг 11.2). Этот файл нужно поместить в каталог res\layout.

Листинг 11.2. Файл camera_int.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical"
        android:gravity="bottom"
        android:layout_gravity="bottom">
<LinearLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_horizontal">
<Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Сфотографировать"
/>
</LinearLayout>
</LinearLayout>
```

Для получения фотографии задействуется метод `takePicture()`. Он требует определения трех обратных вызовов (callbacks):

- `SurfaceCallback()` — служит для управления поверхностью, может также использоваться для применения к полученному фото различных графических эффектов;
- `PictureCallback()` — используется для управления памятью (что делать с фото, если не хватило аппаратной памяти);
- `CompressionCallback()` — используется для сжатия фото.

Для управления поверхностью (surface) служит интерфейс `SurfaceHolder.Callback`. Нам нужно переопределить три его метода:

- `surfaceCreated()` — вызывается во время создания поверхности, используется для инициализации объектов;

- `surfaceChanged` — вызывается после создания поверхности и когда изменены параметры поверхности, например ее размер;
- `surfaceDestroyed()` — вызывается при удалении поверхности, используется для очистки памяти.

Полный код приложения с комментариями приведен в листинге 11.3.

Листинг 11.3. Полный код приложения

```
package com.samples.mycamera;

import android.app.Activity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.PixelFormat;
import android.os.Bundle;
import android.provider.MediaStore.Images;
import android.view.LayoutInflater;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.View.OnClickListener;
import android.view.ViewGroup.LayoutParams;
import android.widget.Button;
import android.widget.Toast;
// Пакеты, необходимые для работы с камерой
import android.hardware.Camera;
import android.hardware.Camera.PictureCallback;
import android.hardware.Camera.ShutterCallback;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MyCamera extends Activity implements SurfaceHolder.Callback {
   private LayoutInflater mInflater = null;
   Camera Cam1;                    // Наша камера
   byte[] tempdata;                // Массив для временных данных
   boolean mPreviewRunning = false;
   // Владелец поверхности (SurfaceHolder)
   private SurfaceHolder PreviewHolder;
   private SurfaceView Preview;
   // Кнопка для получения фото
   Button GetPicture;

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       getWindow().setFormat(PixelFormat.TRANSLUCENT);
```

```
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
        WindowManager.LayoutParams.FLAG_FULLSCREEN);
        setContentView(R.layout.main);

        // Поиск и установка SurfaceHolder
        Preview = (SurfaceView)findViewById(R.id.surface);
        PreviewHolder = Preview.getHolder();
        PreviewHolder.addCallback(this);
        PreviewHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        mInflater = LayoutInflater.from(this);
        View overView = mInflater.inflate(R.layout.cameraoverlay, null);
        this.addContentView(overView,
        new LayoutParams(LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT));

        // Поиск кнопки
        GetPicture = (Button) findViewById(R.id.button);
        // Устанавливаем обработчик нажатия кнопки
        GetPicture.setOnClickListener(new OnClickListener(){
        public void onClick(View view){
        // Вызываем метод takePicture() для получения картинки
        Cam1.takePicture(Shutter, PicCallback, compress);
        }
    });
}

        // Пустая заглушка - ничего не делаем
        ShutterCallback Shutter = new ShutterCallback(){
        @Override
        public void onShutter() {}
        };

        // Пустая заглушка - ничего не делаем
        PictureCallback PicCallback = new PictureCallback() {
        public void onPictureTaken(byte[] data, Camera c) {}
        };

        // Заполняем массив с временными данными и вызываем
        // функцию done()
        PictureCallback compress = new PictureCallback() {
        public void onPictureTaken(byte[] data, Camera c) {
        if(data !=null) {
                tempdata=data;
                done();
        }
        }
};
```

```
void done() {
        // Получаем растровое изображение путем декодирования
        // массива tempdata
        Bitmap bm = BitmapFactory.decodeByteArray(tempdata,
        0, tempdata.length);
        String url = Images.Media.insertImage(getContentResolver(),
                bm, null, null);
        bm.recycle();
        Bundle bundle = new Bundle();
        if(url!=null) {
                bundle.putString("url", url);
                Intent mIntent = new Intent();
                mIntent.putExtras(bundle);
                setResult(RESULT_OK, mIntent);
        } else {
                Toast.makeText(this, "Ошибка!",
                Toast.LENGTH_SHORT).show();
        }
        finish();
}

// Реакция на изменение поверхности
@Override
public void surfaceChanged(SurfaceHolder holder, int format,
int w, int h) {
        try {
                if (mPreviewRunning) {
                        Cam1.stopPreview();
                        // Останавливаем предварительный просмотр
                        mPreviewRunning = false;
                }
        // Получаем параметры камеры
        Camera.Parameters p = Cam1.getParameters();
        // Устанавливаем размер предварительного просмотра
        p.setPreviewSize(w, h);
        // Устанавливаем параметры камеры
        Cam1.setParameters(p);
        // Устанавливаем владельца поверхности
        Cam1.setPreviewDisplay(holder);
        // Запускаем предварительный просмотр
        Cam1.startPreview();
        // Флаг предварительного просмотра
        mPreviewRunning = true;
        } catch(Exception e) {
                // Действие в случае исключения, здесь нужно вывести или
                // сообщение об ошибке или проанализировать ошибку. Для простоты
                // этого примера мы ничего не будем делать
        }
}
```

```
@Override
public void surfaceCreated(SurfaceHolder holder) {
       // Действие при создании поверхности - открываем камеру
       Cam1 = Camera.open();
}
@Override
public void surfaceDestroyed(SurfaceHolder holder) {
       // Останавливаем предварительный просмотр
       Cam1.stopPreview();
       // Сбрасываем флаг
       mPreviewRunning = false;
       // Освобождаем ресурсы
       Cam1.release();
       Cam1=null;
   }
}
```

11.3. Работаем с Bluetooth

Bluetooth — это очень популярная беспроводная технология, которую можно использовать как для подключения дополнительных устройств (например, телефонной гарнитуры hands free), так и для передачи файлов. Подробного экскурса в историю и технические характеристики Bluetooth в этой книге не будет, благо в Интернете этой информации предостаточно. Мы же поговорим о программировании Bluetooth в Android.

Для осуществления передачи данных по Bluetooth нужно включить адаптер Bluetooth, найти доступные устройства (другие смартфоны, ноутбуки, планшеты или стационарные компьютеры с включенным Bluetooth-адаптером), подключиться к одному из этих устройств и произвести обмен данными.

Чтобы использовать Bluetooth-адаптер, нужно в файл манифеста добавить следующие строки:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

В пакете `android.bluetooth` определены следующие классы:

- `BluetoothAdapter` — представляет интерфейс обнаружения и установки Bluetooth-соединений;
- `BluetoothClass` — описывает общие характеристики Bluetooth-устройства;
- `BluetoothDevice` — представляет удаленное Bluetooth-устройство;
- `BluetoothSocket` — сокет или точка соединения для данных, которыми наша система обменивается с другим Bluetooth-устройством;
- `BluetoothServerSocket` — сокет для прослушивания входящих Bluetooth-соединений.

Итак, приступим к реализации первого этапа обмена данными по Bluetooth, а именно — к включению Bluetooth-адаптера. Прежде всего нужно получить адаптер по умолчанию:

```
BluetoothAdapter Bluetooth1 = BluetoothAdapter.getDefaultAdapter();
```

Активировать Bluetooth-адаптер можно с помощью следующего кода:

```
// Если Bluetooth-адаптер выключен
if(!Bluetooth1.isEnabled()) {
// Создаем действие ACTION_REQUEST_ENABLE — запрашивает включение
// адаптера
Intent eIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
// Выполняем действие
startActivity(eIntent);
}
```

Затем нужно обнаружить Bluetooth-устройства, расположенные по соседству. Для упрощения кода найденные устройства мы будем записывать в журнал с помощью параметра `Log` (листинг 11.4). Журнал можно будет просмотреть в Android Studio при запуске приложения. В реальном приложении вы должны будете отобразить найденные устройства в виде списка и предоставить пользователю выбор устройства.

Листинг 11.4. Поиск Bluetooth-устройств

```
import android.util.Log;
...
private final BroadcastReceiver myReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
// Когда найдено устройство
if (BluetoothDevice.ACTION_FOUND.equals(action)) {
       // Получаем объект BluetoothDevice из Intent
       BluetoothDevice device = intent.getParcelableExtra(
       BluetoothDevice.EXTRA_DEVICE);
       // Выводим сообщение в журнал
       Log.v("BlueTooth found: ",device.getName() + "\n"
       + device.getAddress());
       }
   }
};
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(myReceiver, filter);
Bluetooth1.startDiscovery();
```

Затем нужно подключиться к устройству, которое выбрал пользователь. Надо отметить, что вы можете разработать как серверное, так и клиентское приложение. Сер-

верное приложение будет ожидать, что к нему подключится клиент, а клиентское станет подключаться к серверу. Код серверного приложения приведен в листинге 11.5.

Листинг 11.5. Код серверного приложения

```
// Класс AcceptBluetoothThread принимает входящие запросы
private class AcceptBluetoothThread extends Thread {

  private final BluetoothServerSocket myServerSocket;

  public AcceptThread() {
       // Используем временный объект
       BluetoothServerSocket tmp = null;
       try {
       // MY_UUID — идентификатор, также используемый клиентом
       tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME,MY_UUID);
       } catch (IOException e) { }
       // Присваиваем tmp члену класса myServerSocket
       myServerSocket = tmp;
        }
  public void run() {
       BluetoothSocket socket = null;
       // Прослушиваем соединения
       while (true) {
       try {  // Принимаем соединение
              socket = myServerSocket.accept();
       } catch (IOException e) {
       break;
       }
       // Если соединение было принято
       if (socket != null) {
       // Производим обработку соединения — в отдельном потоке
       DoSomethingWith(socket);
       // После обработки соединения закрываем сокет
       myServerSocket.close();
       break;
       }
   }
}
// Действие в случае отмены соединения
  public void cancel() {
try {  // Закрываем сокет
       myServerSocket.close();
} catch (IOException e) { }
  }
}
```

Код клиентского приложения приведен в листинге 11.6.

Листинг 11.6. Клиентское приложение, устанавливающее соединение с сервером

```
private class ConnectThread extends Thread {
  private final BluetoothSocket mySocket;
  private final BluetoothDevice myDevice;

  public ConnectThread(BluetoothDevice device) {
    // Используем временный объект
    BluetoothSocket tmp = null;
    myDevice = device;
    // Получаем BluetoothSocket для соединения с BluetoothDevice
    try {
      // MY_UUID — идентификатор, такой же использует сервер
      tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e) { }
    mySocket = tmp;
    }

    public void run() {
    // Отключаем обнаружение устройств, поскольку оно замедляет
    // соединение
    mAdapter.cancelDiscovery();
    try {
       // Соединяемся с устройством через сокет
       mySocket.connect();
    } catch (IOException connectException) {
       // Невозможно подключиться, закрываем сокет
       try {
              mySocket.close();
       } catch (IOException closeException) { }
       return;
      }
  // Соединение установлено, производим его обработку в
  // отдельном потоке. Далее можно сделать все, что угодно, например,
  // передать данные в Bluetooth-сокет, как в обычный сокет
  DoSomethingWith(mySocket);
}
// Отмена соединения, закрываем сокет
  public void cancel() {
       try {
              mySocket.close();
       } catch (IOException e) { }
   }
}
```

11.4. Виброзвонок

Для привлечения внимания к уведомлению или диалоговому окну программы можно использовать вибрацию. Для управления виброзвонком надо добавить в файл манифеста следующее разрешение:

```
<uses-permission android:name="android.permission.VIBRATE" />
```

Затем следует задействовать класс `Vibrator`:

```
Vibrator Vib = (Vibrator) getSystemService(Context.VIBRATOR_SERVICE);
Vib.vibrate(2000); // Вибрировать 2 секунды
```

Для принудительной отмены вибрации (например, вы установили вибрацию две секунды, а пользователь отреагировал раньше) используется метод `cancel()`:

```
Vib.cancel();
```

11.5. Набор номера

Прежде чем мы рассмотрим более сложный пример с определением номера входящего звонка, давайте разберемся, как выполнить набор номера.

Прежде всего в файл манифеста нужно добавить разрешение на исходящий звонок:

```
<uses-permission android:name="android.permission.CALL_PHONE" />
```

Далее вы можете использовать одну из двух операций: или `ACTION_CALL`, или `ACTION_DIAL`. Первая операция отобразит диалоговое окно с набираемым номером (как обычно бывает при наборе номера вручную), вторая операция наберет номер без показа какого-либо интерфейса пользователя.

Вот пример использования этих двух операций:

```
startActivity(new Intent(Intent.ACTION_CALL, Uri.parse("tel:11122233344")));
startActivity(new Intent(Intent.ACTION_DIAL, Uri.parse("tel: 11122233344")));
```

11.6. Определение номера входящего звонка

Иногда нужно выполнить какие-либо действия только при определенном состоянии смартфона — например, при входящем звонке или при завершении звонка. В этом случае вам требуется прослушивать состояние устройства — как только будет обнаружен входящий (исходящий) звонок, вам надо будет произвести какие-нибудь действия, — например, начать запись разговора.

Для прослушивания состояния смартфона с целью ожидания какого-то события используются так называемые *прослушки*.

В этой книге мы не станем рассматривать все имеющиеся прослушки, а разберемся только с самой часто используемой — `PhoneStateListener.LISTEN_CALL_STATE`, позволяющей определить номер входящего звонка (и, соответственно, выполнить определенные действия при входящем звонке). Возможны три состояния звонка:

- CALL_STATE_IDLE — устройство не используется для телефонного звонка (не принимается входящий звонок и не устанавливается исходящий);
- CALL_STATE_RINGING — устройство принимает входящий звонок;
- CALL_STATE_OFFHOOK — пользователь говорит по телефону.

Сейчас мы напишем программу, которая реагирует на все три состояния звонка и ...ничего не делает (действия вы сможете определить сами). Такое решение я принял, чтобы не захламлять код. А что делать, решите вы сами — можно, например, при получении звонка вывести соответствующее уведомление. Чтобы вы могли определить собственные действия при изменении состояния звонка, мы переопределим метод onCallStateChanged() (листинг 11.7).

Листинг 11.7. Реакция на входящий звонок

```
...
import android.telephony.PhoneStateListener;
import android.telephony.TelephonyManager;
...
public class HardwareTelephony extends Activity {
TextView info;                // Сюда можно выводить информацию о звонке
TelephonyManager tm;

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);

log =(TextView) findViewById(R.id.log);
// Создаем объект класса TelephonyManager
tm = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
// Устанавливаем прослушку для LISTEN_CALL_STATE
tm.listen(new TelListener(),PhoneStateListener.LISTEN_CALL_STATE);
}

private class TelListener extends PhoneStateListener {
public void onCallStateChanged(int state, String incomingNumber) {
super.onCallStateChanged(state, incomingNumber);
switch (state) {
case TelephonyManager.CALL_STATE_IDLE:
  log.setText("IDLE");
                break;
case TelephonyManager.CALL_STATE_OFFHOOK:
  log.SetText("OFFHOOK, Входящий звонок:" +incomingNumber);
                break;
case TelephonyManager.CALL_STATE_RINGING:
                log.SetText("RINGING, Входящий звонок:" +incomingNumber);
                break;
```

```
            default:
        break;
  } // switch
    } // onCallStateChanged
  }
}
```

Наше приложение выводит в текстовую область (TextView) с именем log состояние телефона и номер входящего звонка, если таковой имеется. Чтобы приложение работало корректно, в файл манифеста нужно добавить строку:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

11.7. Получение информации о смартфоне

С помощью класса TelephonyManager можно получить информацию о смартфоне, определить его состояние и набрать номер. Прежде чем приступить к написанию кода, нужно добавить в файл манифеста следующую строку:

```
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
```

Далее создайте в файле разметки текстовое представление с именем info — в него мы будем выводить информацию о состоянии:

```
<TextView
    android:id="@+id/info"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
```

Подготовительный код:

```
import android.telephony.TelephonyManager;
...
       TextView info;              // Текстовая область TextView
       TelephonyManager tm;        // Для информации об устройстве
...
String EOL = "\n";

// Находим текстовую область в разметке
info =(TextView) findViewById(R.id.info);
// Создаем объект tm для получения информации о телефоне
tm = (TelephonyManager)getSystemService(TELEPHONY_SERVICE);
// буфер строк
StringBuilder buffer = new StringBuilder();

Теперь выводим в буфер строк всю информацию о телефоне:

// Общая информация об устройстве
buffer.append("General info:\n\n");
```

```
buffer.append("Device ID :")
.append(tm.getDeviceId()).append(EOL);
buffer.append("Software version: ")
.append(tm.getDeviceSoftwareVersion()).append(EOL);
buffer.append("Number: ")
.append(tm.getLine1Number()).append(EOL);

// Информация об операторе, выдавшем SIM-карту
buffer.append("\nOperator:\n\n");
buffer.append("Country Code (ISO): ")
.append(tm.getSimCountryIso()).append(EOL);
buffer.append("Provider: ")
.append(tm.getSimOperator()).append(EOL);
buffer.append("IMEI: ")
.append(tm.getImei()).append(EOL);
buffer.append("Name: ")
.append(tm.getSimOperatorName()).append(EOL);
buffer.append("Serial number of SIM: ")
.append(tm.getSimSerialNumber()).append(EOL);

// Информация о текущей сети
buffer.append("\nNrtwork:\n\n");
buffer.append("Country Code (ISO): ")
.append(tm.getNetworkCountryIso()).append(EOL);
buffer.append("Operator: ")
.append(tm.getNetworkOperator()).append(EOL);
buffer.append("Name: ")
.append(tm.getNetworkOperatorName()).append(EOL);
```

Теперь осталось поместить буфер строк в область `info`:

```
info.setText(sb.toString());
```

11.8. Ориентация экрана

Любое Android-устройство, оснащенное акселерометром, может определить, в каком положении оно сейчас находится. При этом в зависимости от показаний акселерометра может изменяться ориентация экрана: альбомная или портретная (книжная). Но не всегда это хорошо. Одно дело, если вы разрабатываете офисное приложение, но совсем другое, когда разрабатывается игра. В этом случае изменение ориентации экрана может оказаться не слишком желательным. Но вы можете принудительно задать ориентацию экрана для каждой деятельности. Для этого в файле манифеста в элемент `activity` следует добавить параметр `orientation`:

```
android:screenOrientation="portrait"
android:screenOrientation="landscape"
```

Значение `portrait` означает портретную (книжную) ориентацию, значение `landscape` — альбомную.

Скрыть клавиатуру можно путем добавления следующего параметра:

```
android:configChanges="orientation|keyboardHidden"
```

Иногда нужно знать, когда скрыта клавиатура или когда изменена ориентация экрана. Тогда вам следует переопределить метод `onConfigurationChanged()` (листинг 11.8).

Листинг 11.8. Переопределение метода `nConfigurationChanged`

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

    // Проверяем ориентацию экрана
    if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(this, "landscape", Toast.LENGTH_SHORT).show();
    } else
    if (newConfig.orientation == Configuration.ORIENTATION_PORTRAIT){
        Toast.makeText(this, "portrait", Toast.LENGTH_SHORT).show();
    }
    // Проверяем видимость клавиатуры
if (newConfig.hardKeyboardHidden == Configuration.HARDKEYBOARDHIDDEN_NO) {
       Toast.makeText(this, "keyboard visible",
       Toast.LENGTH_SHORT).show();
    } else
          if (newConfig.hardKeyboardHidden ==
              Configuration.HARDKEYBOARDHIDDEN_YES) {
                    Toast.makeText(this, "keyboard hidden",
                     Toast.LENGTH_SHORT).show();
    }
}
```

* * *

На этом тема о датчиках исчерпана, и в следующей главе мы поговорим об отправке SMS и выходе в Интернет.

ГЛАВА 12

Соединение с внешним миром

12.1. Отправка SMS

Для отправки SMS используется класс `SmsManager`, находящийся в пакете `android.telephony`.

Отправить SMS совсем несложно. Прежде всего нужно добавить в файл манифеста соответствующее разрешение:

```
<uses-permission android:name="android.permission.SEND_SMS">
```

Затем надо определить объект класса `SmsManager` — для этого используется статический метод `getDefault()`. Далее следует определить получателя сообщения (его номер телефона) и текст самого сообщения:

```
import android.telephony.SmsManager;
import android.telephony.SmsMessage;
...
SmsManager sendSMS = SmsManager.getDefault();
String num = "номер получателя";
String msg = "My first SMS";
sendSMS.sendTextMessage(num, null, msg, null, null);
```

Отправляет SMS метод `sendTextMessage()`. Обычно достаточно указать первый и третий его параметры. Первый параметр задает номер телефона получателя, третий — текст сообщения. Второй параметр позволяет указать номер SMS-центра. Если указано значение `null`, будет использован номер SMS-центра, заданный в настройках устройства. Четвертый и пятый параметры служат для отслеживания факта отправки и доставки сообщения соответственно.

Разберемся, как использовать четвертый и пятый параметры:

```
// Определяем флаги отправки и доставки SMS
String SENT_SMS_FLAG = "SENT_SMS";
String DELIVER_SMS_FLAG = "DELIVER_SMS";
```

```
// Создаем соответствующие действия

// Действие, связанное с отправкой SMS
Intent sent_sms = new Intent(SENT_SMS_FLAG);
// Отложенная деятельность, связанная с sent_sms
PendingIntent spin = PendingIntent.getBroadcast(this,0,sent_sms,0);

// Аналогично для доставки:
Intent deliver_sms = new Intent(DELIVER_SMS_FLAG);
PendingIntent dpin = PendingIntent.getBroadcast(this,0,deliver_sms,0);
```

Теперь создаем объект `BroadcastReceiver`, необходимый для получения результата. Такой объект нужно зарегистрировать для каждого отложенного действия:

```
// Получаем отчет об отправке
BroadcastReceiver sentReceiver = new BroadcastReceiver(){
@Override public void onReceive(Context c, Intent in) {
      switch(getResultCode()){
            case Activity.RESULT_OK:
                  // SMS отправлено, выполняем какие-то действия
                 break;
            default:
                 // Сбой
                 break;
     }
} };
// Получаем отчет о доставке
BroadcastReceiver deliverReceiver = new BroadcastReceiver(){
@Override public void onReceive(Context c, Intent in) {
      switch(getResultCode()){
            case Activity.RESULT_OK:
                  // SMS доставлено, выполняем какие-то действия
                  break;
            default:
                  // Сбой
                  break;
     }
} };
// Регистрируем BroadcastReceiver
registerReceiver(sentReceiver, new IntentFilter(SENT_SMS_FLAG));
registerReceiver(deliverReceiver, new IntentFilter(DELIVER_SMS_FLAG));
```

В большинстве случаев SMS не должно превышать 140 байтов. Для отправки более длинных сообщений служит метод `divideMessage()`, разбивающий сообщения на фрагменты, равные максимальному размеру SMS-сообщения. Отправка такого сообщения осуществляется методом `sendMultipartTextMessage()`, который используется вместо метода `sendTextMessage()`. Для получения отчета о доставке (или отправке) нужно задействовать уже не одно отложенное событие, а массив таких

событий. Количество элементов в таком массиве будет равно количеству частей, на которые было разбито исходное сообщение:

```
ArrayList<String> multiSMS = sendSMS.divideMessage(msg);
ArrayList<PendingIntent> sent_sms = new ArrayList<PendingIntent>();
ArrayList<PendingIntent> deliver_sms = new ArrayList<PendingIntent>();

for(int i=0; i< multiSMS.size(); i++){
        sentIns.add(sentIn);
        deliverIns.add(deliverIn);
}

sendSMS.sendMultipartTextMessage(num, null,
multiSMS, sentIns, deliverIns);
```

Ранее было сказано, что максимальная длина SMS составляет 140 байтов. Обратите внимание: именно байтов, а не символов. Когда вы отправляете SMS латинскими символами, то 140 байтов и означает 140 символов. Когда же вы используете в SMS-сообщении символы национальных алфавитов — например, кириллицу, то максимальная длина сокращается до 70 символов. Для кодирования символов национальных алфавитов используется кодировка UCS-2, где каждый символ представлен двумя байтами (16-ю битами), поэтому количество символов, которые можно отправить в одной SMS, сокращается до 70. Помните об этом!

12.2. Работа с браузером

Запустить браузер для отображения заданной страницы Интернета можно с помощью действия `ACTION_NEW` — сделать это достаточно просто:

```
Intent browser = Intent(Intent.ACTION_VIEW);
browser.setData(Uri.parse("https://www.dkws.org.ua"));
startActivity(browser);
```

Но запуск браузера нам мало интересен. Подумайте сами: напишете вы приложение, запускающее браузер. И какой толк от него будет? Пользователю проще напрямую запустить браузер.

Было бы гораздо интереснее создать свой браузер на основе класса `WebView`. Класс `WebView` использует для отображения веб-страниц движок WebKit (открытый движок браузера) — на этом же движке построен браузер Apple Safari и некоторые другие.

В файл манифеста для использования Интернета нужно добавить разрешение:

```
<uses-permission android:name="android.permission.INTERNET" />
```

К приложению надо подключить два пакета:

```
import android.webkit.WebView;
import android.webkit.WebSettings;
```

Есть два способа заполучить объект класса WebView. Первый заключается в использовании конструктора класса WebView, второй — в получении этого объекта из разметки приложения (предварительно его нужно поместить в разметку с помощью визуального редактора разметки):

```
// Первый способ
WebView browser = new WebView(this);
// Второй способ
WebView browser = (WebView) findViewById(R.id.webview);
```

После этого можно загрузить документ:

```
browser.loadUrl("http://www.dkws.org.ua/");
```

При желании можно загружать HTML-код из строки:

```
String html = "<html><body><h1>Hello</h1></body></html>";
browser.loadData(html, "text/html", "utf-8");
```

Для настройки браузера используется класс WebSettings:

```
WebSettings webSettings = webView.getSettings();
// Блокируем картинки для экономии трафика
webSettings.setBlockNetworkImage(true);
// Запрещаем сохранять данные форм
webSettings.setSaveFormData(false);
// Разрешаем JavaScript
webSettings.setJavaScriptEnabled(true);
// Запрещаем сохранять пароли
webSettings.setSavePassword(false);
// Устанавливаем размер шрифта по умолчанию (от 1 до 72)
webSettings.setDefaultFixedFontSize(2);
// Устанавливаем название нашего браузера
webSettings.setUserAgentString("My browser v 1.0");
```

Подробно о методах класса WebSettings (а значит, и о параметрах браузера) вы можете прочитать в руководстве разработчика Android:

http://developer.android.com/reference/android/webkit/WebSettings.html

Класс WebView тоже описан в руководстве разработчика:

http://developer.android.com/reference/android/webkit/WebView.html

ГЛАВА 13

База данных SQLite

13.1. Введение в базы данных для Android

В этой книге мы познакомились почти со всеми основными способами хранения данных на Android-устройстве (см. *главу 8*). Осталось рассмотреть базы данных (БД), которые используются для хранения более сложных структур данных.

Мы не станем здесь изучать основы баз данных и начала языка запросов SQL. Предполагается, что вы уже с этими темами знакомы. Если это не так, то на полках книжного магазина, я уверен, вы найдете много книг, способных заполнить этот пробел в ваших знаниях.

В ОС Android принята система управления базами данных (СУБД) SQLite. Да, возможности SQLite относительно скромны. Но ведь и такая мощная СУБД, как Oracle, в мобильном телефоне ни к чему. Зато SQLite — самая быстрая СУБД для несложных запросов на сегодняшний день.

На Android-устройстве база данных хранится в каталоге /data/data/<имя пакета>/databases. С помощью контент-провайдера несколько приложений могут использовать одну и ту же базу данных.

Основные этапы при работе с базой данных следующие:

- создание и открытие базы данных;
- создание таблицы;
- создание Insert-интерфейса (используется для вставки данных);
- создание Query-интерфейса (используется для выполнения запроса, обычно для выборки данных);
- закрытие базы данных.

Далее будут продемонстрированы основные приемы при работе с базой данных: вставка, удаление и выборка записей.

Если вы раньше работали с PHP и MySQL, то вас здесь не ожидают никакие неожиданности. Для всех остальных читателей постараюсь объяснить все максимально подробно.

Обратите также внимание: приступая к написанию кода, помните, что уже есть готовый класс `SQLiteOpenHelper`, который вы можете наследовать, а не «изобретать колесо» заново. Ведь в своих проектах мы всегда наследуемся от `Activity` (`extends Activity`). Аналогично нужно поступить и работая с базой данных.

13.2. Подготовка вспомогательного класса

Создайте новый проект как обычно. Пусть он называется `android_db`, а пакету присвойте название вида com.example.<ваше_имя>.android_db — например, com.example.den.android_db. В качестве шаблона разметки выберите **Empty Activity** (рис. 13.1).

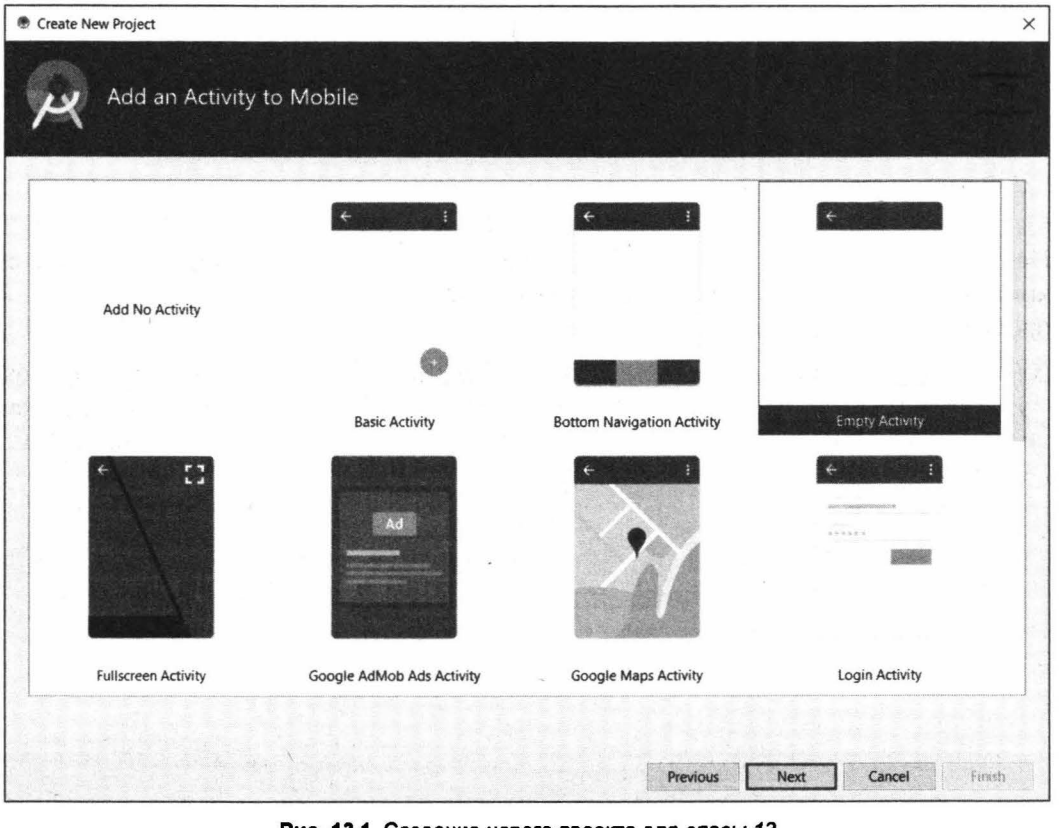


Рис. 13.1. Создание нового проекта для *главы 13*

После этого создайте новый класс: **File | New | Java Class**. В открывшемся диалоговом окне введите название нового класса: `MyDataBase` (рис. 13.2).

Если вы заполнили окно **Create New Class**, как показано на рис. 13.2, в области **Project** основного окна Android Studio появится файл MyDataBase.java. Добавьте в него следующее содержимое (листинг 13.1), как показано на рис. 13.3.

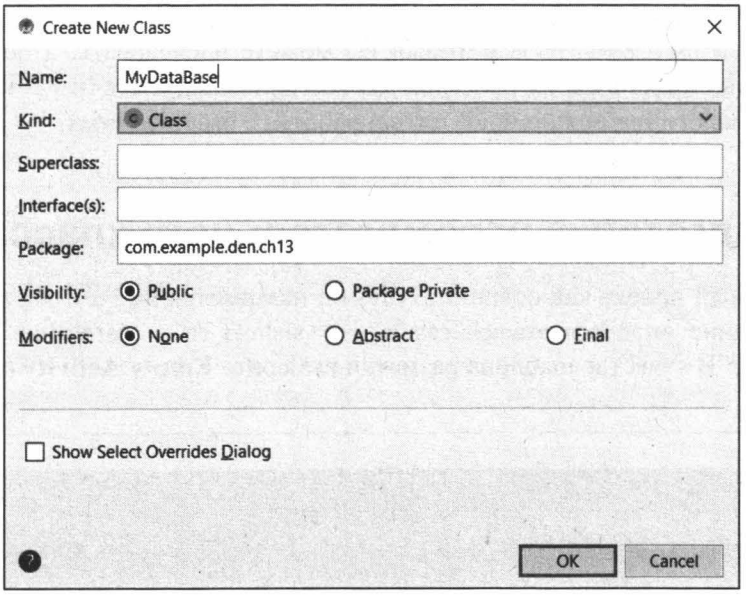


Рис. 13.2. Создание нового Java-класса

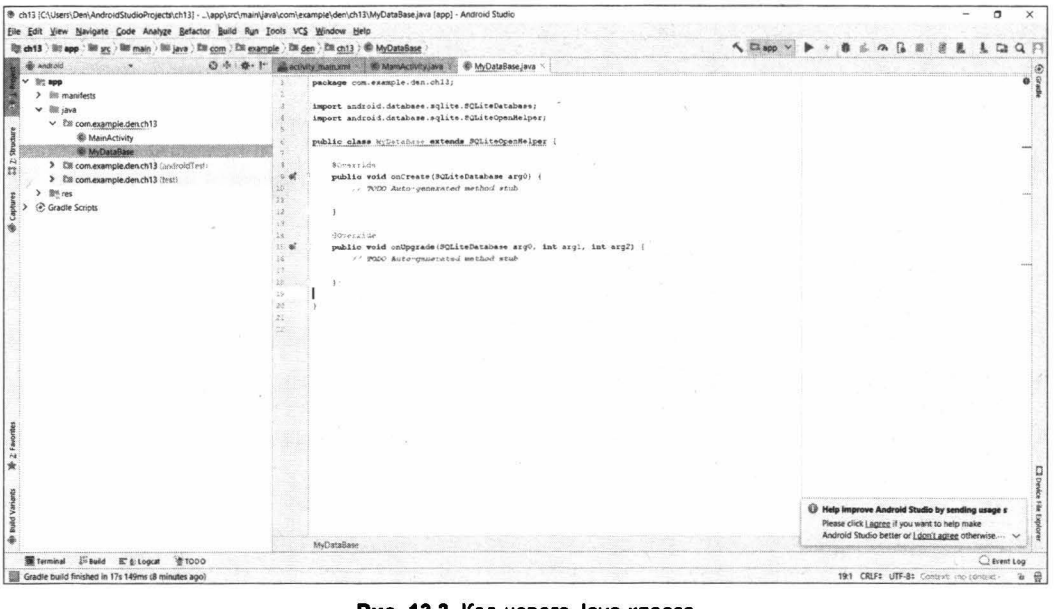


Рис. 13.3. Код нового Java-класса

Листинг 13.1. Заготовка для класса (файл MyDataBase.java)

```
package com.example.den.ch13;

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```

```
public class MyDataBase extends SQLiteOpenHelper {

	@Override
	public void onCreate(SQLiteDatabase arg0) {
		// TODO Auto-generated method stub

	}

	@Override
	public void onUpgrade(SQLiteDatabase arg0, int arg1, int arg2) {
		// TODO Auto-generated method stub

	}

}
```

У класса должны быть два обязательных метода: `onCreate()` и `onUpgrade()`, о которых мы поговорим чуть позже.

Обратите внимание: на рис. 13.3 название класса `MyDataBase` подчеркнуто красной волнистой линией, а напротив строки объявления класса стоит значок ошибки. Все в порядке — просто среда требует, чтобы мы создали конструктор класса.

Вот код этого конструктора:

```
// константы для конструктора
private static final String DATABASE_NAME = "my_database.db";
private static final int DATABASE_VERSION = 1;

public MyDataBase(Context context) {
	// TODO Auto-generated constructor stub
	super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

Кроме кода конструктора мы добавили еще несколько вспомогательных констант: первая константа задает имя файла с базой данных, вторая отвечает за номер базы данных. Принцип работы номера базы такой же, как и у версии приложения. Если вы видите, что вышла версия приложения 11, а у вас 10, то его пора обновить. Аналогично, если программа заметит обновление номера базы данных, она запускает метод `onUpgrade()`, который сгенерирован автоматически. В нем нужно разместить код, который должен вызываться при обновлении базы данных.

С методом `onCreate()` проще — в нем создается сама база данных с данными, необходимыми для работы.

Кроме имени файла базы данных и ее версии надо указать имя таблицы и имена колонок таблицы. Давайте создадим простую таблицу, которая будет содержать идентификатор пользователя (`UID`) и его имя (`UNAME`):

```
public static final String UID = "_id";
public static final String UNAME = "uname";
```

Первая константа — это непросто идентификатор. Это поле с автоматическим инкрементом. Оно будет увеличиваться автоматически при добавлении новой строки в таблицу.

В листинге 13.2 приведен полный код нашего вспомогательного класса (рис. 13.4). Поскольку рассмотрение операторов SQL выходит за рамки этой книги, то без краткого пояснения все же не обойтись, а остальную информацию (если вы не знакомы с SQL) можно найти в Интернете. Итак, для создания таблицы используется оператор `CREATE TABLE`. Для уничтожения таблицы — оператор `DROP TABLE`. SQL-операторы выполняются методом `execSQL()`.

Листинг 13.2. Полный код вспомогательного класса

```
package com.example.den.ch13;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class MyDataBase extends SQLiteOpenHelper {

        private static final String DATABASE_NAME = "my_database.db";
        private static final int DATABASE_VERSION = 11;
        public static final String TABLE_NAME = "contact_table";
        public static final String UID = "_id";
        public static final String UNAME = "uname";

        private static final String SQL_CREATE_ENTRIES = "CREATE TABLE "
           + TABLE_NAME + " (" + UID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
           + UNAME + " VARCHAR(255));";

        private static final String SQL_DELETE_ENTRIES = "DROP TABLE IF EXISTS "
           + TABLE_NAME;

        public MyDataBase(Context context) {
           // TODO Auto-generated constructor stub
           super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {
           // TODO Auto-generated method stub
           db.execSQL(SQL_CREATE_ENTRIES);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        // TODO Auto-generated method stub
        Log.w("LOG_TAG", "Upgrading DB from version " + oldVersion
                + " to version " + newVersion);

        // Удаляем предыдущую таблицу при обновлении
        db.execSQL(SQL_DELETE_ENTRIES);
        // Создаем новый экземпляр таблицы
        onCreate(db);
    }
}
```

Рис. 13.4. Полный код вспомогательного класса

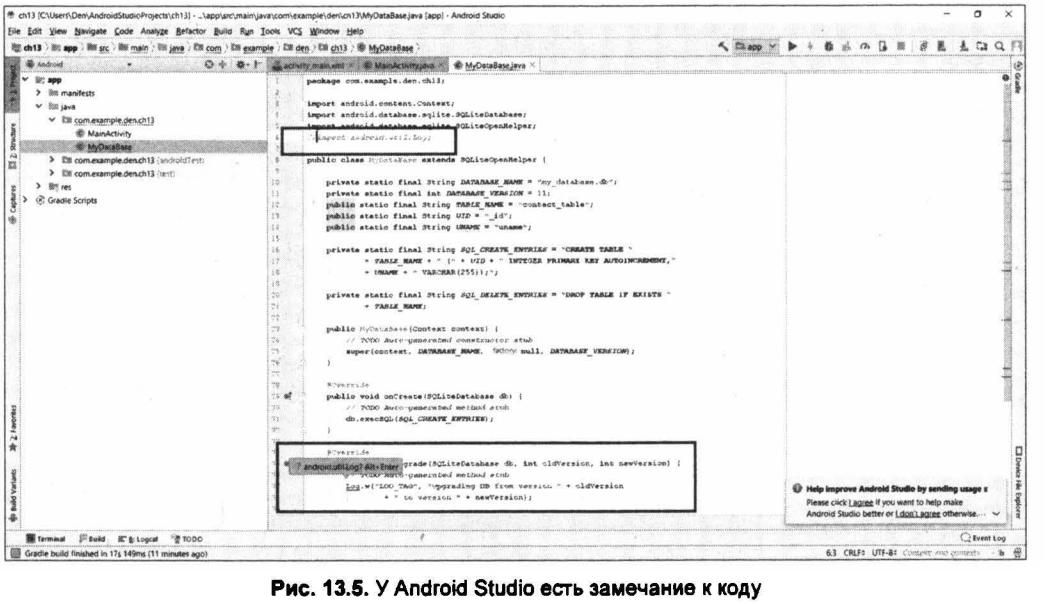


Рис. 13.5. У Android Studio есть замечание к коду

Зачем я дублирую на рис. 13.4 исходный код двух предыдущих листингов? Чтобы вы видели, что у среды нет никаких замечаний к коду. Если же у вас что-то подчеркнуто или имеется какое-либо предупреждение, значит, вы где-то допустили ошибку. Например, если забыть импортировать `android.util.Log`, то Android Studio сразу обратит на этот факт ваше внимание (рис. 13.5).

13.3. Работа с базой данных

13.3.1. Создание базы данных

Половина этой главы была потрачена на написания вспомогательного класса, но до результата еще далеко. Поэтому самое время перейти на нашу главную деятельность (`activity`) и изменить ее метод `onCreate()` (рис. 13.6):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Инициализируем наш вспомогательный класс
    MyDataBase mydb = new MyDataBase(this);

    // Открываем БД для чтения и записи
    SQLiteDatabase sqdb = mydb.getWritableDatabase();

    // Закрываем соединения с базой данных
    sqdb.close();
    mydb.close();
}
```

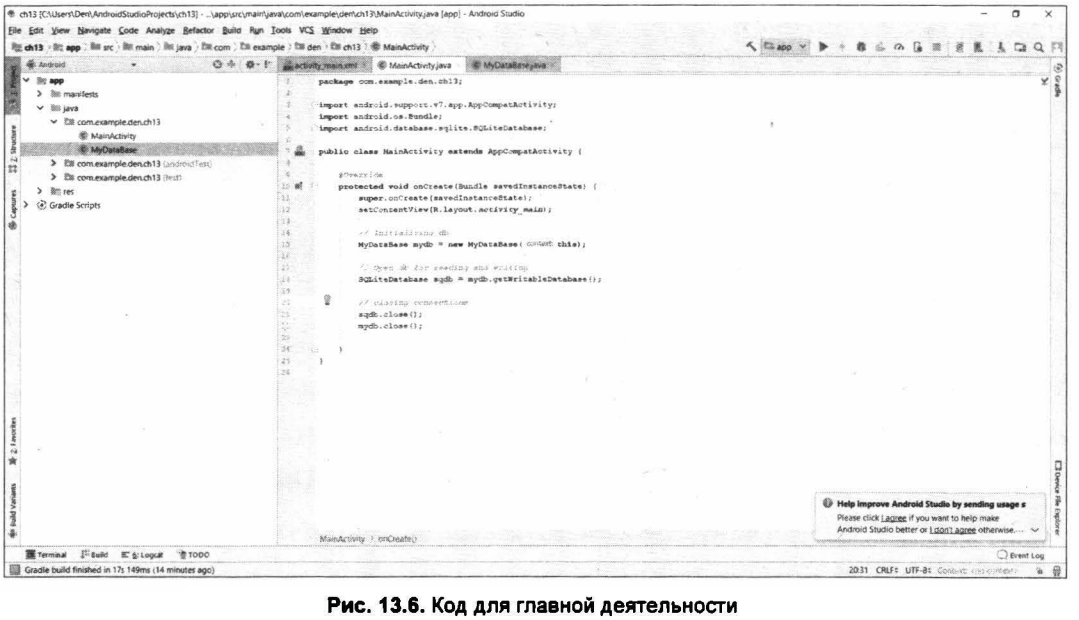


Рис. 13.6. Код для главной деятельности

Теперь запустите проект. Вам покажется, что ничего не произошло, но на самом деле в каталоге \data\data\com.example.den.android_db\databases эмулятора (или реального устройства — смотря как вы предпочитаете запускать приложения) появится файл my_database.db.

Чтобы убедиться в этом, командой меню **View | Tool Windows** откройте **Device File Explorer** и просмотрите структуру файлов виртуального устройства (рис. 13.7).

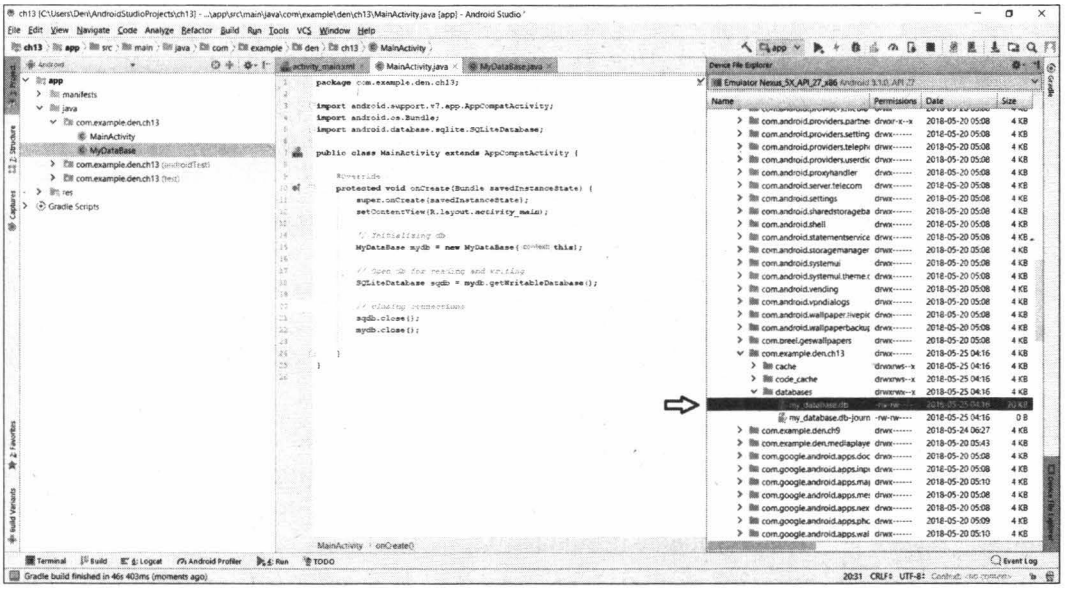


Рис. 13.7. База данных создана

На этом подготовка базы данных к работе завершена, и мы можем начать с ней работать, т. е. вставлять в нее данные и извлекать их.

13.3.2. Вставка записей

Для вставки записей в БД можно использовать или класс `ContentValues`, или обычный SQL-запрос `INSERT`. Если вы раньше имели дело с SQL (хоть в PHP, хоть в любом другом языке программирования), то вам будет проще использовать именно SQL-запрос `INSERT`.

Вот пример вставки данных с использованием SQL-запроса `INSERT`:

```
String insertQuery = "INSERT INTO " +
    MyDataBase.TABLE_NAME + " (" + MyDataBase.UNAME + ") VALUES ('" +
     txtData.getText().toString() + "')";
    sqdb.execSQL(insertQuery);
```

Тот же код, но в сокращенном виде:

```
String insertQuery = "SQL-запрос";
sqdb.execSQL(insertQuery);                    // выполняем запрос
```

13.3.3. Чтение данных

Прочитать данные из базы данных можно или методом `query()`, или методом `rawQuery()`. Рассмотрим сначала первый метод:

```
Cursor cursor = sqdb.query(MyDataBase.TABLE_NAME, new String[] {
                    MyDataBase._ID, MyDataBase.UNAME },
                    null,  // Колонки для WHERE-части
                    null,  // Значения для WHERE-части
                    null,  // Не группировать строки
                    null,  // Не фильтровать группы строк
                    null   // Порядок сортировки
                    );
      while (cursor.moveToNext()) {
            // Получаем индексы колонки и значения этих колонок
            int id = cursor.getInt(cursor.getColumnIndex(MyDataBase._ID));
            String name = cursor.getString(cursor
                        .getColumnIndex(MyDataBase.CATNAME));
            Log.i("LOG_TAG", "ROW " + id + " HAS UNAME " + name);
      }
      cursor.close();
```

Если вы внимательно прочитали комментарии и знакомы со структурой оператора `SELECT`, метод `query()` должен быть вам понятен. Мы сформировали запрос и передали его базе данных. В ответ мы получаем все колонки, которые соответствуют запросу, и выводим их в журнал с помощью `Log.i()`.

Метод `rawQuery()` отличается тем, что ему можно непосредственно передать SQL-запрос:

```
String query = "SELECT " + MyDataBase._ID + ", "
                         + MyDataBase.UNAME + " FROM " + MyDataBase.TABLE_NAME;

   Cursor cursor2 = sqdb.rawQuery(query, null);

   while (cursor2.moveToNext()) {
     int id = cursor2.getInt(cursor2.getColumnIndex(MyDataBase._ID));
     String name = cursor2.getString(cursor2.getColumnIndex(MyDataBase.UNAME));
     Log.i("LOG_TAG", "ROW " + id + " HAS UNAME " + name);
   }
   cursor2.close();
```

Здесь мы сначала формируем запрос. Затем передаем его методу `rawQuery()`. После этого читаем все строки результата с помощью метода `moveToNext()`. Получаем результаты и выводим их с помощью `Log.i()` в журнал.

* * *

Итак, мы научились создавать базу данных, вставлять в нее данные и извлекать их из нее. Извлеченные данные мы выводили в журнал, просмотреть который можно с помощью среды Android Studio. В реальных же приложениях вы вольны делать с данными все, что пожелаете, — например, вывести их в `TextView`.

ГЛАВА 14

Создание анимации

14.1. Анимация преобразований

В Android используются два подхода к созданию анимации: кадровая анимация (Frame Animation) и анимация преобразований (Tween Animation). Сначала мы рассмотрим анимацию преобразований, а затем — кадровую анимацию.

Анимация преобразований выполняется путем выполнения над изображением ряда преобразований: вращения, добавления прозрачности, изменения размера и т. п. В пакете `android.view.animationpackages` имеются все необходимые для выполнения преобразований классы:

- `AnimationSet` (XML-элемент `<set>`) — представляет группу анимаций, которые запускаются вместе;
- `AlphaAnimation` (XML-элемент `<alpha>`) — управляет прозрачностью объекта;
- `RotateAnimation` (XML-элемент `<rotate>`) — управляет вращением объекта;
- `ScaleAnimation` (XML-элемент `<scale>`) — управляет масштабированием (изменением размера) объекта;
- `TranslateAnimation` (XML-элемент `<translate>`) — управляет позиционированием объекта.

Анимацию преобразования можно создать как в XML-файле, так и в программном коде. Начнем с XML-файла. Файл анимации помещают в каталог res\anim. В этом файле имеется единственный корневой элемент. Таким элементом может быть один из элементов преобразования: `<rotate>`, `<alpha>`, `<scale>`, `<translate>` или `<set>`, который является контейнером для этих четырех элементов (он позволяет задать несколько преобразований).

По умолчанию все описанные в `<set>` преобразования выполняются одновременно. Чтобы они выполнялись последовательно (в порядке описания в XML-файле), нужно использовать атрибут `startOffset`, задающий задержку между выполнением преобразований в миллисекундах:

```
android:startOffset="1000"
```

У всех элементов преобразований есть общие атрибуты, описанные в табл. 14.1.

***Таблица 14.1.** Общие атрибуты элементов преобразований*

Атрибут	Описание
`duration`	Продолжительность преобразования в миллисекундах
`startOffset`	Время смещения перед выполнением заданного эффекта, в миллисекундах. Можно использовать для задержки между преобразованиями
`fillBefore`	Если равен `true`, преобразование анимации выполняется перед началом анимации
`fillAfter`	Если равен `true`, преобразование выполняется после завершения анимации
`repeatCount`	Определяет количество повторений анимации
`repeatMode`	Позволяет задать режим повтора: 1 — анимация начинается заново с самого начала, 2 — анимация будет произведена в обратном порядке
`zAdjustment`	Смещение по оси Z: 0 (обычное смещение, без изменений); 1 (вершина); –1 (основание)
`interpolator`	Позволяет указать интерполятор анимации. Имя интерполятора указывается в формате `@[+][package:]type:name`. Пример: `android:interpolator="@android:anim/decelerate_interpolator`

Теперь рассмотрим собственные атрибуты элементов преобразования.

- У элемента `<set>` есть атрибут `shareInterpolator`. Если он равен `true`, то интерполятор, заданный атрибутом `interpolator` для `<set>`, будет использоваться для всех дочерних элементов преобразования, входящих в этот элемент `<set>`.
- У элемента `<alpha>` есть атрибуты `fromAlpha` и `toAlpha`. Первый задает начальное значение прозрачности объекта, второй — конечное. Значения прозрачности находятся в диапазоне от 0 до 1, где 0 — полная прозрачность объекта (объект почти невидим).
- У элемента `<translate>`, создающего вертикальную или горизонтальную анимацию, атрибутов больше:
  - `fromXDelta` — начальное положение по оси *X*;
  - `toXDelta` — конечное положение по оси *X*;
  - `fromYDelta` — начальное положение по оси *Y*;
  - `toYDelta` — конечное положение по оси *Y*.

  Атрибуты могут принимать либо абсолютное значение, либо значение в процентах: от –100 до 100%. Можно также указывать процент относительно родителя: от `-100%p` до `100%p`.
- Элемент `rotate` служит для анимации вращения и поддерживает атрибуты:
  - `fromDegrees` — начальный угол вращения в градусах;
  - `toDegrees` — конечный угол вращения в градусах;

- pivotX — задает координату *X* центра вращения в пикселах;
- pivotY — задает координату *Y* центра вращения в пикселах.

- Элемент <scale> управляет изменением размера объекта. Вы можете использовать следующие атрибуты:
  - fromXScale — начальный масштаб по оси *X*;
  - toXScale — конечный масштаб по оси *X*;
  - fromYScale — начальный масштаб по оси *Y*;
  - toYScale — конечный масштаб по оси *Y*;
  - pivotX — координата (*X*) закрепленного центра;
  - pivotY — координата (*Y*) закрепленного центра.

Вернемся к практике. Прежде всего нужно создать XML-файл анимации в каталоге res\anim. В листинге 14.1 приведен простой файл анимации, содержащий только элемент <alpha> внутри элемента <set>. В листинге 14.2 приведена более сложная анимация, т. к. задано два анимационных эффекта.

Листинг 14.1. Файл res\anim\alpha.xml

```
<?xml version="1.0" encoding="utf-8"
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">
<alpha
       android:fromAlpha="0.0"
       android:toAlpha="1.0"
       android:startOffset="0"
       android:duration="3000"/>
</set>
```

Листинг 14.2. Файл res\anim\advanced.xml

```
<?xml version="1.0" encoding="utf-8"
<set xmlns:android="http://schemas.android.com/apk/res/android"
android:shareInterpolator="false">
<alpha
       android:fromAlpha="0.0"
       android:toAlpha="1.0"
       android:startOffset="0"
       android:duration="3000"/>
<translate
       android:toYDelta="-100"
       android:fillAfter="true"
       android:duration="2500"/>
</set>
```

Теперь займемся Java-кодом. Сначала нужно подключить необходимые пакеты. Далее — загрузить картинку, над которой будет производиться анимация. Затем загрузить саму анимацию. Это можно сделать методом `loadAnimation()` из `AnimationUtils`. После чего запустить анимацию методом `startAnimation()`. Весь этот процесс описан в листинге 14.3.

Листинг 14.3. Создание анимации

```
// Подключаем необходимые пакеты
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.view.animation.Animation.AnimationListener;
...
// Загружаем картинку
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла alpha.xml
Animation animation = AnimationUtils.loadAnimation(this, R.anim.alpha);
// Запускаем анимацию
image.startAnimation(animation);
```

Как видите, ничего сложного в анимации преобразования нет. Необходимо только подобрать нужный эффект (или группу эффектов).

14.2. Традиционная кадровая анимация

Традиционная кадровая анимация представляет собой последовательную смену различных изображений — подобно киноленте. Основой для этого типа анимации является класс `AnimationDrawable`.

Как и в случае с анимацией преобразований, начнем создавать анимацию с XML-файла, который, как обычно, нужно поместить в каталог **res\anim**. В листинге 14.4 приведен пример такого файла.

Листинг 14.4. Файл res\anim\frames.xml

```
<animation-list xmlns:android="http://schemas.android.com/apk/res/android"
android:oneshot="false">
<item android:drawable="@drawable/frame1" android:duration="100"/>
<item android:drawable="@drawable/frame2" android:duration="100"/>
<item android:drawable="@drawable/frame3" android:duration="100"/>
<item android:drawable="@drawable/frame4" android:duration="100"/>
<item android:drawable="@drawable/frame5" android:duration="100"/>
</animation-list>
```

У нас есть пять кадров (`@drawable/frame1` — `@drawable/frame5`), длительность отображения каждого кадра — 100 мс. Если параметр `android:oneshot` установлен

в false, то анимация будет циклической. Если же этот параметр принимает значение true, то анимация отобразится только один раз, а после остановки будет показан последний кадр.

Код запуска анимации несложен:

```
// Находим область ImageView
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла frames.xml
image.setBackgroundResource(R.anim.frames);
AnimationDrawable myAnim = (AnimationDrawable)image.getBackground();
```

Запустить и остановить анимацию можно методами start() и stop():

```
myAnim.start();
myAnim.stop();
```

В листинге 14.5 приведен более полный код запуска анимации.

Листинг 14.5. Запуск кадровой анимации

```
// Подключаем необходимый пакет
import android.graphics.drawable.AnimationDrawable;
...
// Находим область ImageView
ImageView image = (ImageView)findViewById(R.id.image);
// Загружаем анимацию из файла frames.xml
image.setBackgroundResource(R.anim.frames);
AnimationDrawable myAnim = (AnimationDrawable)image.getBackground();
// Запускаем анимацию
myAnim.start();
```

В случае с кадровой анимацией можно обойтись и без XML-файла. Сначала нужно заполучить кадры, которые мы будем использовать для анимации (файлы frame1.png — frame5.png):

```
BitmapDrawabie frame1 =
(BitmapDrawable)getResources().getDrawable(R.drawable.frame1);
...
BitmapDrawabie frame5 =
(BitmapDrawable)getResources().getDrawable(R.drawable.frame5);
```

Затем создать объект класса AnimationDrawable. Атрибут oneshot устанавливается методом setOneShot() этого класса, а добавить кадры можно методом addFrame():

```
AnimationDrawable myAnim = new AnimationDrawable();
myAnim.setOneShot(false);
myAnim.addFrame(frame1, 100);
...
myAnim.addFrame(frame5, 100);
```

Далее порядок действий тот же: нужно установить анимацию в качестве фона для объекта класса `ImageView` (объект `image`), сделать объект `AnimationDrawable` видимым и запустить анимацию:

```
image.setBackgroundDrawable(MyAnim);
MyAnim.setVisible(true, true);
MyAnim.start();
```

Какой из способов выбрать — решать вам. XML-файлы удобно использовать для постоянной анимации (кадры которой не изменяются), а вот формирование кадров через метод `addFrame()` полезно, когда производится загрузка файлов из внешнего источника, например из SD-карты устройства.

ЧАСТЬ IV

Дополнительные материалы

ГЛАВА 15

App Inventor — среда быстрой разработки приложений

15.1. Введение в App Inventor

App Inventor — это среда визуальной разработки Android-приложений, не требующая, как утверждается, навыков программирования. Первоначально она была разработана для Google Labs, а затем передана Массачусетскому технологическому институту (другими словами, сейчас сама компания Google не имеет никакого отношения к этой среде).

Разработка приложений в App Inventor осуществляется с помощью визуального языка программирования, похожего на язык Scratch, но если вы не имеете отношения к программированию, то это мало вам что скажет.

Итак, давайте разберемся, что же представляет собой App Inventor. Разработчики Android уверяют, что с помощью App Inventor даже домохозяйка сможет разработать свое приложение без минимальных навыков программирования. Вам в это верится? Мне — нет. Однако, используя App Inventor, у вас все-таки получится создать приложение без знаний языка Java и, вообще, без знания какого-либо языка программирования. Приложения в App Inventor создаются по принципу конструктора: расставил кнопки и другие элементы управления в окне «смартфона», собрал выполняемые программой действия и, вуаля — программа готова. Ошибиться в действиях как бы тоже невозможно, поскольку все компоненты приложения собираются по принципу пазла.

В Интернете есть видео, где девушка, якобы не имеющая никакого отношения к программированию, создает за пару минут приложение. Приложение простое: нажимаешь на изображение кота — слышишь звук его мява:

http://www.youtube.com/watch?v=8ADwPLSFeY8.

Полагаю, общее впечатление о том, что такое App Inventor, у вас сформировалось. Теперь — мое отношение к этой среде. Если хочешь что-то выучить, то нужно учиться, разбираться, пробовать. Ничего просто так не дается. Да, App Inventor — хорошая попытка создать RAD-систему (систему быстрой разработки) для Android, но, на мой взгляд, попытка удалась не вполне.

В App Inventor мне не понравилось две особенности. Первая — весьма скудный функционал самой среды. Сложное Android-приложение с помощью App Inventor

вы вряд ли сможете построить (именно построить, а не разработать). Когда вы пишете приложение на Java, то ваши возможности ограничены, по сути, только вашими навыками в программировании. Если же в App Inventor не окажется нужного «пазла», то приложение создать не получится. Вторая — с помощью App Inventor простенькое приложение вы создадите, но размер APK-файла будет неприлично большим, я бы даже сказал — огромным.

Надо, впрочем, отметить, что большая часть моих претензий к среде App Inventor относится к ее первой версии. Вторая версия среды (App Inventor Version 2) по сравнению с первой стала значительно лучше. Во-первых, больше нет мучительной процедуры установки. Среда якобы была рассчитана на домохозяек, но для ее запуска на компьютере домохозяйки требовался опытный системный администратор. Сейчас надо лишь зайти на сайт среды и нажать кнопку **Create** — далее все будет работать, как и должно, без каких-либо танцев с бубном. Во-вторых, улучшена функциональность самой среды, появились дополнительные пазлы, создавать приложения стало удобнее. Но все равно вы остаетесь ограничены имеющейся функциональностью. Приведу небольшой пример. В Android нет диалогового окна выбора файла, который есть в API Windows. Вы не замечали, что у всех приложений Windows диалоговые окна открытия и сохранения файлов одинаковые? Это потому, что они реализованы на уровне API операционной системы. В Android такой реализации нет. Однако, используя Java и стандартные виджеты Android, вы можете реализовать необходимые диалоговые окна. А вот в App Inventor подобная реализация отсутствует. Следовательно, создать Android программу с диалоговым окном открытия файла с помощью App Inventor у вас не получится.

15.2. Начало работы с App Inventor

Для работы с App Inventor нужно установить виртуальную машину Java. Последнюю ее версию можно получить по адресу: **http://www.java.com/ru/download/**.

Загрузив и установив виртуальную машину Java, перейдите на сайт MIT App Inventor (рис. 15.1) по адресу: **http://appinventor.mit.edu/explore/**.

На этом сайте находится много вспомогательных материалов, которые помогут разобраться со средой. Обязательно изучите их на досуге. А сейчас нажмите кнопку **Create apps!** в верхнем правом углу — вы увидите запрос на доступ к вашему Google-аккаунту. Разрешите приложению MIT AppInventor Version 2 такой доступ.

Далее среда предложит вам просмотреть краткий обзор (рис. 15.2). Вы можете нажать кнопку **Take Survey Now**, чтобы просмотреть обзор прямо сейчас, кнопку **Take Survey Later**, чтобы просмотреть обзор позже, и, наконец, кнопку **Never Take Survey**, чтобы вовсе отказаться от просмотра. Пока же, чтобы не отвлекаться от чтения книги, нажмите кнопку **Take Survey Later** — среда подскажет, как подключить и настроить Android-устройство или эмулятор (рис. 15.3).

Ознакомьтесь с приведенными ссылками (откройте их в новом окне или новой вкладке) и нажмите кнопку **Continue** — вы увидите список созданных вами проек-

тов. Понятное дело, список поначалу будет пуст, поскольку вы еще не успели ничего создать, а вот у меня уже кое-что создано (рис. 15.4).

Нажмите кнопку **New Project** для создания нового проекта. Введите название проекта и нажмите кнопку **OK** (рис. 15.5). Имя проекта не должно содержать пробелов и символов национальных алфавитов, должно начинаться с буквы, может содержать цифры и знак подчеркивания.

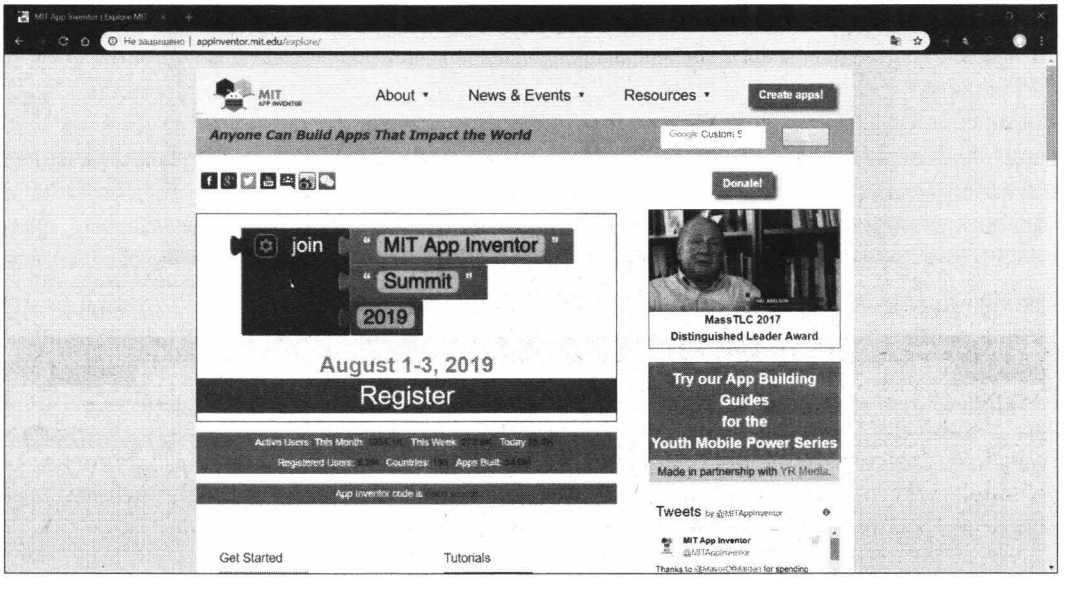


Рис. 15.1. Сайт appinventor.mit.edu

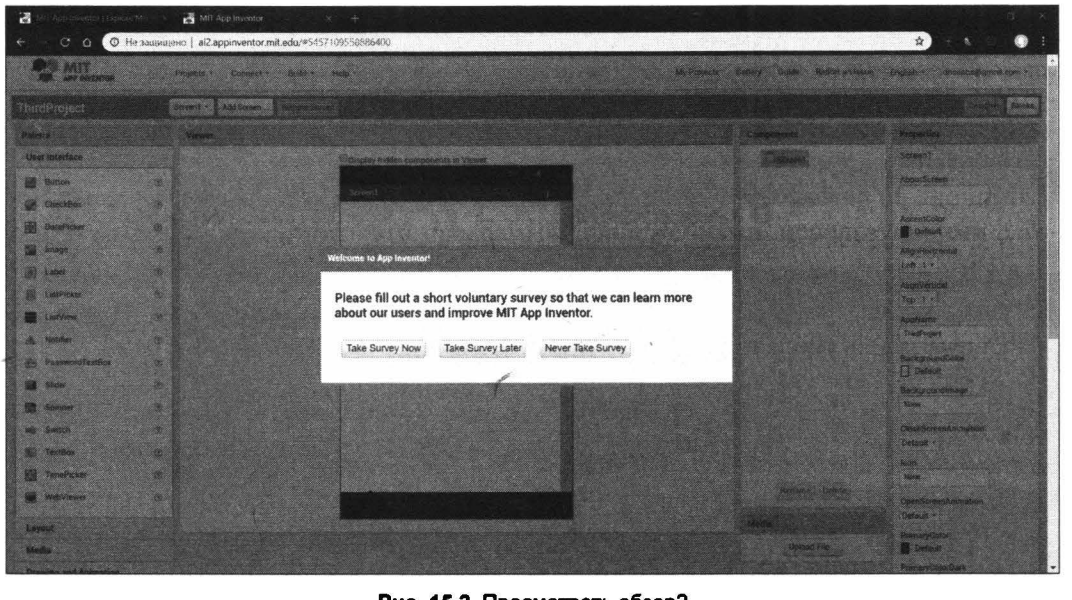


Рис. 15.2. Просмотреть обзор?

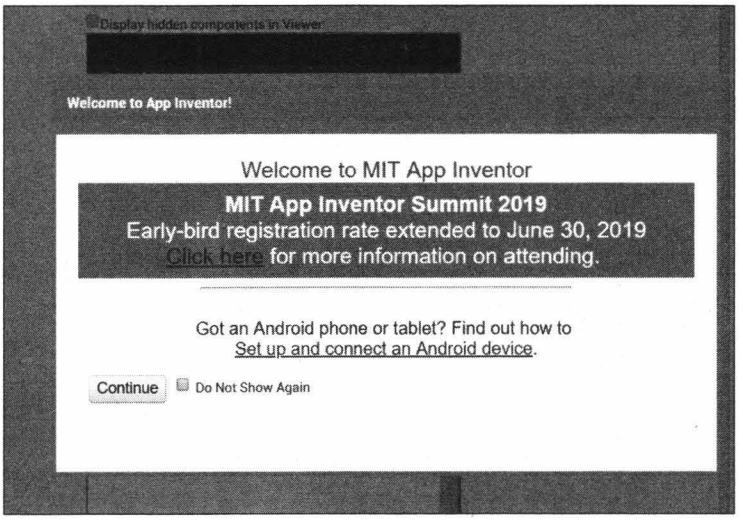


Рис. 15.3. Нажмите кнопку **Continue**

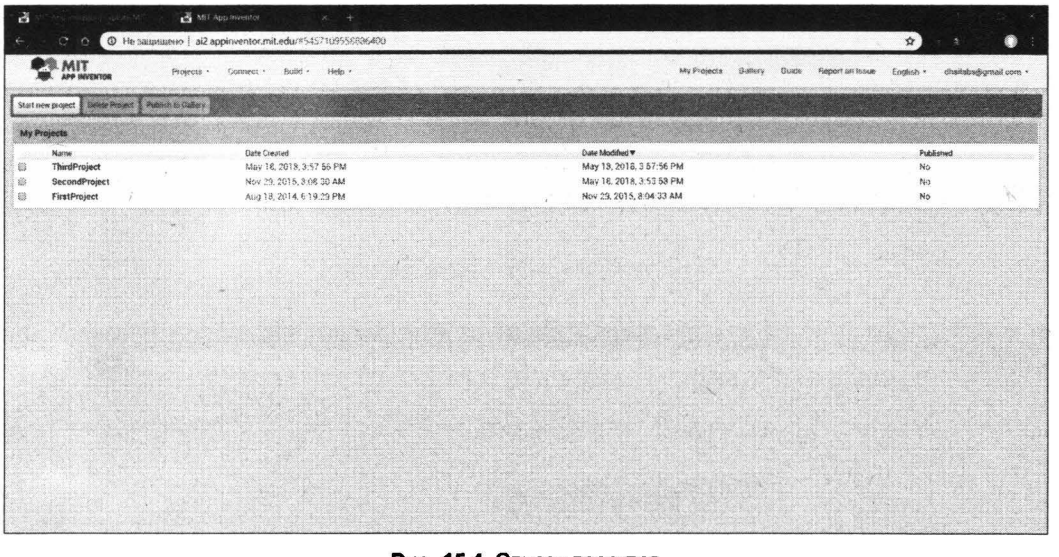


Рис. 15.4. Список проектов

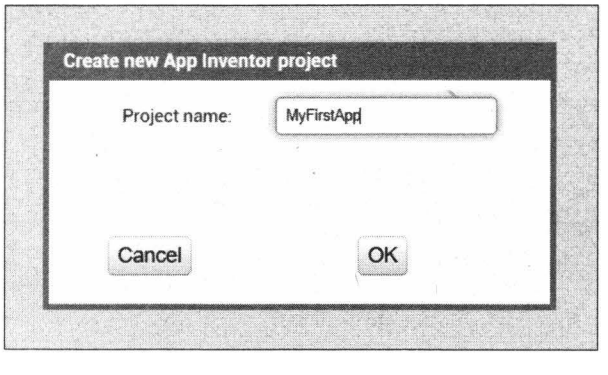


Рис. 15.5. Создание нового проекта

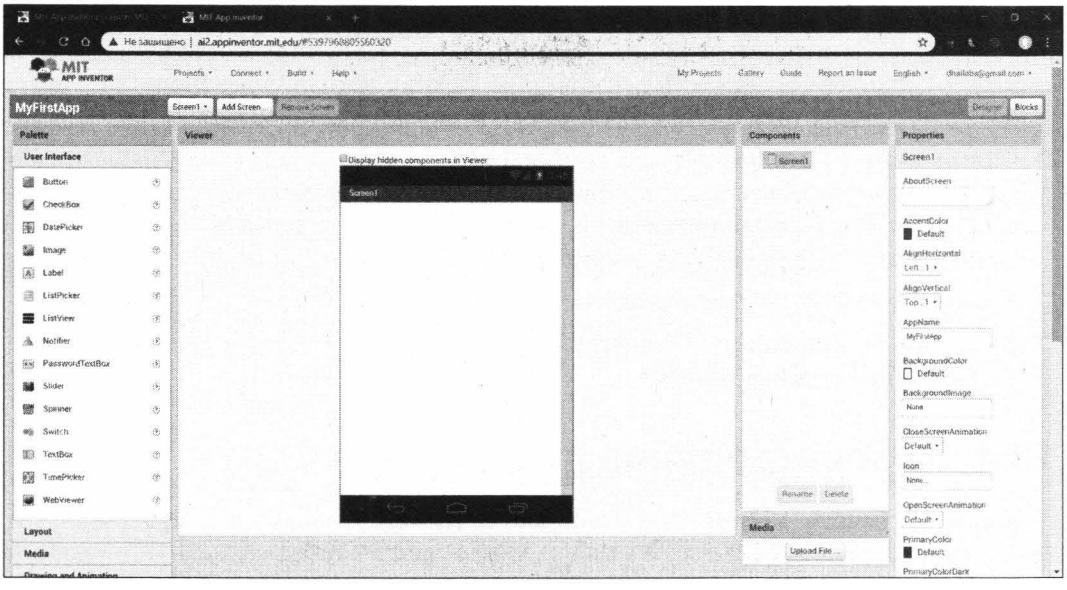


Рис. 15.6. Среда App Inventor

Наконец, вы увидите саму среду (рис. 15.6), описание которой приводится в следующем разделе.

15.3. Основной экран App Inventor

Прежде чем приступить к созданию первого приложения, давайте исследуем меню среды App Inventor (см. рис. 15.6).

- В меню **Project** сосредоточены основные команды управления проектом:
  - **My Projects** — выводит список ваших проектов.
  - **Start new project** — создает новый проект.
  - **Save project** — сохраняет проект.
  - **Save project as** — сохраняет проект под другим именем.
- Меню **Connect** содержит команды подключения к устройству (**USB**) или к эмулятору (**Emulator**).
- В меню **Build** находятся команды построения проекта. Так, команда **App (save .apk to my computer)** сохраняет APK-файл созданного приложения на ваш компьютер. Далее вы можете запустить его в эмуляторе (см. *главу 2*) или установить в своем смартфоне.

Весь экран в режиме проектирования разделен на пять областей:

- **Palette** — отображает палитру компонентов (элементов графического интерфейса): кнопок, надписей, полей ввода и т. д.;
- **Viewer** — область предварительного просмотра и проектирования графического интерфейса пользователя. Посмотрите на рис. 15.6 — сейчас у нас там есть один

экран: **Screen1**. Если вы добавите второй экран (с помощью кнопки **Add Screen**), то кнопка для переключения на редактирование этого второго экрана появится рядом с кнопкой **Screen1**. Переключатель **Display hidden components in Viewer** позволяет отобразить в области **Viewer** невидимые компоненты;

- **Components** — отображает все компоненты, добавленные на экран. Даже если компонент не отображается в области **Viewer**, он все равно будет присутствовать в области **Components**. На рис. 15.6 видно, что на первый экран (**Screen1**) не добавлено компонентов (есть только **Screen1**);
- **Media** — эта область отображает медиафайлы;
- **Properties** — свойства выбранного компонента. Например, на рис. 15.6 изображены свойства экрана **Screen1**.

15.4. Проектирование приложения

Добавьте в область **Viewer** два компонента: **Button** (кнопку) и **Label** (надпись), чтобы компоненты в областях **Viewer** и **Components** отображались, как показано на рис. 15.7.

Затем щелкните на надписи и в области **Properties** очистите свойство **Text** — это текст надписи. После этого надпись исчезнет из **Viewer** — точнее, она там останется, но так как вы очистили ее текст, она станет не видна. Чтобы вновь редактировать ее свойства, вам нужно будет щелкнуть на компоненте **Label1** в области **Components**.

Итак, вы научились изменять свойства компонента! Теперь опять выберите компонент **Label1** и присвойте свойству **Text** какой-нибудь текст, — например, `Hello!` Ведь по традиции первая программа должна содержать это слово!

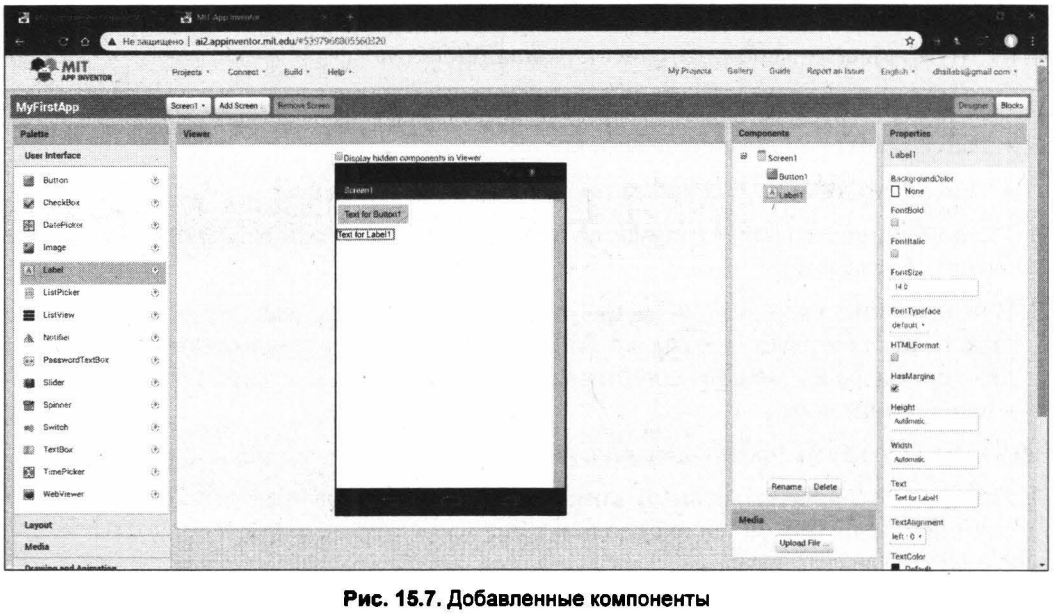


Рис. 15.7. Добавленные компоненты

Что ж, полдела сделано — мы расположили компоненты на экране. Теперь приступим к описанию действий программы. Наша программа не будет слишком оригинальной — при запуске на экране будут видны кнопка и надпись. По нажатию на кнопку надпись с экрана исчезнет. Пусть это и не самое сложное Android-приложение, но как первый опыт — вполне сгодится.

Нажмите кнопку **Blocks** для открытия редактора блоков. Начнем собирать пазлы! Сначала область редактора блоков будет пуста. Подумаем, что должно происходить? Мы же нажимаем на кнопку — следовательно, нам нужно действие-обработчик нажатия на кнопку.

Ваши компоненты отображаются в секции **Screen1** (или **Screen2**, **Screen3** — в зависимости от экрана). Выберите **Button1** — вы увидите все действия, доступные для выбранного компонента, т. е. для кнопки (рис. 15.8).

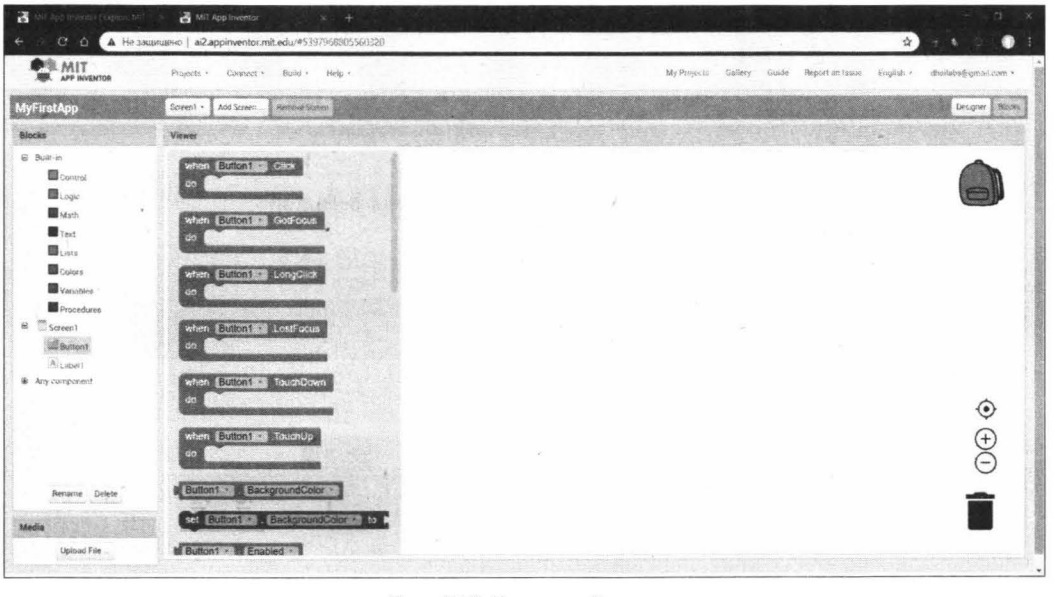


Рис. 15.8. Редактор блоков

Выберите пазл **when Button1.Click do** и добавьте его в свободную область редактора блоков (рис. 15.9). Поясню, что мы сделали, — выбрали обработчик нажатия кнопки **Button1**. Теперь осталось определить, что произойдет, когда будет нажата эта кнопка. Нам ведь надо скрыть надпись. Значит, первым делом в **Screen1** нужно выбрать надпись **Label1**, чтобы посмотреть, что можно с ней сделать.

Выберите действие **set Label1 Visible to**. Это действие устанавливает свойство **Visible** надписи **Label1** в определенное значение. У вас должна получиться конструкция, показанная на рис. 15.10.

Чтобы скрыть надпись, нужно установить ее свойство **Visible** в **false**. Вот и последний пазл нашей программы — из группы **Built-in** выбрать группу **Logic**, а из нее — значение **false**. У вас должна получиться конструкция, показанная на рис. 15.11.

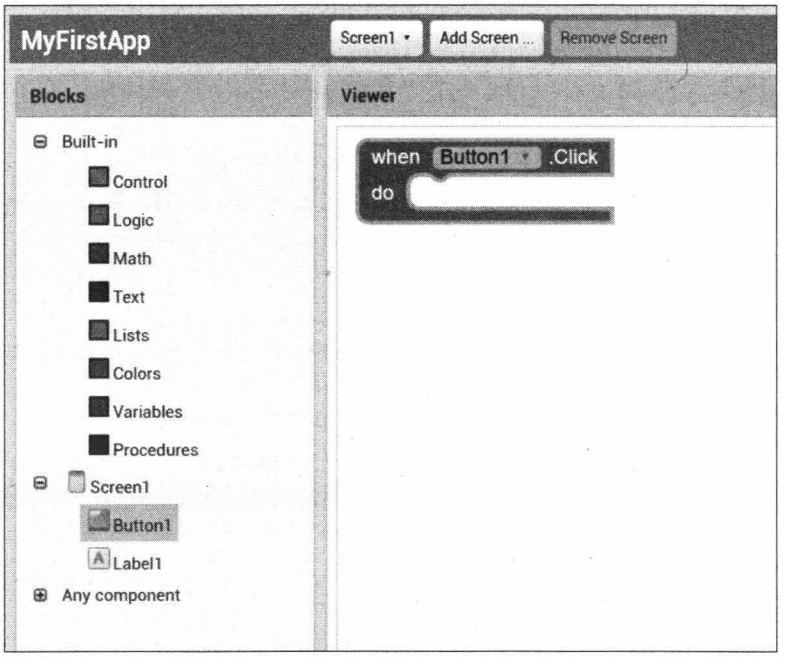


Рис. 15.9. Создание обработчика для кнопки **Button1**

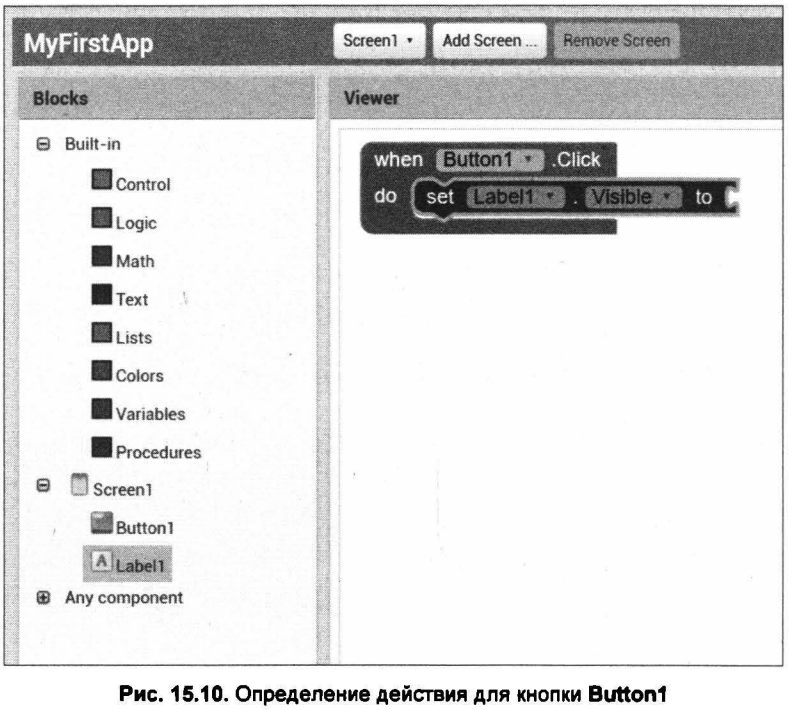


Рис. 15.10. Определение действия для кнопки **Button1**

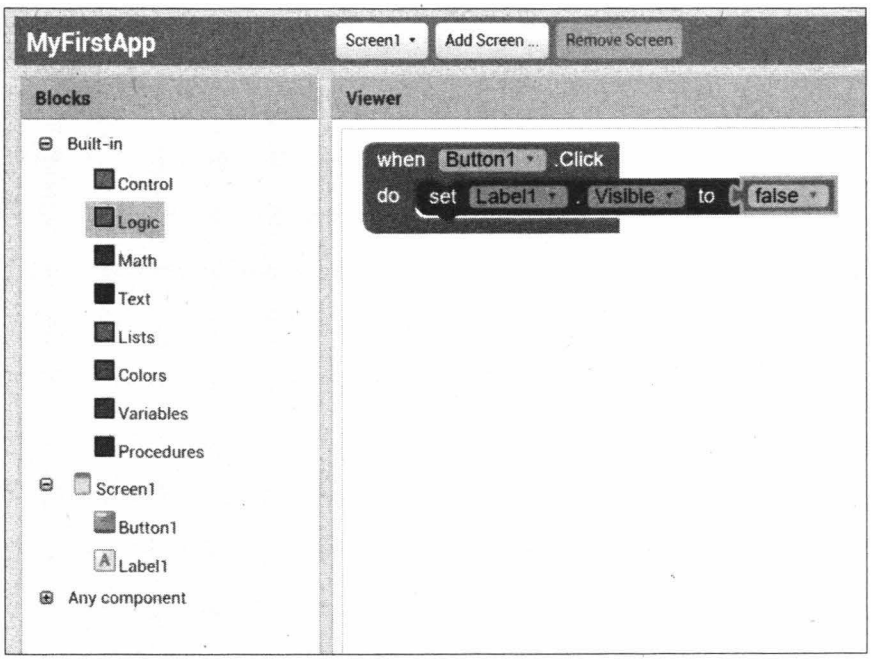


Рис. 15.11. Сборка программы завершена

Если нужно удалить какой-либо блок, просто перетащите его в корзину. По сути, наша первая программа готова. Выберите команду меню **App (save .apk to my computer)** из меню **Build**, а затем — каталог, в который нужно сохранить APK-файл. Далее или протестируйте его в эмуляторе, или установите в смартфоне. Напомню, что для тестирования программы в эмуляторе нужно открыть командную строку, перейти в каталог platform-tools и ввести команду:

```
adb install путь\MyFirstApp.apk
```

Как видите, создать программу с помощью App Inventor достаточно просто. Впрочем, скорее всего вы не будете использовать App Inventor при разработке реальных проектов, поскольку ее возможности сильно ограничены. Зато среда отлично подойдет для обучения программированию — например, если у вас появится желание научить своего ребенка создавать приложения для Android.

ГЛАВА 16

Отладка приложений

16.1. Используем Android Studio

Среда Android Studio содержит весьма мощные средства отладки приложений. Но в дополнение к ним вы можете использовать утилиты отладки из Android SDK, адаптированные для отладки именно Android-приложений. Сначала мы познакомимся со стандартными средствами среды, а затем рассмотрим утилиты отладки из Android SDK.

16.1.1. Выбор конфигурации запуска

Команда меню Android Studio **Run | Run...** позволяет выбрать один из профилей запуска приложения. Можно запустить программу как Android-приложение, как Java-апплет, как Java-приложение и т. п. Все зависит от того, какую программу мы разрабатываем (рис. 16.1). Разумеется, приложение должно поддерживать такой тип запуска. Если ваше приложение, например, не является сервисом, то и запустить его как сервис вы не можете.

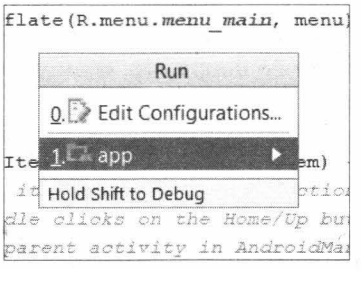


Рис. 16.1. Выбор конфигурации запуска

Для редактирования конфигураций запуска используется команда меню **Run | Edit Configurations**. В открывшемся окне вы можете добавить, отредактировать или удалить конфигурации запуска и отладки.

Конфигурацию запуска можно изменять отдельно для каждого проекта. Параметры каждой конфигурации задаются на четырех вкладках. Конфигурации запуска по-

зволяют запускать один и тот же проект при разных условиях. Так, вы можете создать одну конфигурацию, позволяющую запустить приложение в эмуляторе, а другую — на реальном устройстве.

Вкладка **General** (рис. 16.2) позволяет выбрать опции установки, запуска (например, можно выбрать активность, которая будет загружаться первой), а также цель. По умолчанию используется цель **Open Select Deployment Target Dialog** — откроется диалоговое окно выбора цели, в котором будут отображаться созданные эмуляторы и физические устройства. Вы же можете выбрать, где запускать приложение — на реальном устройстве или в эмуляторе, чтобы каждый раз не видеть это диалоговое окно.

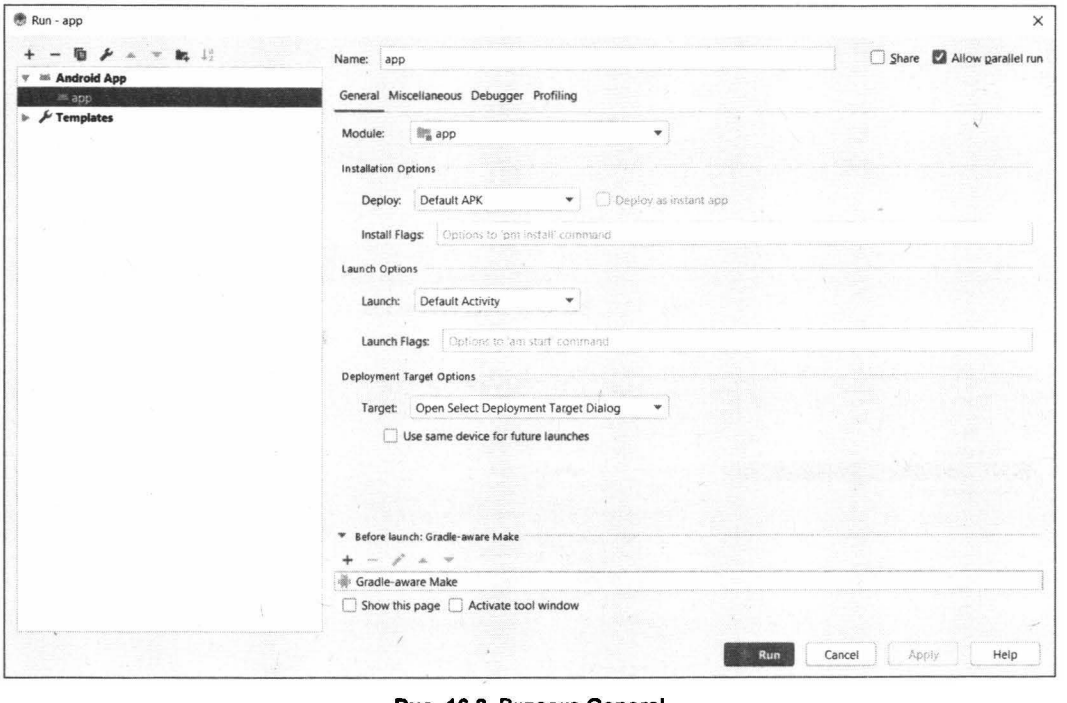


Рис. 16.2. Вкладка General

Вкладка **Miscellaneous** (рис. 16.3) позволяет установить несколько разных параметров. Например, **Clear log before launch** — очистка журнала перед запуском. Это весьма полезный параметр — чтобы при запуске вы видели только сообщения, относящиеся к этому запуску. Также можно открывать окно журнала автоматически — **Show logcat automatically**.

Вкладка **Debugger** содержит различные опции отладчика, которые вам вряд ли придется редактировать. А вот вкладка **Profiling** (рис. 16.4) позволяет включить расширенные опции профайлера — будут отслеживаться различные данные, такие как сетевая нагрузка, события приложения, счетчики объектов. Для этого уровень API должен быть меньше 26.

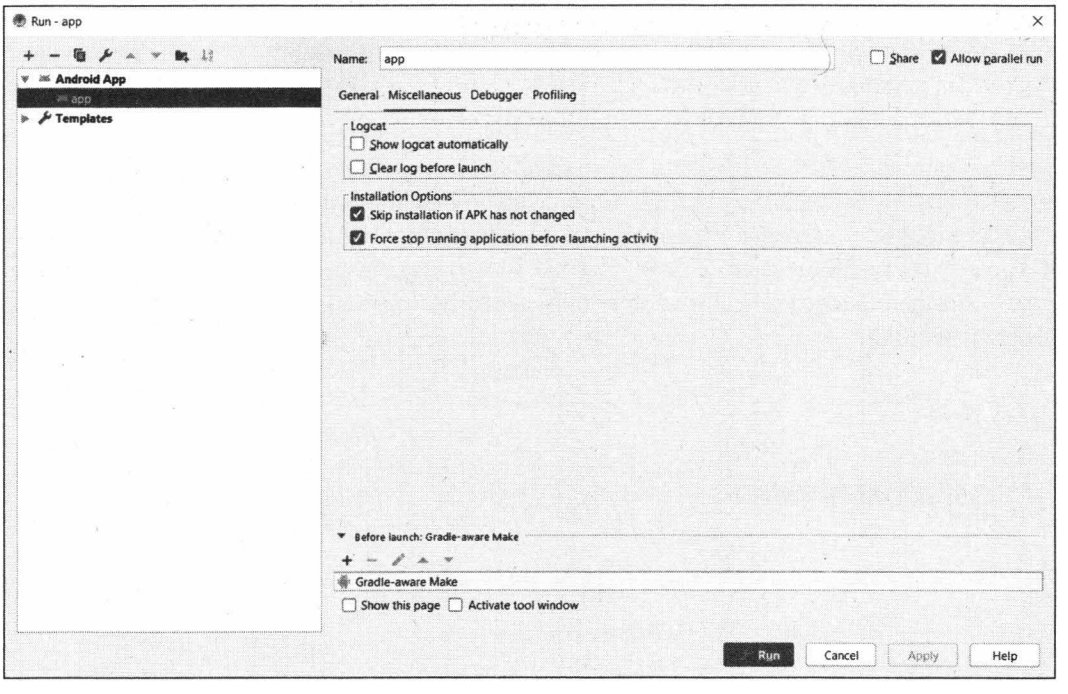


Рис. 16.3. Вкладка Miscellaneous

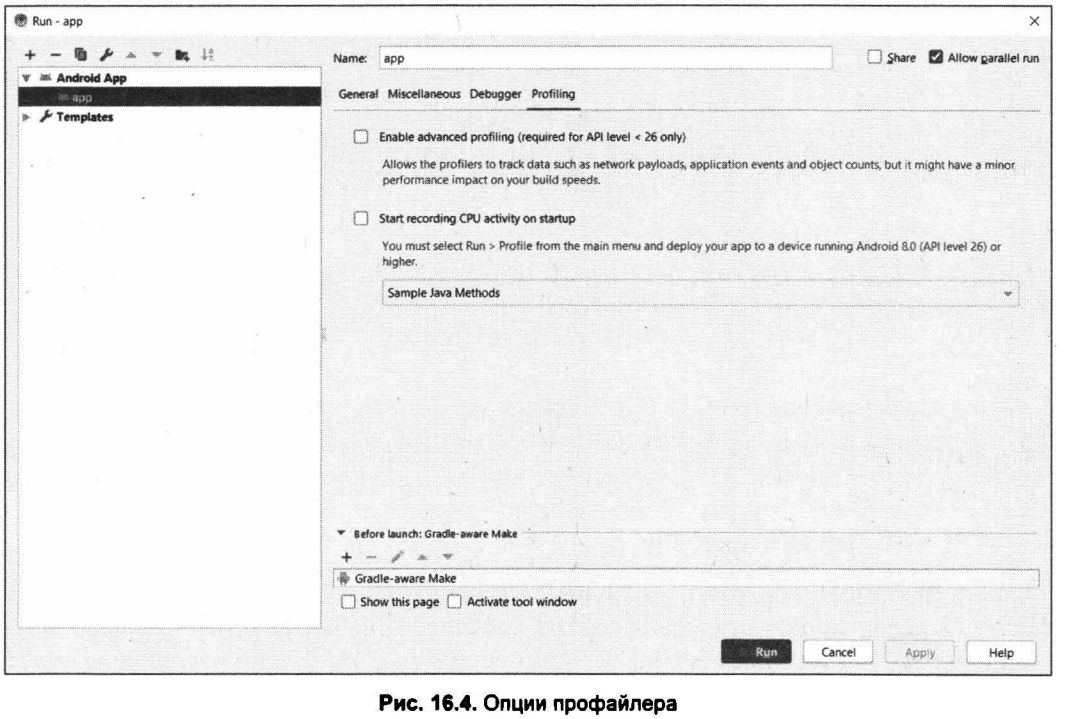


Рис. 16.4. Опции профайлера

16.1.2. Запуск процесса отладки

Для запуска процесса отладки используется команда **Run | Debug**. В коде вы можете установить точки останова (breakpoints). Выполнение программы будет приостановлено, как только она дойдет до очередной точки останова. Чтобы установить точку останова, щелкните один раз на поле слева от строки кода, которая будет использована как breakpoint. На рис. 16.5 показано, что установлена точка останова для 45-й строки кода файла MainActivity.java.

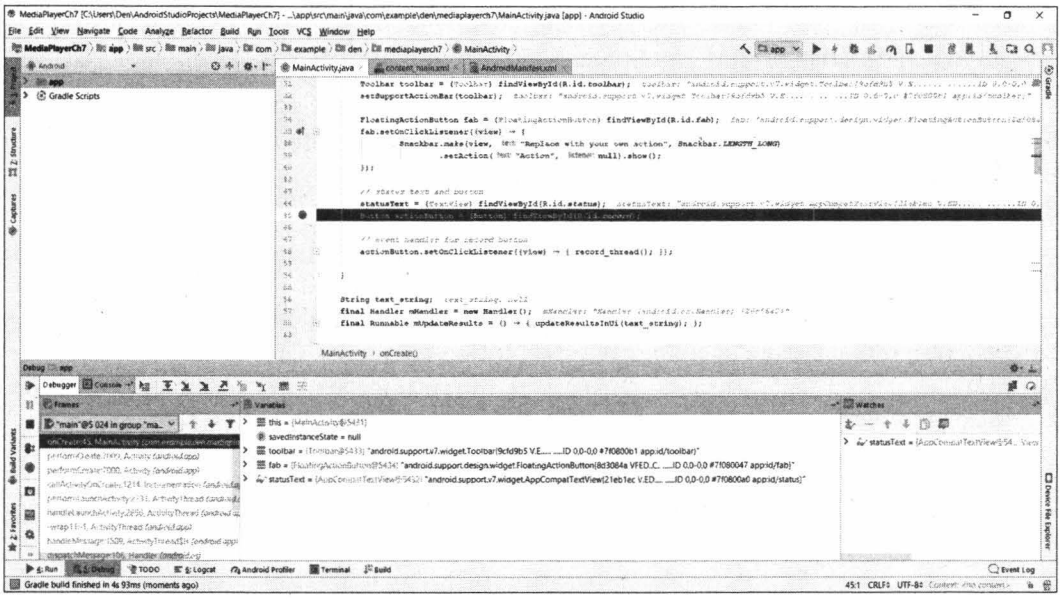


Рис. 16.5. Процесс отладки

Когда вы запустите процесс отладки, вам станет доступна вкладка **Debug** в нижней части окна. Вы можете добавить интересующую вас переменную в области **Variables** и **Watches**. В результате можно будет наблюдать за тем, как изменяется их значение.

Управлять дальнейшим выполнением программы можно или через меню **Run** (в нем станут доступны команды, относящиеся к отладчику), или через область **Debug**:

- **Resume Program** — продолжить выполнение (до следующей точки останова);
- **Pause Program** — приостановить выполнение программы;
- **Stop 'app'** — завершить выполнение программы;
- **Step Into** — когда выполнение программы дойдет до оператора вызова функции, операторы функции будут выполнены пошагово: по одному за одно нажатие клавиши <F5> (именно эта клавиша вызывает команду **Step Into**);
- **Step Over** — выполнение всей функции будет произведено за один шаг;

- **Run to Cursor** — запустить программу до достижения определенной строки кода. Как только выполнение достигнет заданной строки, оно будет приостановлено;
- **View Breakpoints** — показать все точки останова;
- **Mute Breakpoints** — отключить все точки останова (эта команда организована в виде кнопки на панели **Debug** — слева под кнопкой **View Breakpoints**).

Это не все команды, относящиеся к отладчику, но их будет вполне достаточно для организации всего процесса отладки.

16.1.3. Профайлинг

Профайлинг — это сбор характеристик работы программы: времени выполнения ее отдельных фрагментов, потребляемых ресурсов и т. п. Обычно профайлинг выполняется для оптимизации выполнения программы и поиска узких мест в ее производительности.

Ранее для профайлинга использовалась утилита Dalvik Debug Monitoring Service (DDMS). Однако сейчас вместо нее используется Android Profiler. Перейдите на соответствующую вкладку в нижней части окна Android Studio. Если у вас эта вкладка не отображается, выполните команду меню **View | Tool Windows | Android Profiler**. На рис. 16.6 показаны диаграммы потребляемых приложением ресурсов, которые вы можете получить с помощью **Android Profiler**.

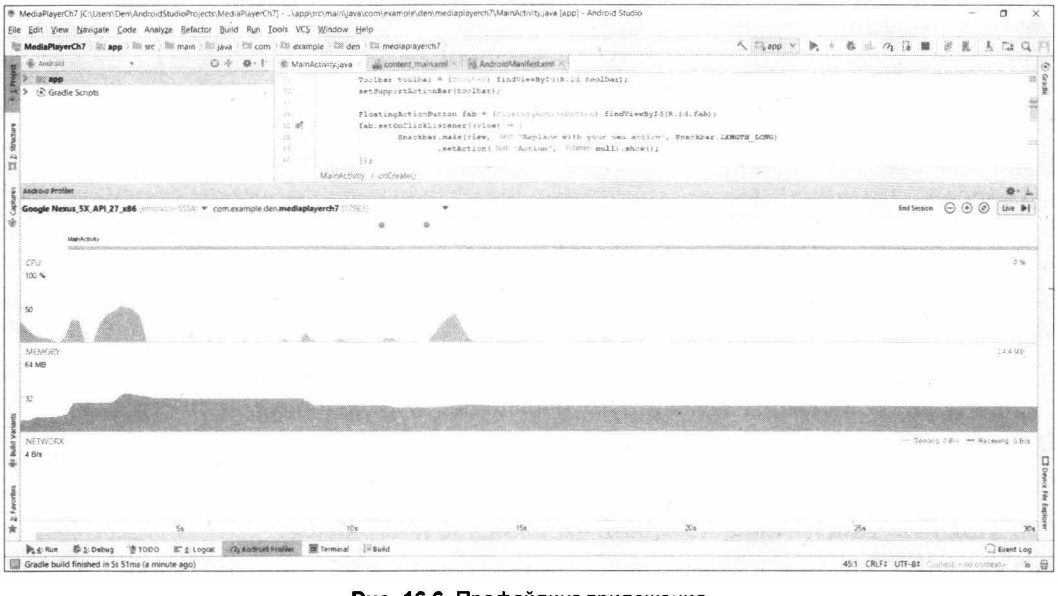


Рис. 16.6. Профайлинг приложения

Теперь рассмотрим практический пример. Взгляните еще раз на рис. 16.6 — на нем изображен всплеск потребления оперативной памяти и ресурсов процессора. Щелкните на пике — например, на пике потребления памяти. Откроется подробная

Рис. 16.7. Исследуем потребление памяти

информация, позволяющая узнать, какие объекты потребляли больше всего памяти (рис. 16.7).

16.1.4. Исследуем файловую систему устройства

Выполните команду меню **View | Tool Windows | Device File Explorer** — в правой части окна Android Studio появится область **Device File Explorer** (рис. 16.8), где вы

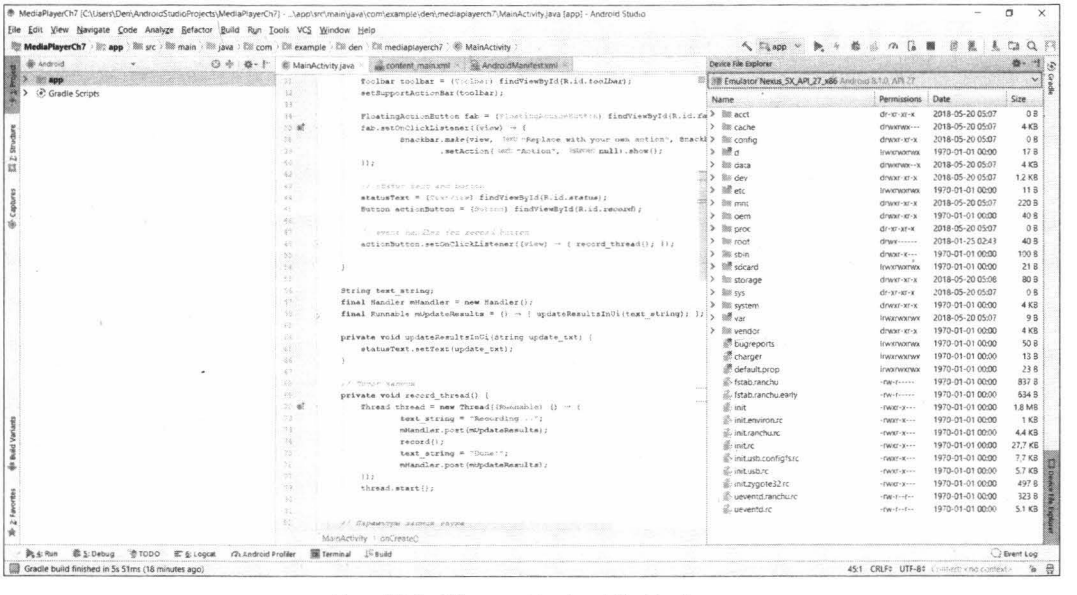


Рис. 16.8. Область **Device File Explorer**

сможете просмотреть файловую систему устройства, на котором сейчас выполняется ваше приложение: или эмулятора, или реального устройства.

16.1.5. Утилита LogCat

LogCat — утилита журналирования реального времени, доступная для операционной системы Android. Она собирает все сообщения системы и приложения, которые можно просмотреть и отфильтровать. Утилиту LogCat можно запускать как отдельно, с помощью команды `adb logcat`, так и воспользоваться вкладкой **LogCat**. Использовать вкладку **LogCat** намного удобнее — вы можете просматривать сообщения системы и приложения, не закрывая окна Android Studio (рис. 16.9).

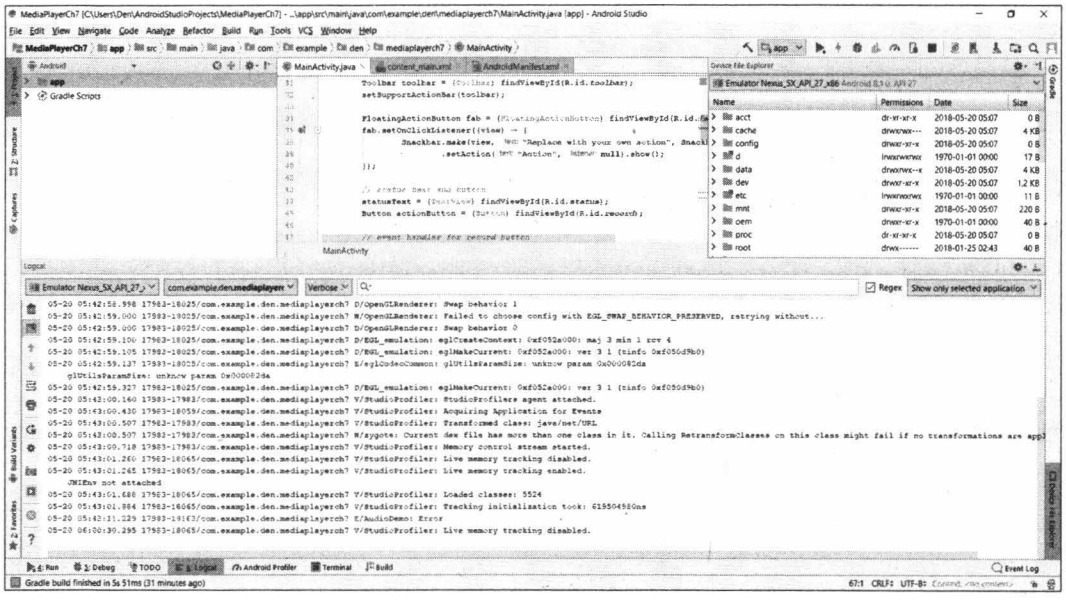


Рис. 16.9. Вкладка LogCat

Вкладка **LogCat** может также использоваться для создания скриншота программы. Нажмите кнопку **Screen Capture** (с изображением фотоаппарата) — вы увидите скриншот приложения (рис. 16.10). Вам останется нажать кнопку **Save** для его сохранения.

С помощью фильтра в верхнем правом углу вкладки **LogCat** вы можете отфильтровать сообщения определенного типа — например, выбрав из него **Error**, вы можете просмотреть только ошибки (рис. 16.11).

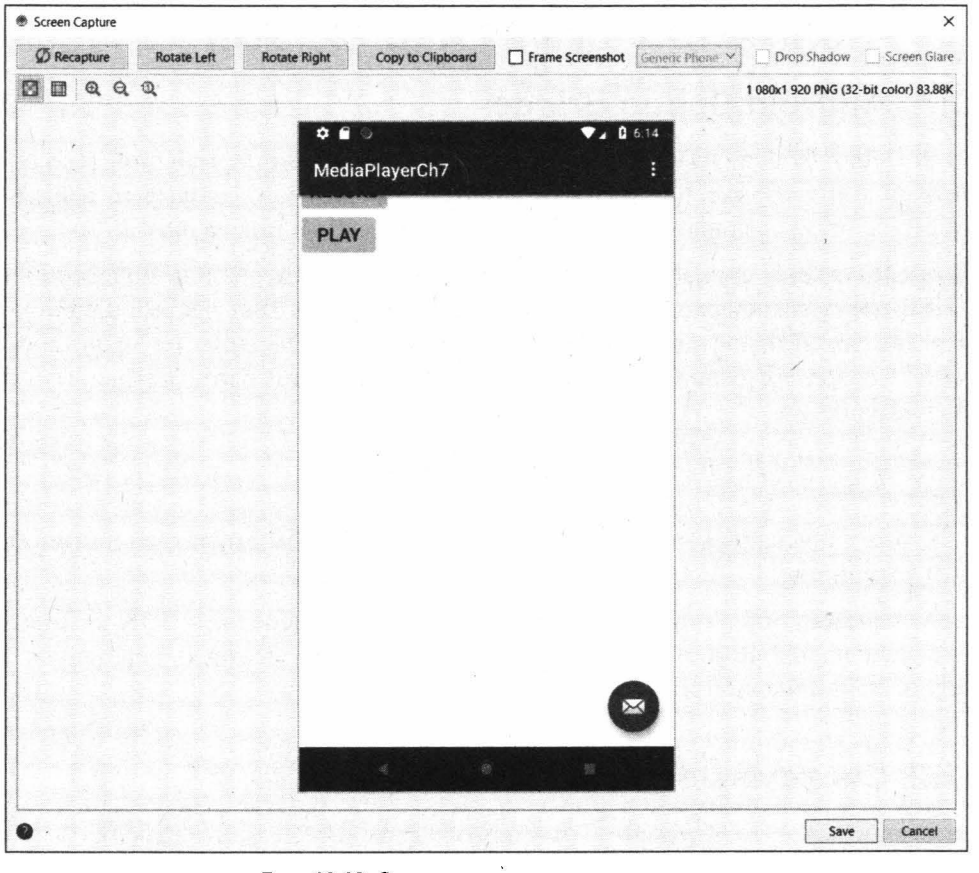


Рис. 16.10. Создание скриншота приложения

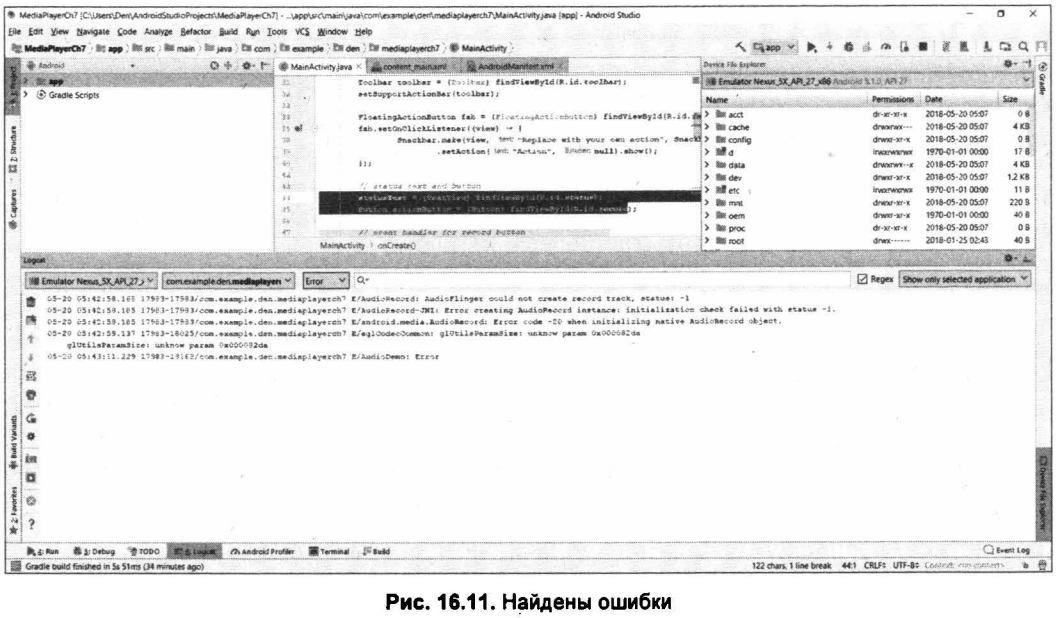


Рис. 16.11. Найдены ошибки

16.2. Утилиты отладки из Android SDK

16.2.1. Android Debug Bridge

Android Debug Bridge (ADB) — это очень полезная утилита командной строки. Исполнимый файл adb.exe находится в каталоге platform-tools. Запустите adb.exe без параметров, чтобы ознакомиться с ее возможностями (см. также *разд. 2.7*).

ADB предоставляет способ управления состоянием эмулятора или подсоединенного к USB-порту физического устройства. ADB состоит из трех частей: клиента, сервера и демона. Клиент инициализируется с помощью сценария на машине разработчика. Сервер запускается как фоновый процесс и может быть запущен или остановлен командами:

```
adb start-server
adb kill-server
```

Демон тоже запускается как фоновый процесс, но в самом эмуляторе или на физическом устройстве.

Обычно перезапускать сервер не нужно — это делает менеджер SDK автоматически в случае необходимости, но теперь вы знаете, как осуществить перезапуск ADB вручную, если понадобится.

16.2.2. Системные утилиты отладки

Важно понимать, что Android — это почти Linux. Вы можете выполнять любые стандартные Linux-утилиты как на виртуальном, так и на реальном устройстве. Для этого следует воспользоваться утилитой adb:

```
adb shell <команда оболочки Linux>
```

Например, в Linux для просмотра запущенных процессов служит команда top. Выполнить эту команду на текущем устройстве (виртуальном или реальном — без разницы) поможет следующая команда (рис. 16.12):

```
adb shell top
```

Можно просмотреть содержимое корневого каталога:

```
adb shell ls
```

Чтобы просмотреть содержимое другого каталога, нужно сначала в него перейти, а затем вызвать команду ls:

```
adb shell cd /etc; ls
```

Хотя намного удобнее просматривать файловую систему устройства с помощью области **Device File Explorer** (см. *разд. 16.1.4*).

Просмотреть распределение памяти можно с помощью команды dumpsys:

```
adb shell dumpsys meminfo <имя пакета>
```

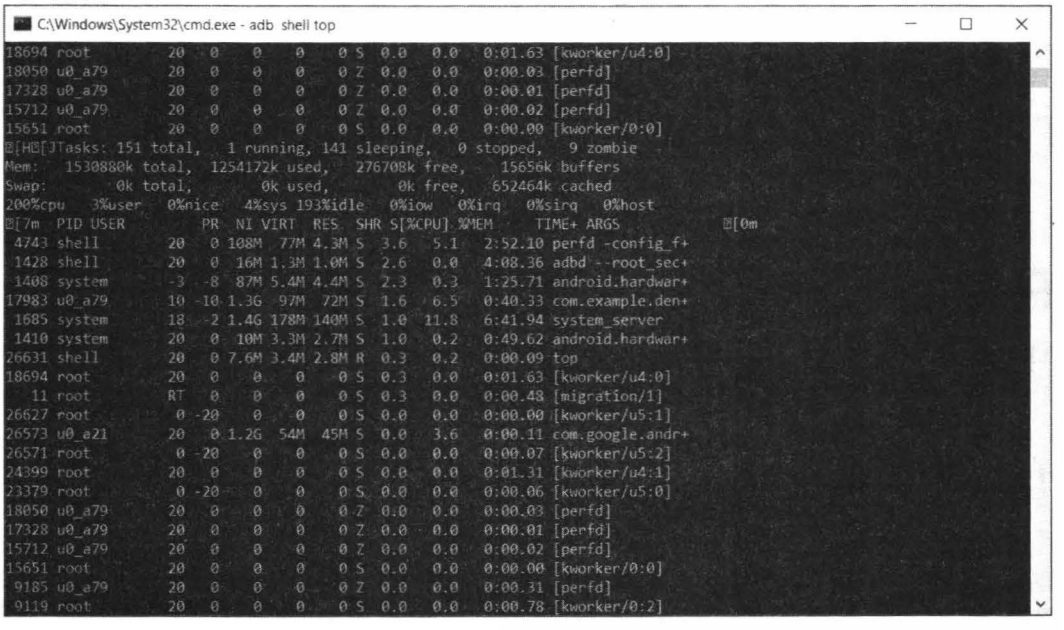


Рис. 16.12. Команда `adb shell top`

16.2.3. Отладчик *gdb* и Android-приложения

Если вы ранее программировали в Linux, то наверняка знакомы с отладчиком `gdb`. Некоторые программисты не представляют себе жизнь без этого отладчика.

В Android вы тоже можете установить этот отладчик. Для этого введите команды:

```
> adb shell
> adb /data/
> mkdir myfolder
> exit
> adb push gdbserver /data/myfolder
```

Теперь осталось запустить `gdb`:

```
> adb shell /data/myfolder/gdbserver 10.1.1.1:1234 myprogram
```

Здесь `10.1.1.1` — IP-адрес Android-устройства, а `1234` — номер порта. Последний аргумент — отлаживаемая программа.

Далее вы можете работать с `gdb` как обычно, используя его встроенные команды.

ГЛАВА 17

Распространение ваших программ через Google Play

17.1. Введение в Google Play

Создать хорошее приложение — мало. Нужно его еще распространить. Можно создать собственный сайт и заниматься раскруткой и сайта, и приложения. Такой подход исповедуют крупные компании с большим ассортиментом приложений и с бюджетом, позволяющим вложиться в серьезную рекламную кампанию для своего сайта. Ведь если о нем никто не узнает, то и дохода от созданных программ не будет.

Кроме рекламной кампании нужно будет решить еще ряд технических вопросов. Первый — это борьба с пиратством. Согласитесь, кто-то может единожды заплатить и скачать ваше приложение, а потом распространять его по всему Интернету бесплатно. Второй — механизм получения оплаты и загрузки приложения. Ну, и собственно разработка сайта. Если вы не являетесь веб-разработчиком, придется воспользоваться услугами профессионального веб-дизайнера. Для большой фирмы — это не проблема, в ее штате уже есть веб-разработчики. А для отдельного программиста создание такого сайта может вылиться в копеечку.

Небольшой компании или отдельному программисту гораздо выгоднее распространять свои Android-приложения через Google Play (**https://play.google.com/store/**). В результате решается множество проблем, которые Google Play берет на себя, а именно:

- распространение программы;
- обновление программы — уже купившие программу или просто загрузившие (для бесплатных программ) пользователи получат информацию о новой версии;
- защита от несанкционированного распространения;
- получение оплаты за программу.

Google Play — это хранилище и одновременно интернет-магазин Android-приложений. Приложения могут быть коммерческими (для их установки требуется оплата) или бесплатными (пользователь может скачать приложение бесплатно). В первом случае Google Play выступает в качестве посредника между разработчи-

ком и пользователем. Во втором случае Google Play представляет собой обычное хранилище, куда разработчики могут загрузить приложения, а пользователи — скачать.

Забегая вперед, скажу, что Google Play — не панацея. Существуют и другие сайты, помогающие распространять ваши приложения — например, сайт **http://androidapplications.com/**.

Есть у Google Play еще одна особенность. Далеко не всем дозволяется распространять, продавать и покупать приложения. Списки стран меняются, поэтому лучше время от времени просматривать сайт Google Play.

По следующей ссылке можно узнать список стран, в которых разрешена регистрация аккаунта разработчика: **https://support.google.com/googleplay/android-developer/answer/9306917?hl=ru**.

А вот страны, в которых можно распространять приложения через Google Play: **https://support.google.com/googleplay/android-developer/table/3541286?hl=ru.**

На этих страницах вы получите исчерпывающую информацию о географии Google Play. В настоящее время разработчики из России могут зарегистрировать как аккаунт разработчика, так и аккаунт продавца приложений. Валютой по умолчанию является американский доллар.

17.2. Правила размещения приложений на Google Play

При регистрации аккаунта разработчика вам придется принять условия соглашения Developer Distribution Agreement. В нем весьма много правил и всевозможной информации, поэтому мы остановимся только на самых важных его моментах.

- Вы должны полностью принять условия Соглашения. За нарушение правил Соглашения вас забанят и удалят ваши продукты из Google Play.
- Ничто не дается даром. Принимая условия Google Play, вы соглашаетесь, что Google будет удерживать 30% ваших доходов. Другими словами, если вы продаете программу стоимостью 10 долларов, то получите только 7 долларов (3 доллара будет удержано).
- Но 30% — это еще не все. Получая 7 долларов за каждое приложение, вы не должны забывать, что с них нужно заплатить налоги (их размер зависит от вашей страны). Я не силен в налоговом законодательстве, поэтому хороший бухгалтер или хотя бы консультация с налоговым инспектором вам не помешают.
- Если в течение 48 часов пользователь удалит приобретенное приложение (например, оно ему не понравилось), ему будут возвращены деньги.
- Разработчикам запрещается собирать в процессе использования приложения какие-либо личные данные пользователей, загрузивших (купивших) их приложение. Это правило можно изменить, если вы и пользователь заключите между собой специальное соглашение, но в этом случае вы обязаны явно (например, при

первом запуске) предупредить пользователя о том, что ваше приложение будет собирать и обрабатывать его персональные данные.

- Google не запрещает распространять ваши Android-приложения через другие каналы (например, через собственный сайт). Но если вы распространяете какое-либо приложение через Google Play, вам запрещено использовать другие каналы для продажи этого приложения.
- Старайтесь создавать качественные приложения. На страницах Google Play работает система рейтинга, и скачавшие вашу программу пользователи могут оценить ее по пятибалльной шкале. Если оценка будет слишком низкой, это отрицательно скажется на продажах приложения.
- Продавая свое приложение, вы предоставляете пользователю неэксклюзивное право использовать ваш продукт на его устройстве. Но если такого короткого соглашения мало, вы можете написать отдельное пользовательское соглашение (End User Licence Agreement, EULA).
- Вы не можете в названии приложения использовать слово «Android», также запрещено использовать логотип Android. Вместо «Android» нужно писать «for Android». Например, «Android WordProcessor» — это неправильное название. Правильное выглядит так: «WordProcessor for Android». Подробную информацию вы можете получить по адресу: **http://developer.android.com/distribute/tools/promote/brand.html**.

17.3. Теория и практика

Теоретически все просто: написали приложение, опубликовали на Google Play и получаете оплату за каждую загрузку, попивая коктейль на каких-то островах. Мечта фрилансера. Но на практике все не так радужно.

Действительно, вы можете опубликовать приложение на Google Play. Но, скорее всего, ваше приложение не уникально, и найдутся тысячи других почти аналогичных приложений. Уникальными сегодня могут быть только узкоспециализированные приложения, на которых также много денег не заработаешь, если не считать случаев, когда такие приложения создаются под определенного клиента, который готов расстаться с несколькими тысячами или даже десятками тысяч евро, — именно столько может стоить разработка приложения под заказ.

Чтобы ваше приложение было выбрано пользователями, оно должно быть популярнее, чем другие аналогичные приложения. Заметьте, не лучше, а именно популярнее. Ваше приложение может быть лучше всех аналогов, но его никто не купит, если о нем не будет всем известно.

Как Google Play определяет популярность приложения? Все просто: по количеству установок. Если у какого-то приложения миллион установок (на самом деле — это немного), а у вашего — всего тысяча, сами понимаете, оно окажется где-то в конце списка приложений его категории, — ведь между вашим и самым популярным (миллион!) будет еще множество других, более популярных приложений.

В итоге получается замкнутый круг — пока приложение не станет популярным, вы не сможете на нем заработать. А как же оно может стать популярным, если площадка, призванная распространять его, не способствует росту популярности? Это как с трудоустройством после университета — все хотят сотрудников с опытом работы, но где его взять, если без опыта на работу не берут?

Получается, что загрузить на Google Play приложение можно, но вот раскручивать его придется своими силами. А Google Play будет только принимать оплату за приложение. Вот некоторые рекомендации по раскрутке приложений:

- Расскажите о приложении на профильных форумах, на форумах, посвященных мобильным приложениям, там, где обсуждаются другие подобные программы, на всевозможных универсальных IT-форумах. Один из самых популярных ресурсов в нашей стране — **4pda.ru**.
- Используйте гостевые посты в блогах для размещения информации о своем приложении. Будьте готовы, что в большинстве случаев размещение гостевых постов — платное.
- Напишите статью и разместите ее на ресурсах, где обсуждаются мобильные приложения. Если не умеете красиво писать, попросите, и вам помогут — как правило, на таких ресурсах есть собственные авторы, принимающие заказы на статьи. Разумеется, все это (и написание статьи, и ее размещение) — не бесплатно.
- Создайте бесплатную версию своей программы с ограниченным функционалом — так пользователи смогут познакомиться с вашим приложением. Некоторые пользователи устанавливают только бесплатные программы, но если им понравится ваша программа, и они захотят получить расширенный функционал, они ее купят.
- Если у вас еще нет сайта для вашей программы, создайте его и занимайтесь его раскруткой. На сайте обязательно разместите ссылки на Google Play, чтобы пользователи могли скачать ваше приложение.
- Есть и не вполне честные методы раскрутки. Некоторые сервисы (не буду упоминать их в книге) предлагают за определенную плату обеспечить определенное количество установок вашего продукта. Если он бесплатный, то вы платите только за установку, а если платный, то нужно оплачивать установку + стоимость самого продукта. Один из самых дешевых сервисов просит 1 доллар США за одну установку, если она без отзыва. Если установка с отзывом — чуть более трех долларов. Вот и считайте сами, насколько это рентабельно. Чтобы набрать 1 миллион установок, нужно заплатить 1 миллион долларов. Целесообразность сомнительная.
- Используйте средства мониторинга ваших позиций в Google Play. В качестве примера можно привести **https://www.appannie.com**. Такие сервисы тоже платные, но они дают вам возможность оценить, насколько эффективным оказалось ваше то или иное решение. Приведу пример: вы разместили на каком-либо ресурсе статью за 500 долларов (гипотетическая стоимость размещения) и получили 1000 дополнительных (в добавок к обычному вашему уровню) установок

сразу после ее размещения и еще 400 на протяжении трех следующих дней. Получается, что стоимость одной установки вам обошлась в 35 центов, что выгоднее, чем оплачивать «искусственные» установки. А если приложение платное и стоит, например, 1 доллар, вы заработали 1400 × 0,7 = 980 долларов (без вычета налогов), т. е. полностью окупили размещение статьи. С другой стороны, если вы получили всего 200–300 установок, то можете на этом ресурсе статьи более не размещать, — это оказалось не выгодно. Поэтому средства мониторинга очень полезны.

17.4. Регистрация аккаунта разработчика

Чтобы зарегистрировать аккаунт разработчика Android, вам понадобятся 25 долларов (именно столько стоит аккаунт с правом публикации приложений) и аккаунт Google (его можно получить, зарегистрировавшись на **www.gmail.com**), — но он уже у вас, скорее всего, есть.

Весь процесс создания аккаунта разработчика состоит из трех этапов: создания аккаунта, оплаты взноса в 25 долларов и принятия соглашения о распространении ПО через Google Play. Соглашение можно прочитать прямо сейчас: **https://play.google.com/about/developer-distribution-agreement.html**.

Итак, представим, что Google-аккаунт и заветные 25 долларов у вас имеются. Перейдите по ссылке **https://play.google.com/apps/publish/signup/** для регистрации аккаунта разработчика.

Процесс регистрации предельно прост — вам нужно будет указать свое имя, телефон и сайт. Все эти поля являются обязательными для заполнения. После ввода личных данных нужно заплатить регистрационный взнос с помощью Google Checkout. Пока вы не заплатите 25 долларов, вы не получите доступ к консоли разработчика. Когда же взнос уплачен, открыть консоль управления можно по ссылке: **https://play.google.com/apps/publish/**.

Интерфейс консоли управления предельно прост — с ним без проблем разберется любой, кто освоил создание приложений для Android.

17.5. Подготовка приложения к продаже

17.5.1. Тестирования на разных устройствах

Обязательно убедитесь, что ваше приложение нормально работает на разных смартфонах. Для этого понадобится несколько устройств различных производителей, желательно с разной версией Android и разными разрешениями экрана. Где взять смартфоны? Один из вариантов — купить. Второй вариант — одолжить их у друзей. Одно только можно сказать: не стоит выкладывать ваше приложение на Google Play, если оно не протестировано хотя бы на 3–4 смартфонах разных производителей. Желательно также проверить его не только на смартфонах, но и на планшетах, поскольку они отличаются от смартфонов разрешением экрана.

17.5.2. Поддержка другого размера экрана

Не забудьте проверить, как выглядит ваше приложение на экране, размер которого отличается от размера экрана эмулятора по умолчанию. Здесь все намного проще: создайте несколько эмуляторов с разными размерами экрана и запустите приложение на каждом из них.

17.5.3. Локализация

Чтобы расширить аудиторию вашего приложения и, следовательно, больше заработать на нем, нужно выполнить локализацию, т. е. перевод приложения на разные языки.

С технической точки зрения, выполнить локализацию достаточно просто. Создайте в каталоге res несколько каталогов values-*: values-en — для английского языка, values-de — для немецкого и т. д. Затем скопируйте в эти каталоги оригинальный файл strings.xml из res/values. Ясно, что локализацией нужно заниматься, когда приложение уже создано, все ошибки исправлены, и больше не планируется добавлять в файл strings.xml новые строковые ресурсы.

Затем в настройки приложения добавляете параметр **Язык** (Language), позволяющий выбрать язык. Выбранный язык будет сохраняться с помощью предпочтений. При запуске приложения оно проверяет предпочтение и загружает соответствующие строковые ресурсы.

Кроме строковых ресурсов не забудьте также локализировать изображения (из каталогов res/drawable-*) — если они содержат надписи, и меню приложения.

Вот только, чтобы не опростоволоситься, привлеките профессионального переводчика, который и выполнит перевод файлов strings.xml для каждого языка. Возможно, понадобится несколько переводчиков.

17.5.4. Значок приложения

Весьма часто начинающие разработчики забывают изменить значок приложения. Согласитесь, коммерческое приложение за 5–10 долларов со значком по умолчанию выглядит, по меньшей мере, смешно.

Изменить значок приложения очень просто. Подготовьте файл значка и поместите его в каталог res/drawable. Затем измените атрибут `android:icon` элемента `<application>` в файле манифеста:

```
<application anroid:icon="@drawable/my_icon" ...
```

17.5.5. Подготовка APK-файла к загрузке

Прежде чем вы опубликуете свой APK-файл на Google Play, его нужно подписать, т. е. создать для вашего приложения сертификат. Проще всего это сделать с помощью утилиты `keytool`, которая находится в каталоге C:\Program Files\Java\jdk1.X.X_XX\bin (в подкаталоге bin каталога, в который вы установили JDK):

```
keytool -genkey -v -keystore c:\my_keys\app.keystore -alias my_android_app -
storepass 123456 -keypass 111456 -keyalg RSA -validity 9125
```

Параметры утилиты `keytool` приведены в табл. 17.1 в порядке их указания в представленной команде.

Таблица 17.1. *Параметры утилиты* `keytool`

Параметр	Описание
`genkey`	Генерирует пару ключей: публичный и приватный
`v`	Выводит подробные сообщения (обычно этот параметр можно опустить)
`keystore`	Путь к хранилищу ключей — в нашем случае мы используем файл c:\keys\app.keystore
`alias`	Псевдоним в базе данных ключей. Чтобы вам было понятнее, используйте в качестве псевдонима название приложения (без пробелов, разумеется)
`storepass`	Пароль для хранилища
`keypass`	Пароль для приватного ключа. Укажите более сложные пароли, чем мои. Пароли для хранилища и приватного ключа могут совпадать, но лучше, когда они разные
`keyalg`	Алгоритм ключа — обычно используется RSA
`validity`	Период проверки в днях: в нашем случае 9125 дней, т. е. 25 лет. Двадцать пять лет — это не много. Вы же можете создать сертификат на 50 лет — этого более чем достаточно

Утилита `keytool` задаст ряд вопросов, а затем вы увидите сообщение о том, что сертификат сгенерирован:

```
Generating 1,024 bit RSA key pair and self-signed certificate (SHA1withRSA)
with a validity of 9125 days
        for: CN=Den, OU=Dev, O=Home Ltd, L=Russia, ST=Unknown, C=RU
[Storing c:\ keys\app.keystore]
```

Теперь у вас есть цифровой сертификат, которым вы можете подписать ваш APK-файл. Для этого вам нужно использовать утилиту `jarsigner`:

```
jarsigner -keystore c:\keys\app.keystore -storepass 123456
-keypass 111456 путь_к_apk_файлу my_android_app
```

Давайте разберемся, что здесь есть что. Параметр `keystore` указывает путь к хранилищу ключей. Мы договорились, что будем использовать файл c:\keys\app.keystore. Затем указываются пароли к хранилищу и к приватному ключу. Следующий параметр — это путь к APK-файлу вашего приложения, последний параметр — псевдоним, указанный при создании ключа.

Уже подписанный APK-файл лучше всего передать программе `zipalign` для приведения несжатых данных в соответствие с границами памяти, что позволит оптимизировать программу — она будет быстрее выполняться. У процессоров Android-

устройств четырехбайтная граница памяти, поэтому команда `zipalign` будет выглядеть так:

```
zipalign -v 4 signed.apk optimized.apk
```

Как вы уже догадались, `signed.apk` — это подписанный APK-файл, а `optimized.apk` — это имя нового, оптимизированного, APK-файла, который полностью готов для публикации на Google Play.

Но загружать файл пока рано. Попробуйте его установить вручную в эмулятор или физическое устройство. Для этого или запустите эмулятор, или подключите Android-устройство к компьютеру, после чего введите команду:

```
adb install <имя_apk_файла>
```

Запустите и протестируйте приложение. Затем попытайтесь удалить его:

```
adb uninstall <имя_пакета>
```

Например:

```
adb uninstall com.samples.my_app
```

Если все прошло успешно, тогда перейдите в консоль разработчика — вы готовы загрузить свое приложение на Google Play.

ГЛАВА 18

Эмулятор Genymotion

18.1. Основные сведения о Genymotion

В качестве бонуса для уважаемых читателей мы рассмотрим здесь очень популярный эмулятор Genymotion. Каковы его преимущества по сравнению с обычным эмулятором среды Android? Если вкратце: скорость и комфорт.

Эмулятор Genymotion является оболочкой для VirtualBox, а виртуальные устройства — это не что иное, как гостевые машины с Linux/Android. По сути, VirtualBox выполняет в окне Genymotion гостевую операционную систему. В этом есть и преимущества, и недостатки. К преимуществам можно отнести скорость. Эмулятор Genymotion работает очень быстро, гораздо быстрее стандартного эмулятора. При желании вы можете сравнить производительность какого-нибудь приложения на физическом устройстве и на этом эмуляторе. Особой разницы вы не заметите — если у вас, конечно, мощный компьютер.

К недостаткам решения виртуализации VirtualBox можно отнести сложность настройки. Нет, в идеальном случае все будет просто. Надеюсь, что ваш случай как раз такой: установили Genymotion, добавили устройство и запустили его. В противном случае вам придется побороться за право использования VirtualBox — настраивать его можно достаточно долго. Описывать какие-либо примеры ошибок не стану — их весьма много, и есть вероятность, что даже если в книге будут описаны 10 самых популярных, у вас возникнет одиннадцатая. Подробные инструкции по устранению тех или иных ошибок VirtualBox есть в Интернете, и с помощью Google не составит труда их найти.

Теперь о комфорте. Эмулятор Genymotion гораздо удобнее. Хотите установить APK-файл? Просто перетащите его в окно эмулятора. Хотите изменить прошивку виртуального устройства? Просто перетащите ее в окно эмулятора. Нужно передать в эмулятор файлы? Вы уже догадались, что нужно сделать? Никакой консоли, никакого `adb` (хотя использовать `adb` можно). Я уже молчу о возможности копирования текста между физическим компьютером и эмулятором: просто скопировали текст, скажем, из Word и вставили его в приложение, запущенное в эмуляторе. За это многим и нравится Genymotion. Этот эмулятор может быть полезен не только

программистам, но и обычным пользователям — например, чтобы играть в любимые Android-игры за обычным компьютером.

С другой стороны, не всегда получается заставить его работать, во всяком случае быстро. Помню, как я потратил на его настройку два дня: то не запускалась виртуальная машина, то не устанавливались сервисы Google, то устанавливались, но отказывались запускаться. Тем не менее я настроил все, что мне было нужно, однако потратил много времени.

Как раз для таких случаев с сайта **genymotion.com** можно скачать 30-дневную версию и попробовать эмулятор в действии. Если он вам понравится, придется выложить 136 долларов в год — именно столько стоит лицензия для индивидуальных разработчиков.

Ранее к преимуществам этого эмулятора над стандартным можно было отнести поддержку Google Play. Однако сегодня штатный эмулятор Android Studio также поддерживает Google Play, поэтому такая особенность теперь не может быть первоочередной.

> ***Примечание***
>
> С сайта разработчиков эмулятор Genymotion можно скачать как совместно с VirtualBox, так и без него. Если VirtualBox еще у вас не установлен, то вам нужно выбрать версию с ним — полагаю, это очевидно. Если же VirtualBox у вас установлен, можно попробовать установить Genymotion без него. Если не получится «подружить» вашу версию VirtualBox с этим эмулятором, то придется VirtualBox удалить и установить Genymotion вместе с ним.

18.2. Создание виртуального устройства

Запущенный эмулятор Genymotion показан на рис. 18.1. В верхней области окна эмулятора выводятся уже установленные в эмулятор шаблоны устройств. Как можно здесь видеть, пока никакие шаблоны не установлены, т. е. новых устройств в эмулятор еще не добавлено. В нижней области окна эмулятора выводятся доступные шаблоны. Выберите нужный шаблон, щелкните на нем правой кнопкой мыши и выберите команду **Install**.

Можно выбрать какое-то конкретное устройство — например, какой-либо **Nexus**, а можно и **Custom Phone**, если важна только версия Android. На рис. 18.2 добавляется шаблон, содержащий 9-ю версию Android.

После выбора типа устройства (планшет или смартфон) и версии Android эмулятор отображает сводку по будущему устройству. Обратите внимание на размер оперативной памяти (**Memory Size**) — если двух гигабайт много для вашего компьютера, то нужно в настройках эмулятора уменьшить этот размер.

Каждый шаблон «весит» несколько сотен мегабайт, поэтому, если соединение с Интернетом не очень быстрое, придется немного подождать (рис. 18.3).

Когда образ будет загружен, добавленное вами устройство появится в списке в верхней области окна эмулятора (рис. 18.4).

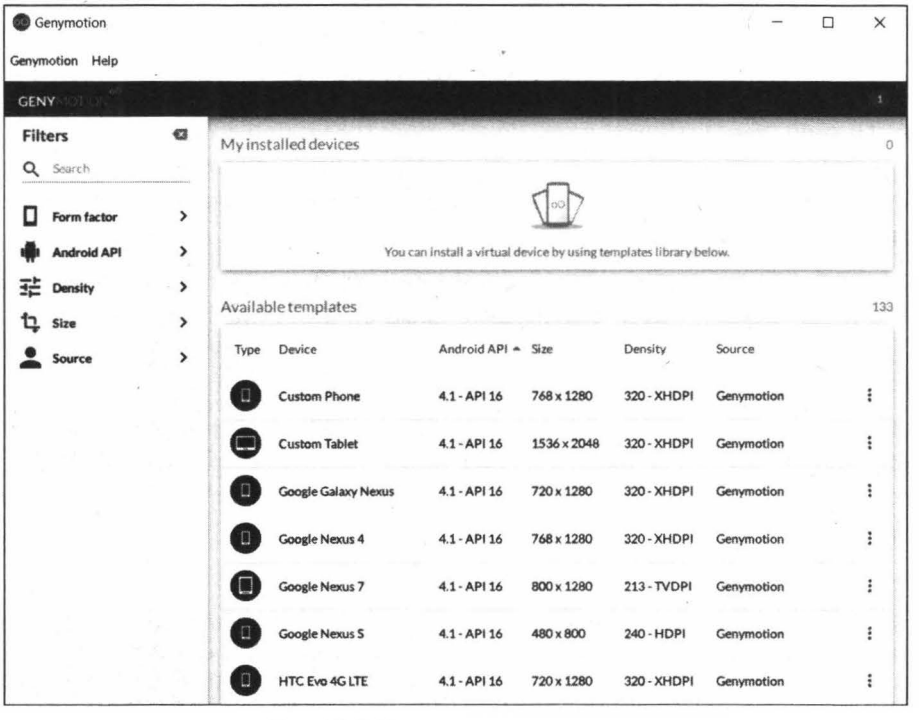


Рис. 18.1. Основное окно эмулятора

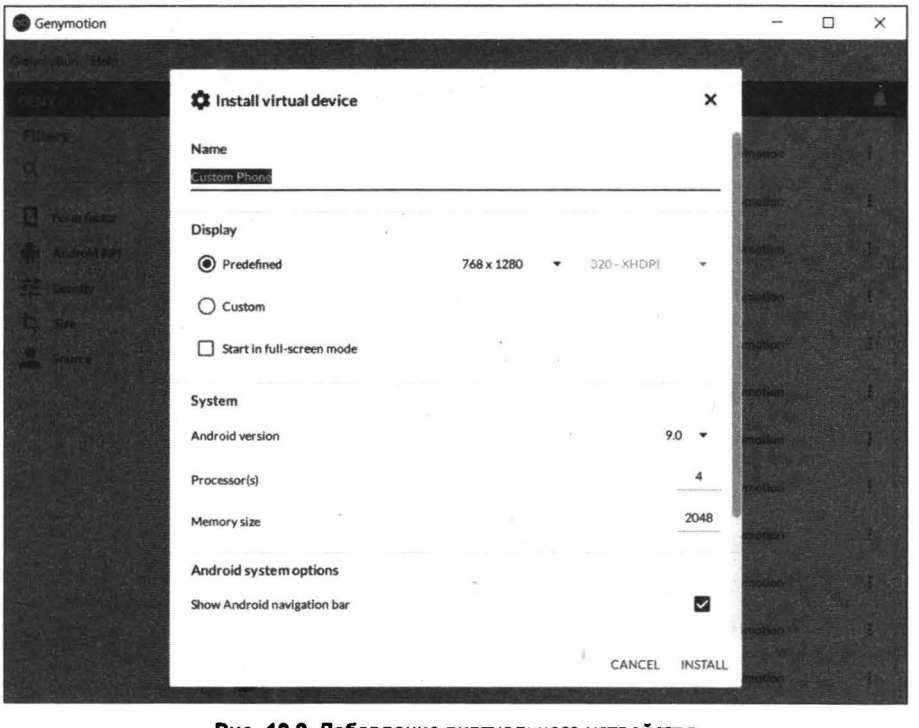


Рис. 18.2. Добавление виртуального устройства

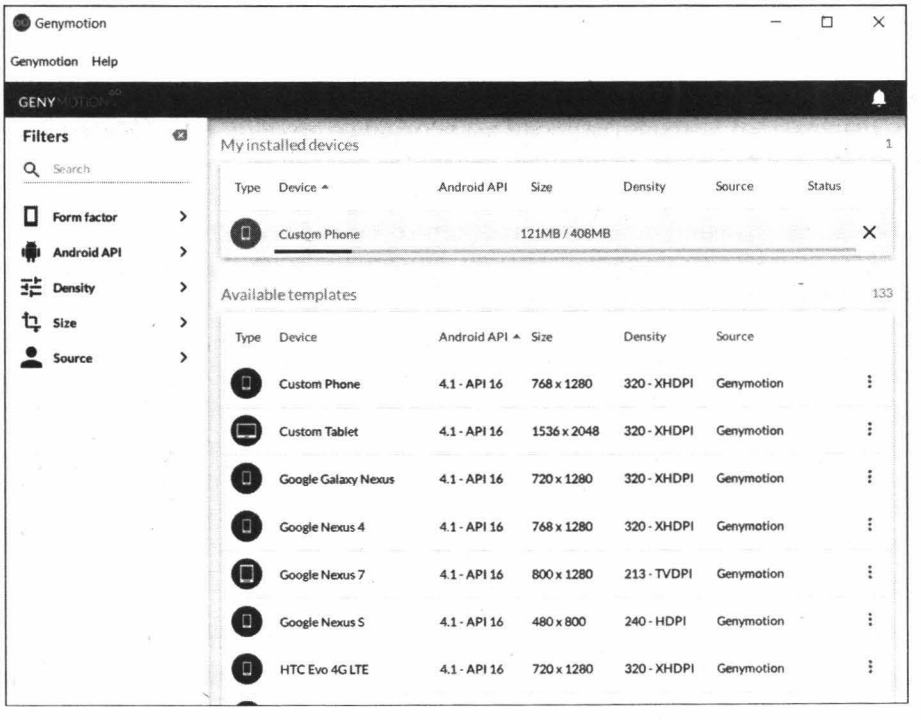


Рис. 18.3. Идет загрузка шаблона

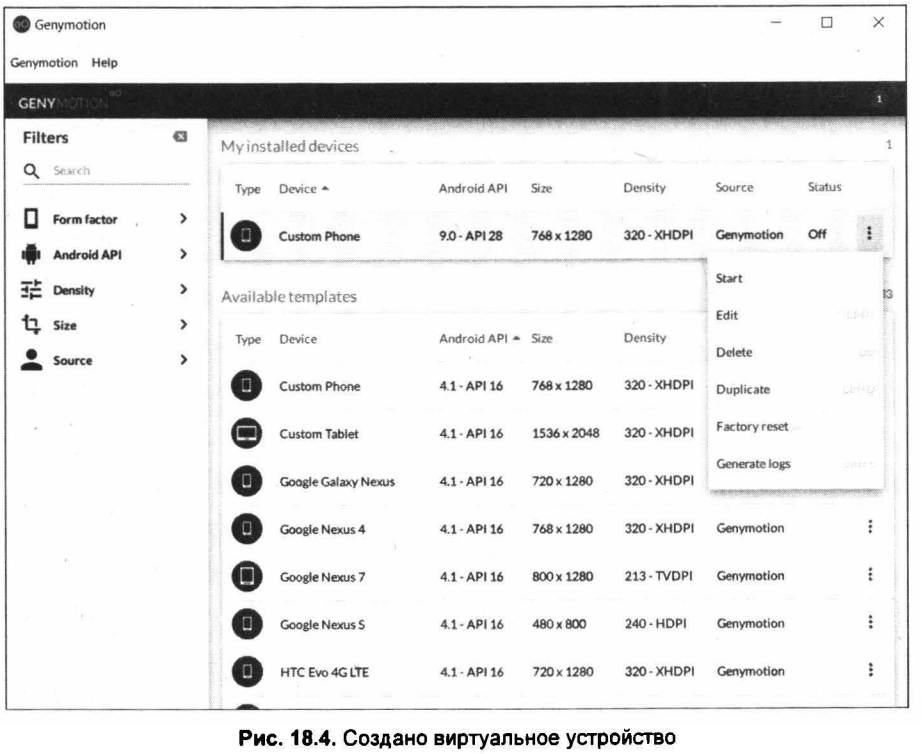


Рис. 18.4. Создано виртуальное устройство

Не спешите запускать виртуальное устройство. Выберите команду **Edit**, установите размер оперативной памяти и выберите другое разрешение. Сначала о памяти. Дело в том, что по умолчанию Genymotion предлагает установить 2 Гбайт оперативной памяти для каждого устройства. Если вы собрались эмулировать версию 9.0, то это вполне нормально, а если, например, версию 4.4, то двух гигабайт для нее много. Реальные устройства с такой версией Android, как правило, были оснащены 512–1024 Мбайт оперативной памяти. Устройства с версией 6.0–7.0 — от 1 до 1,5 Гбайт. Так что вы можете не только сэкономить память, но и протестировать работу приложения в условиях, близких к реальным.

Что же касается разрешения, то для смартфонов (**Custom Phone**) предлагается разрешение 768×1280 точек. Здесь имеется в виду, что 1280 точек — это по высоте. Плюс обрамление окна (изображение смартфона и окна Windows). Если у вас разрешение монитора по вертикали больше 1280 пикселов, можете разрешение не изменять, в противном случае эмулятор постарается подогнать выбранное разрешение под размер вашего монитора, и качество картинки в самом эмуляторе будет хуже, что очень критично при записи видео или создании снимков экрана.

* * *

Эмулятор Genymotion — продукт интересный и неоднозначный. Он удобнее обычного эмулятора, и некоторые пользователи выбирают Genymotion, чтобы играть в игры для Android на обычном ПК. При желании установить игру можно и в штатном эмуляторе, но для этого нужны дополнительные знания — как минимум, нужно знать о существовании Android Studio и о том, как запустить в ней эмулятор. С другой стороны, Genymotion — платный. Использовать его или нет, решать только вам.

Вместо заключения

Разработка приложений для Android — процесс весьма интересный и перспективный. Интересный — хотя бы потому, что пишешь программу не для компьютера (чем сейчас никого не удивишь), а для мобильного устройства. Перспективный — потому, что разработчиков для мобильных устройств на наших просторах не так уж и много.

Ясное дело, что в одной книге полностью не раскроешь всех аспектов программирования для Android. Но книг на русском языке откровенно мало. Дополнительную информацию можно получить в книге А. Голощапова «Google Android: программирование для мобильных устройств»: **http://www.bhv.ru/books/book.php?id=187679**.

Связаться со мной и задать вопросы, если таковые появятся в ходе чтения книги, можно здесь: **https://www.dkws.org.ua**.

Специально для разработчиков приложений для Android я создал на своем форуме отдельный раздел, где вы найдете много интересного:
https://www.dkws.org.ua/phpbb2/viewforum.php?f=60.

Предметный указатель

А

В

Г

Д

К

М

О

П

Р

С

Т

У

Ф

Ш

Э

самоучитель

Программирование для Android

Эффективная разработка Android-приложений

Колисниченко Денис Николаевич, инженер-программист и системный администратор. Автор более 50 книг компьютерной тематики, в том числе «Android для пользователя. Полезные программы и советы», «PHP и MySQL. Разработка веб-приложений», «Самоучитель системного администратора» и др. Сайт автора: https://dkws.org.ua.

Рассмотрены основные этапы разработки приложений для мобильной платформы Android: установка необходимого программного обеспечения, использование эмулятора Android, создание интерфейса пользователя, работа с графикой, воспроизведение звука и видео, методы хранения данных (в том числе СУБД SQLite), взаимодействие с аппаратными средствами смартфона, отладка приложений и их публикация в магазине Google Play.

Особое внимание уделено взаимодействию с аппаратными средствами смартфона. Показано, как получить информацию об устройстве и определить его состояние, использовать его датчики (акселерометр, датчик света, датчик температуры, датчик давления), камеру, Bluetooth-адаптер.

Приведены решения для различных нештатных ситуаций (отказ эмулятора, проблема с установкой программного обеспечения и т. д.), что поможет начинающему программисту.

ISBN 978-5-9775-6587-5

191036, Санкт-Петербург,
Гончарная ул., 20
Тел.: (812) 717-10-50,
339-54-17, 339-54-28
E-mail: mail@bhv.ru
Internet: www.bhv.ru

bhv®