

Модуль 1. Основные возможности мобильных платформ

1.1. Основы построения приложения

Приложение GeoQuiz состоит из *активности* (activity) и *макета* (layout):

- ① Активность представлена экземпляром **Activity** — класса из Android. Она отвечает за взаимодействие пользователя с информацией на экране.

Чтобы реализовать функциональность, необходимую приложению, разработчик пишет subclasses **Activity**. В простом приложении бывает достаточно одного subclasses; в сложном приложении их может потребоваться несколько.

GeoQuiz — простое приложение, поэтому в нем используется всего один subclasses **Activity** с именем **QuizActivity**. Класс **QuizActivity** управляет пользовательским интерфейсом, изображенным на рис. 1.1.

- ① Макет определяет набор объектов пользовательского интерфейса и их расположение на экране. Макет формируется из определений, написанных на языке XML. Каждое определение используется для создания объекта, выводимого на экране (например, кнопки или текста).

Приложение GeoQuiz включает файл макета с именем **activity_quiz.xml**. Разметка XML в этом файле определяет пользовательский интерфейс, изображенный на рис. 1.1.

Отношения между **QuizActivity** и **activity_quiz.xml** изображены на рис. 1.2.

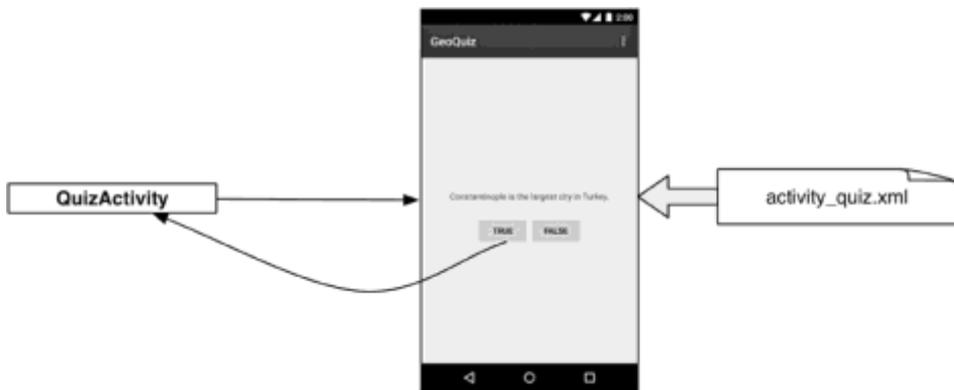


Рис. 1.2. **QuizActivity** управляет интерфейсом, определяемым в файле **activity_quiz.xml**

Учитывая все сказанное, давайте построим приложение.

Создание проекта Android

Работа начинается с создания *проекта Android*. Проект Android содержит файлы, из которых состоит приложение. Чтобы создать новый проект, откройте Android Studio.

Если Android Studio запускается на вашем компьютере впервые, на экране появляется диалоговое окно с приветствием (рис. 1.3).



Рис. 1.3. Добро пожаловать в Android Studio

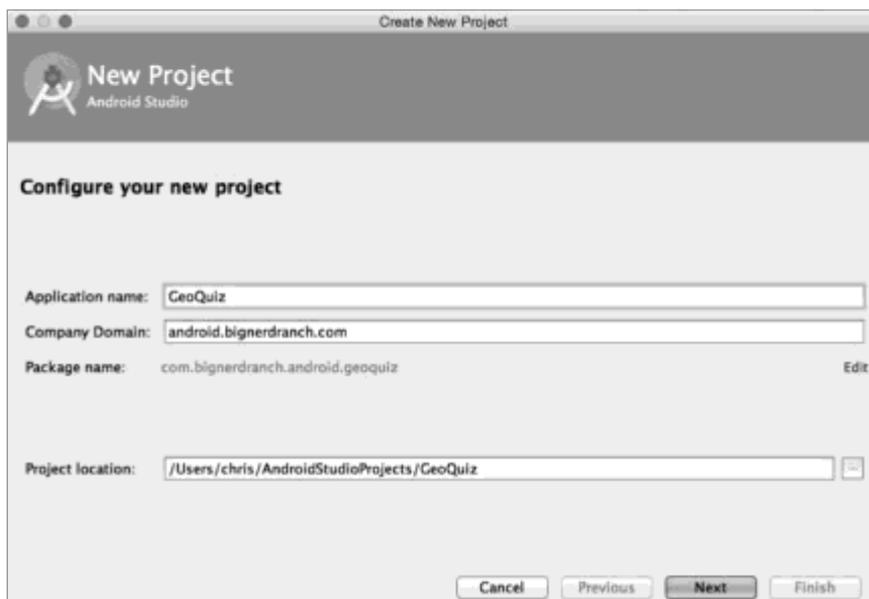


Рис. 1.4. Создание нового проекта

Выберите в диалоговом окне команду Start a new Android Studio project. Если диалоговое окно не отображается при запуске, значит, ранее вы уже создавали другие проекты. В таком случае выполните команду File ▶ New Project....

Открывается мастер создания проекта. На первом экране мастера введите имя приложения GeoQuiz (рис. 1.4). В поле Company Domain введите строку *android.bignerdranch.com*; сгенерированное имя пакета (Package Name) при этом автоматически меняется на *com.bignerdranch.android.geoquiz*. В поле Project location введите любую папку своей файловой системы на свое усмотрение.

Обратите внимание: в имени пакета используется схема «обратного DNS», согласно которой доменное имя вашей организации записывается в обратном порядке с присоединением суффиксов дополнительных идентификаторов. Эта схема обеспечивает уникальность имен пакетов и позволяет различать приложения на устройстве и в Google Play.

Щелкните на кнопке Next. На следующем экране можно ввести дополнительную информацию об устройствах, которые вы намерены поддерживать. Приложение GeoQuiz будет поддерживать только телефоны, поэтому установите только флажок Phone and Tablet. Выберите в списке минимальную версию SDK API 16: Android 4.1 (Jelly Bean) (рис. 1.5). Разные версии Android более подробно рассматриваются в главе 6.



Рис. 1.5. Определение поддерживаемых устройств

Инструментарий Android обновляется несколько раз в год, поэтому окно мастера на вашем компьютере может несколько отличаться от моего. Обычно это не создает проблем; принимаемые решения остаются практически неизменными. (Если ваше окно не имеет ничего общего с изображенным, значит, инструментарий изменился более радикально. Без паники. Загляните на форум книги *forums.bignerdranch.com*, и мы поможем вам найти обновленную версию.)

Щелкните на кнопке Next.

На следующем экране вам будет предложено выбрать шаблон первого экрана GeoQuiz (рис. 1.6). Выберите пустую активность (Blank Activity) и щелкните на кнопке Next.



Рис. 1.6. Выбор типа активности

В последнем диалоговом окне мастера введите имя subclasses активности `QuizActivity` (рис. 1.7). Обратите внимание на суффикс **Activity** в имени класса. Его присутствие не обязательно, но это очень полезное соглашение, которое стоит соблюдать.

Имя макета автоматически заменяется на `activity_quiz` в соответствии с переименованием активности. Имя макета записывается в порядке, обратном имени активности; в нем используются символы нижнего регистра, а слова разделяются символами подчеркивания. Эту схему формирования имен рекомендуется применять как для макетов, так и для других ресурсов, о которых вы узнаете позднее.

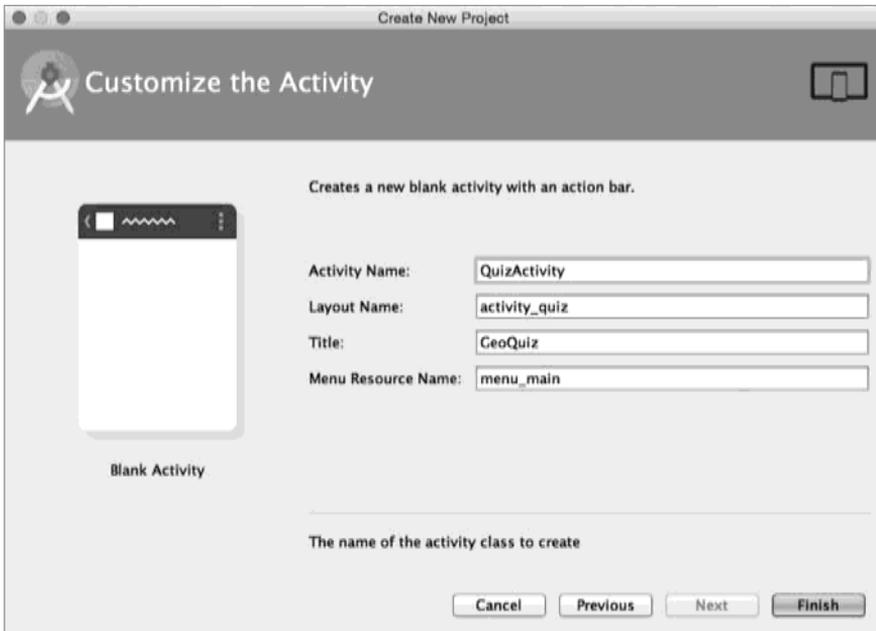


Рис. 1.7. Настройка новой активности

В поле Title введите строку *GeoQuiz* (имя приложения). Оставьте содержимое поля Menu Resource Name без изменений и щелкните на кнопке Finish. Среда Android Studio создает и открывает новый проект.

Навигация в Android Studio

Среда Android Studio открывает ваш проект в окне, изображенном на рис. 1.8.

Различные части окна проекта называются *панелями*.

Слева располагается *окно инструментов Project*. В нем выполняются операции с файлами, относящимися к вашему проекту.

В середине находится панель *редактора*. Чтобы вам было проще приступить к работе, Android Studio открывает в редакторе файл `activity_quiz.xml`. (Если в редакторе выводится изображение, щелкните на вкладке Text в нижней части панели.) На панели в правой части окна показано, как выглядит содержимое файла в режиме предварительного просмотра.

Видимостью различных панелей можно управлять, щелкая на их именах на полосе кнопок инструментов у левого, правого или нижнего края экрана. Также для многих панелей определены специальные комбинации клавиш. Если полосы с кнопками не отображаются, щелкните на серой квадратной кнопке в левом нижнем углу главного окна или выполните команду View ▶ Tool Buttons.



Рис. 1.8. Окно нового проекта

Построение макета пользовательского интерфейса

В настоящее время файл activity_quiz.xml определяет разметку для активности по умолчанию. Такая разметка часто изменяется, но XML будет выглядеть примерно так, как показано в листинге 1.1.

Листинг 1.1. Разметка для активности

```
<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"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".QuizActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

Макет активности по умолчанию определяет два *виджета* (widgets): RelativeLayout и TextView.

Виджеты представляют собой структурные элементы, из которых составляется пользовательский интерфейс. Виджет может выводить текст или графику, взаимодействовать с пользователем или размещать другие виджеты на экране. Кнопки, текстовые поля, флажки — все это разновидности виджетов.

Android SDK включает множество виджетов, которые можно настраивать для получения нужного оформления и поведения. Каждый виджет является экземпляром класса View или одного из его subclasses (например, TextView или Button).

На рис. 1.9 показано, как выглядят на экране виджеты RelativeLayout и TextView, определенные в листинге 1.1.

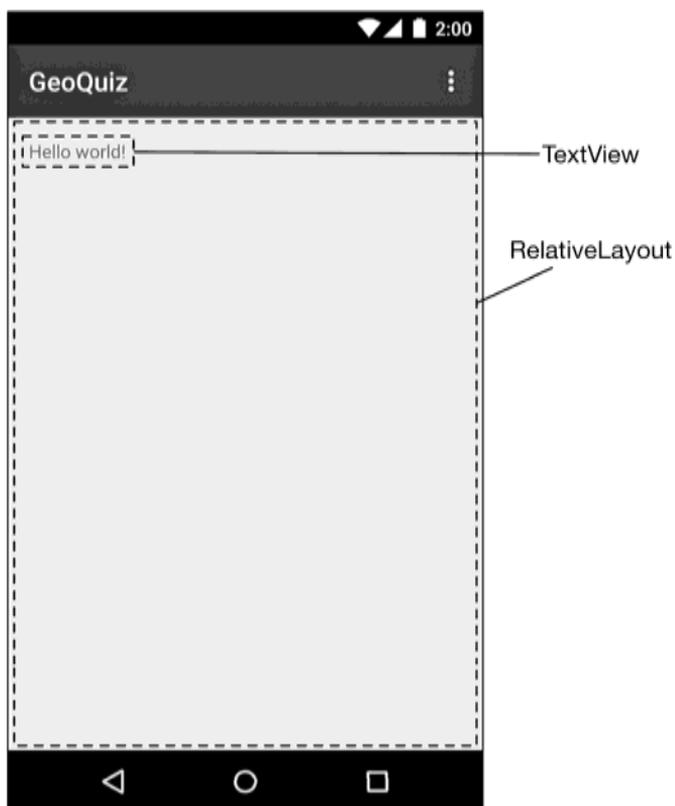


Рис. 1.9. Виджеты по умолчанию на экране

Впрочем, это не те виджеты, которые нам нужны. В интерфейсе QuizActivity задействованы пять виджетов:

- ① вертикальный виджет LinearLayout;
- ① TextView;

- ① горизонтальный виджет LinearLayout;
- ① две кнопки Button.

На рис. 1.10 показано, как из этих виджетов образуется интерфейс QuizActivity.

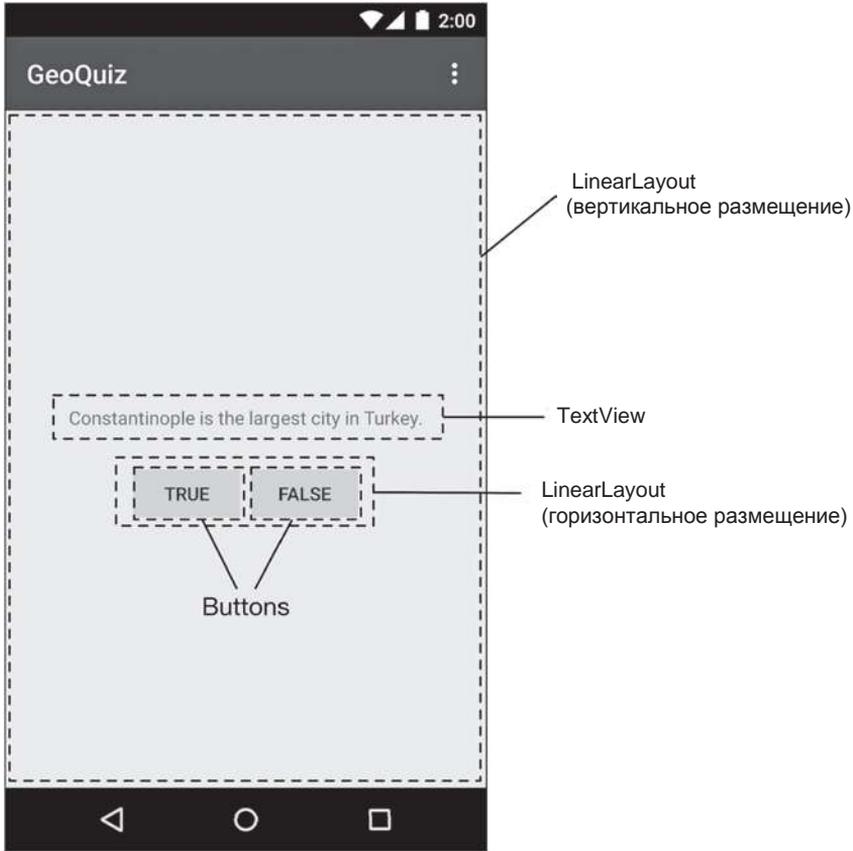


Рис. 1.10. Запланированное расположение виджетов на экране

Теперь нужно определить эти виджеты в файле activity_quiz.xml.

Внесите в файл activity_quiz.xml изменения, представленные в листинге 1.2. Разметка XML, которую нужно удалить, выделена перечеркиванием, а добавляемая разметка XML выделена жирным шрифтом. Эти обозначения будут использоваться в книге.

Небеспокойтесь, если смысл вводимой разметки остается непонятным; скоро вы узнаете, как она работает. Но будьте внимательны: разметка макета не проверяется, и опечатки рано или поздно создадут проблемы.

В зависимости от вашей версии инструментария в трех строках, начинающихся с android:text, могут быть обнаружены ошибки. Пока не обращайтесь внимания, мы их скоро исправим.

Листинг 1.2. Определение виджетов в XML (activity_quiz.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"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".QuizActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical" >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/question_text" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/true_button" />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/false_button" />

    </LinearLayout>

</LinearLayout>
```

Сравните XML с пользовательским интерфейсом, изображенным на рис. 1.10. Каждому виджету в разметке соответствует элемент XML. Имя элемента определяет тип виджета.

Каждый элемент обладает набором *атрибутов* XML. Атрибуты можно рассматривать как инструкции по настройке виджетов.

Чтобы лучше понять, как работают элементы и атрибуты, полезно взглянуть на разметку с точки зрения иерархии.

Иерархия представлений

Виджеты входят в иерархию объектов View, называемую *иерархией представлений*. На рис. 1.11 изображена иерархия виджетов для разметки XML из листинга 1.2.

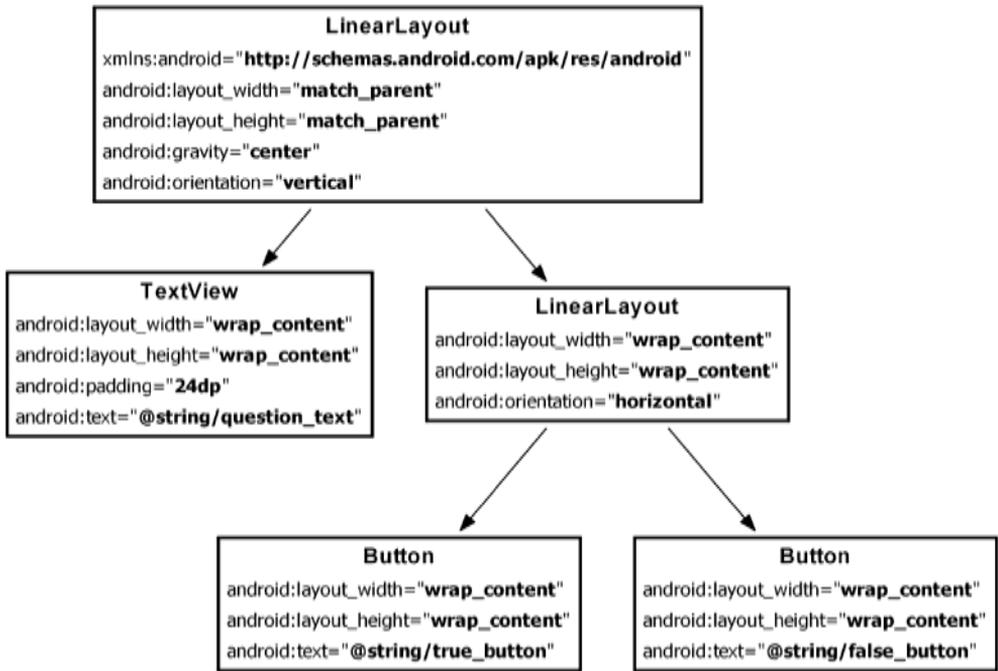


Рис. 1.11. Иерархия виджетов и атрибутов

Корневым элементом иерархии представлений в этом макете является элемент LinearLayout. В нем должно быть указано пространство имен XML ресурсов Android <http://schemas.android.com/apk/res/android>.

LinearLayout наследует от subclasses View с именем ViewGroup. Виджет ViewGroup предназначен для хранения и размещения других виджетов. LinearLayout используется в тех случаях, когда вы хотите выстроить виджеты в один столбец или строку. Другие subclasses ViewGroup — FrameLayout, TableLayout и RelativeLayout.

Если виджет содержится в ViewGroup, он называется *потомком* (child) ViewGroup. Корневой элемент LinearLayout имеет двух потомков: TextView и другой элемент LinearLayout. У LinearLayout имеются два собственных потомка Button.

Атрибуты виджетов

Рассмотрим некоторые атрибуты, используемые для настройки виджетов.

android:layout_width и **android:layout_height**

Атрибуты `android:layout_width` и `android:layout_height`, определяющие ширину и высоту, необходимы практически для всех разновидностей виджетов. Как правило, им задаются значения `match_parent` или `wrap_content`:

- ① `match_parent` — размеры представления определяются размерами родителя;
- ① `wrap_content` — размеры представления определяются размерами содержимого.

(Иногда в разметке встречается значение `fill_parent`. Это устаревшее значение эквивалентно `match_parent`.)

В корневом элементе `LinearLayout` атрибуты ширины и высоты равны `match_parent`. Элемент `LinearLayout` является корневым, но у него все равно есть родитель — представление, которое предоставляет Android для размещения иерархии представлений вашего приложения.

У других виджетов макета ширине и высоте задается значение `wrap_content`. На рис. 1.10 показано, как в этом случае определяются их размеры.

Виджет `TextView` чуть больше содержащегося в нем текста из-за атрибута `android:padding="24dp"`. Этот атрибут приказывает виджету добавить заданный отступ вокруг содержимого при определении размера, чтобы текст вопроса не соприкасался с кнопкой. (Интересуетесь, что это за единицы — `dp`? Это пиксели, не зависящие от плотности (`density-independent pixels`), о которых будет рассказано в главе 8.)

android:orientation

Атрибут `android:orientation` двух виджетов `LinearLayout` определяет, как будут выстраиваться потомки — по вертикали или горизонтали. Корневой элемент `LinearLayout` имеет вертикальную ориентацию; у его потомка `LinearLayout` горизонтальная ориентация.

Порядок определения потомков определяет порядок их отображения на экране. В вертикальном элементе `LinearLayout` потомок, определенный первым, располагается выше остальных. В горизонтальном элементе `LinearLayout` первый потомок является крайним левым. (Если только на устройстве не используется язык с письменностью справа налево, например арабский или иврит; в этом случае первый потомок будет находиться в крайней правой позиции.)

android:text

Виджеты `TextView` и `Button` содержат атрибуты `android:text`. Этот атрибут сообщает виджету, какой текст должен в нем отображаться.

Обратите внимание: значения атрибутов представляют собой не строковые литералы, а ссылки на строковые ресурсы.

Строковый ресурс — строка, находящаяся в отдельном файле XML, который называется *строковым файлом*. Виджету можно назначить фиксированную строку (например, `android:text="True"`), но так делать не стоит. Лучше размещать строки в отдельном файле, а затем ссылаться на них, так как использование строковых ресурсов упрощает локализацию.

Строковые ресурсы, на которые мы ссылаемся в `activity_quiz.xml`, еще не существуют. Давайте исправим этот недостаток.

Создание строковых ресурсов

Каждый проект включает строковый файл по умолчанию с именем `strings.xml`.

Найдите в окне Project каталог `app/res/values`, раскройте его и откройте файл `strings.xml`.

Графический интерфейс нас пока не интересует; выберите вкладку `strings.xml` в нижней части редактора.

В шаблоне уже включено несколько строковых ресурсов. Удалите неиспользуемую строку с именем `hello_world` и добавьте три новые строки для вашего макета.

Листинг 1.3. Добавление строковых ресурсов (`strings.xml`)

```
<resources>
  <string name="app_name">GeoQuiz</string>
  <string name="hello_world">Hello world!</string>
  <string name="question_text">
    Constantinople is the largest city in Turkey.
  </string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="action_settings">Settings</string>
</resources>
```

(Не удаляйте строку `menu_settings` — ваш проект содержит готовое меню. Удаление `menu_settings` вызовет каскадные ошибки в других файлах, относящихся к меню.)

Теперь по ссылке `@string/false_button` в любом файле XML проекта GeoQuiz вы будете получать строковый литерал "False" на стадии выполнения.

Сохраните файл `strings.xml`. Если в файле `activity_quiz.xml` оставались ошибки, связанные с отсутствием строковых ресурсов, они должны исчезнуть. (Если ошибки остались, проверьте оба файла — возможно, где-то допущена опечатка.)

Строковый файл по умолчанию называется `strings.xml`, но ему можно присвоить любое имя на ваше усмотрение. Проект может содержать несколько строковых файлов. Если файл находится в каталоге `res/values/`, содержит корневой элемент `resources` и дочерние элементы `string`, ваши строки будут найдены и правильно использованы приложением.

Предварительный просмотр макета

Макет готов, и его можно просмотреть в графическом конструкторе (рис. 1.12). Прежде всего убедитесь в том, что файлы сохранены и не содержат ошибок. Затем вернитесь к файлу `activity_quiz.xml` и откройте панель Preview при помощи вкладки в правой части редактора.



Рис. 1.12. Предварительный просмотр в графическом конструкторе макетов (`activity_quiz.xml`)

От разметки XML к объектам View

Как элементы XML в файле `activity_quiz.xml` превращаются в объекты View? Ответ на этот вопрос начинается с класса `QuizActivity`.

При создании проекта `GeoQuiz` был автоматически создан subclass `Activity` с именем `QuizActivity`. Файл класса `QuizActivity` находится в каталоге `app/java` (в котором хранится Java-код вашего проекта).

В окне инструментов `Project` откройте каталог `app/java`, а затем содержимое пакета `com.bignerdranch.android.geoquiz`. Откройте файл `QuizActivity.java` и просмотрите его содержимое (листинг 1.4).

Листинг 1.4. Файл класса QuizActivity по умолчанию (QuizActivity.java)

```

package com.bignerdranch.android.geoquiz;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class QuizActivity extends AppCompatActivity {

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

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}

```

(Интересуетесь, что такое AppCompatActivity? Это субкласс, наследующий от класса Android Activity и обеспечивающий поддержку старых версий Android. Более подробная информация об AppCompatActivity приведена в главе 13.)

(Если вы не видите все директивы import, щелкните на значке ⊕ слева от первой директивы import, чтобы раскрыть список.)

Файл содержит три метода Activity: onCreate(Bundle), onCreateOptionsMenu(Menu) и onOptionsItemSelected(MenuItem).

Пока не обращайтесь внимания на методы onCreateOptionsMenu(Menu) и onOptionsItemSelected(MenuItem). Мы вернемся к теме меню в главе 13.

Метод onCreate(Bundle) вызывается при создании экземпляра субкласса активности. Такому классу нужен пользовательский интерфейс, которым он будет управлять. Чтобы предоставить классу активности его пользовательский интерфейс, следует вызвать следующий метод Activity:

```
public void setContentView(int layoutResID)
```

Этот метод *заполняет* (inflates) макет и выводит его на экран. При заполнении макета создаются экземпляры всех виджетов в файле макета с параметрами, опреде-

ляемыми его атрибутами. Чтобы указать, какой именно макет следует заполнить, вы передаете идентификатор ресурса макета.

Ресурсы и идентификаторы ресурсов

Макет представляет собой *ресурс*. Ресурсом называется часть приложения, которая не является кодом, — графические файлы, аудиофайлы, файлы XML и т. д.

Ресурсы проекта находятся в подкаталоге каталога `app/res`. В окне инструментов Project видно, что файл `activity_quiz.xml` находится в каталоге `res/layout/`. Строковый файл, содержащий строковые ресурсы, находится в `res/values/`.

Для обращения к ресурсу в коде используется его идентификатор ресурса. Нашему макету назначен идентификатор ресурса `R.layout.activity_quiz`.

Чтобы просмотреть текущие идентификаторы ресурсов проекта GeoQuiz, необходимо сначала изменить режим представления проекта. По умолчанию Android Studio использует режим представления Android (рис. 1.13). В этом режиме истинная структура каталогов проекта Android скрывается, чтобы вы могли сосредоточиться на тех файлах и папках, которые чаще всего нужны программисту.

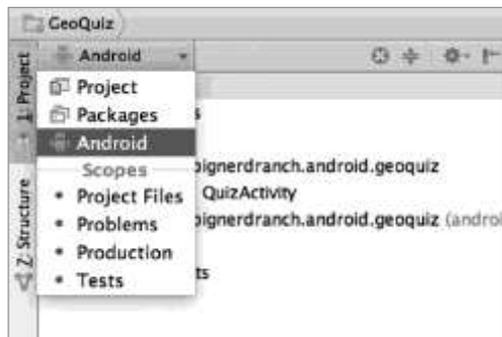


Рис. 1.13. Изменение режима представления проекта

Найдите раскрывающийся список в верхней части окна инструментов Project и выберите вместо режима Android режим Project. В этом режиме файлы и папки проекта представлены в своем фактическом состоянии.

Чтобы просмотреть ресурсы приложения GeoQuiz, раскройте содержимое каталога `app/build/generated/source/r/debug`. В этом каталоге найдите имя пакета своего проекта и откройте файл `R.java` из этого пакета. Поскольку этот файл генерируется процессом сборки Android, вам не следует его изменять, о чем деликатно предупреждает надпись в начале файла.

Возможно, внесение изменений в ресурсы не приведет к моментальному обновлению этого файла. Android Studio поддерживает скрытый файл `R.java`, который используется при построении вашего кода. Тот файл `R.java`, который вы видите, был сгенерирован для вашего приложения перед его установкой на устройстве или эмуляторе. Вы увидите, что файл будет обновлен при запуске приложения.

Листинг 1.5. Текущие идентификаторы GeoQuiz

```

/* AUTO-GENERATED FILE. DO NOT MODIFY.
 *
 * ---
 */

package com.bignerdranch.android.geoquiz;

public final class R {
    public static final class anim {
        ---
    }
    ---

    public static final class id {
        ---
    }
    public static final class layout {
        ---
        public static final int activity_quiz=0x7f030017;
    }
    public static final class mipmap {
        public static final int ic_launcher=0x7f030000;
    }
    public static final class string {
        ---
        public static final int app_name=0x7f0a0010;
        public static final int correct_toast=0x7f0a0011;
        public static final int false_button=0x7f0a0012;
        public static final int incorrect_toast=0x7f0a0013;
        public static final int question_text=0x7f0a0014;
        public static final int true_button=0x7f0a0015;
    }
}

```

Файл `R.java` может быть довольно большим; в листинге 1.5 значительная часть его содержимого не показана.

Теперь понятно, откуда взялось имя `R.layout.activity_quiz` — это целочисленная константа с именем `activity_quiz` из внутреннего класса `layout` класса `R`.

Строкам также назначаются идентификаторы ресурсов. Мы еще не ссылались на строки в коде, но эти ссылки обычно выглядят так:

```
setTitle(R.string.app_name);
```

Android генерирует идентификатор ресурса для всего макета и для каждой строки, но не для отдельных виджетов из файла `activity_quiz.xml`. Не каждому виджету нужен идентификатор ресурса. В этой главе мы будем взаимодействовать лишь с двумя кнопками, поэтому идентификаторы ресурсов нужны только им.

Прежде чем генерировать идентификаторы ресурсов, переключитесь обратно в режим представления Android. В книге мы будем работать в этом режиме — но ничто не мешает вам использовать режим Project, если вы этого хотите.

Чтобы сгенерировать идентификатор ресурса для виджета, включите в определение виджета атрибут `android:id`. В файле `activity_quiz.xml` добавьте атрибут `android:id` для каждой кнопки.

Листинг 1.6. Добавление идентификаторов кнопок (`activity_quiz.xml`)

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
... >

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="24dp"
    android:text="@string/question_text" />

<LinearLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

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

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

</LinearLayout>

</LinearLayout>
```

Обратите внимание: знак `+` присутствует в значениях `android:id`, но не в значениях `android:text`. Это связано с тем, что мы *создаем* идентификаторы, а на строки только *ссылаемся*.

Подключение виджетов к программе

Теперь, когда кнопкам назначены идентификаторы ресурсов, к ним можно обращаться в `QuizActivity`. Все начинается с добавления двух переменных.

Введите следующий код в `QuizActivity.java`. (Не используйте автозавершение; введите его самостоятельно.) После сохранения файла выводятся два сообщения об ошибках.

Листинг 1.7. Добавление полей (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

    ...
}
```

Сейчас мы исправим ошибки, а пока обратите внимание на префикс имен двух полей (переменных экземпляров). Этот префикс соответствует схеме формирования имен Android, которая будет использоваться в этой книге.

Наведите указатель мыши на красные индикаторы ошибок. Они сообщают об одинаковой проблеме: «Не удастся разрешить символическое имя Button» (Cannot resolve symbol 'Button').

Чтобы избавиться от ошибок, следует импортировать класс `android.widget.Button` в `QuizActivity.java`. Введите следующую директиву импортирования в начале файла:

```
import android.widget.Button;
```

А можно пойти по простому пути и поручить эту работу Android Studio. Нажмите `Option+Return` (или `Alt+Enter`) — волшебство IntelliJ приходит на помощь. Новая директива `import` теперь появляется под другими директивами в начале файла. Этот прием часто бывает полезным, если ваш код работает не так, как положено. Экспериментируйте с ним почаще!

Ошибки должны исчезнуть. (Если они остались, поищите опечатки в коде и XML.)

Теперь вы можете подключить свои виджеты-кнопки. Процедура состоит из двух шагов:

- ① получение ссылок на заполненные объекты `View`;
- ① назначение для этих объектов слушателей, реагирующих на действия пользователя.

Получение ссылок на виджеты

В классе активности можно получить ссылку на заполненный виджет, для чего используется следующий метод `Activity`:

```
public View findViewById(int id)
```

Метод получает идентификатор ресурса виджета и возвращает объект `View`.

В файле QuizActivity.java по идентификаторам ресурсов ваших кнопок можно получить заполненные объекты и присвоить их полям. Учтите, что возвращенный объект View перед присваиванием необходимо преобразовать в Button.

Листинг 1.8. Получение ссылок на виджеты (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {

    private Button mTrueButton;
    private Button mFalseButton;

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

        mTrueButton = (Button) findViewById(R.id.true_button);
        mFalseButton = (Button) findViewById(R.id.false_button);
    }

    ...
}
```

Назначение слушателей

Приложения Android обычно *управляются событиями* (event-driven). В отличие от программ командной строки или сценариев, такие приложения запускаются и ожидают наступления некоторого события, например нажатия кнопки пользователем. (События также могут инициироваться ОС или другим приложением, но события, инициируемые пользователем, наиболее очевидны.)

Когда ваше приложение ожидает наступления конкретного события, мы говорим, что оно «прослушивает» данное событие. Объект, создаваемый для ответа на событие, называется *слушателем* (listener). Такой объект реализует *интерфейс слушателя* данного события.

Android SDK поставляется с интерфейсами слушателей для разных событий, поэтому вам не придется писать собственные реализации. В нашем случае прослушиваемым событием является «щелчок» на кнопке, поэтому слушатель должен реализовать интерфейс View.OnClickListener.

Начнем с кнопки True. В файле QuizActivity.java включите следующий фрагмент кода в метод onCreate(...) непосредственно после присваивания.

Листинг 1.9. Назначение слушателя для кнопки True (QuizActivity.java)

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

    mTrueButton = (Button) findViewById(R.id.true_button);
```

```

mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Пока ничего не делает, но скоро будет!
    }
});

mFalseButton = (Button)findViewById(R.id.false_button);
}
}

```

(Если вы получите ошибку «View cannot be resolved to a type», воспользуйтесь комбинацией Option+Return (Alt+Enter) для импортирования класса View.)

В листинге 1.9 назначается слушатель, информирующий о нажатии виджета Button с именем mTrueButton. Метод setOnClickListener(OnClickListener) получает в аргументе слушателя, а конкретнее — объект, реализующий OnClickListener.

Анонимные внутренние классы

Слушатель реализован в виде *анонимного внутреннего класса*. Возможно, синтаксис не очевиден; просто запомните: все, что заключено во внешнюю пару круглых скобок, передается setOnClickListener(OnClickListener). В круглых скобках создается новый безымянный класс, вся реализация которого передается вызываемому методу.

```

mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Пока ничего не делает, но скоро будет!
    }
});

```

Все слушатели в этой книге будут реализованы в виде анонимных внутренних классов. В этом случае реализация методов слушателя находится непосредственно там, где вы хотите ее видеть, а мы избегаем затрат ресурсов на создание именованного класса, который будет использоваться только в одном месте.

Так как анонимный класс реализует OnClickListener, он должен реализовать единственный метод этого интерфейса onClick(View). Мы пока оставили реализацию onClick(View) пустой, и компилятор не протестует. Интерфейс слушателя требует, чтобы метод onClick(View) был реализован, но не устанавливает никаких правил относительно того, *как именно* он будет реализован.

(Если ваши знания в области анонимных внутренних классов, слушателей или интерфейсов оставляют желать лучшего, полистайте учебник по Java перед тем, как продолжать, или хотя бы держите справочник под рукой.)

Назначьте аналогичного слушателя для кнопки False.

Листинг 1.10. Назначение слушателя для кнопки False (QuizActivity.java)

```
...
    mTrueButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Пока ничего не делает, но скоро будет!
        }
    });

    mFalseButton = (Button)findViewById(R.id.false_button);
    mFalseButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            // Пока ничего не делает, но скоро будет!
        }
    });
}
```

Уведомления

Пора заставить кнопки делать что-то полезное. В нашем приложении каждая кнопка будет выводить на экран временное *уведомление* (toast) — короткое сообщение, которое содержит какую-либо информацию для пользователя, но не требует ввода, ни действий. Наши уведомления будут сообщать пользователю, правильно ли он ответил на вопрос (рис. 1.14).

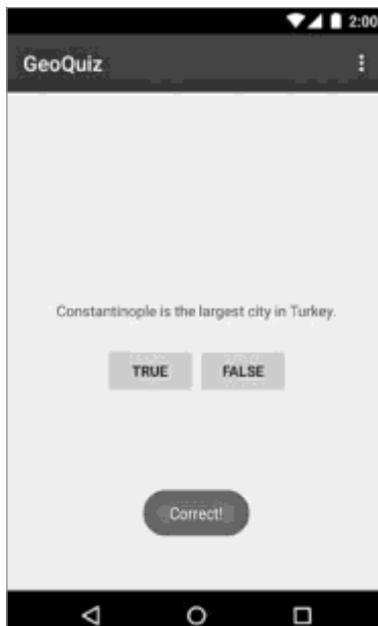


Рис. 1.14. Уведомление с информацией для пользователя

Для начала вернитесь к файлу `strings.xml` и добавьте строковые ресурсы, которые будут отображаться в уведомлении.

Листинг 1.11. Добавление строк уведомлений (`strings.xml`)

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

  <string name="question_text">Constantinople is the largest city
    in Turkey.</string>
  <string name="true_button">True</string>
  <string name="false_button">False</string>
  <string name="correct_toast">Correct!</string>
  <string name="incorrect_toast">Incorrect!</string>
  <string name="menu_settings">Settings</string>
</resources>
```

Уведомление создается вызовом следующего метода класса `Toast`:

```
public static Toast makeText(Context context, int resId, int duration)
```

Параметр `Context` обычно содержит экземпляр `Activity` (`Activity` является subclassом `Context`). Во втором параметре передается идентификатор ресурса строки, которая должна выводиться в уведомлении. Параметр `Context` необходим классу `Toast` для поиска и использования идентификатора ресурса строки. Третий параметр обычно содержит одну из двух констант `Toast`, определяющих продолжительность пребывания уведомления на экране.

После того как объект уведомления будет создан, вызовите `Toast.show()`, чтобы уведомление появилось на экране.

В классе `QuizActivity` вызов `makeText(...)` будет присутствовать в слушателе каждой кнопки (листинг 1.12). Вместо того чтобы вводить все вручную, попробуйте добавить эти вызовы с использованием функции автозавершения среды `Android Studio`.

Автозавершение

Автозавершение экономит много времени, так что с ним стоит познакомиться пораньше.

Начните вводить новый код из листинга 1.12. Когда вы доберетесь до точки после класса `Toast`, на экране появляется список методов и констант класса `Toast`.

Чтобы выбрать одну из рекомендаций, используйте клавиши \uparrow и \downarrow . (Если вы не собираетесь использовать автозавершение, просто продолжайте печатать. Без нажатия клавиши `Tab`, клавиши `Return/Enter` или щелчка на окне подсказки подстановка не выполняется.)

Выберите в списке рекомендаций метод `makeText(Context context, int resID, int duration)`. Механизм автозавершения добавляет полный вызов метода.

Задайте параметры метода `makeText` так, как показано в листинге 1.12.

Листинг 1.12. Создание уведомлений (QuizActivity.java)

```
...
mTrueButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.incorrect_toast,
                       Toast.LENGTH_SHORT).show();
        // Пока ничего не делает, но скоро будет!
    }
});

mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
                       R.string.correct_toast,
                       Toast.LENGTH_SHORT).show();
        // Пока ничего не делает, но скоро будет!
    }
});
```

В вызове `makeText(...)` экземпляр `QuizActivity` передается в аргументе `Context`. Но как и следовало ожидать, просто передать `this` нельзя. В этом месте кода мы определяем анонимный класс, где `this` обозначает `View.OnClickListener`.

Благодаря использованию автозавершения вам не придется ничего специально делать для импортирования класса `Toast`. Когда вы соглашаетесь на рекомендацию автозавершения, необходимые классы импортируются автоматически.

Сохраните внесенные изменения. Посмотрим, как работает новое приложение.

Выполнение в эмуляторе

Для запуска приложений Android необходимо устройство — физическое или *виртуальное*. Виртуальные устройства работают под управлением эмулятора Android, включенного в поставку средств разработчика.

Чтобы создать виртуальное устройство Android (AVD, Android Virtual Device), выполните команду `Tools ▸ Android ▸ AVD Manager`. Когда на экране появится окно `AVD Manager`, щелкните на кнопке `Create Virtual Device...` в левой части этого окна.

Открывается диалоговое окно с многочисленными параметрами настройки виртуального устройства. Выберите эмуляцию устройства `Nexus 5`, как показано на рис. 1.15. Щелкните на кнопке `Next`.

На следующем экране выберите образ системы, на основе которого будет работать эмулятор. Выберите эмулятор `x86 Lollipop` и щелкните на кнопке `Next` (рис. 1.16).

Наконец, просмотрите и при необходимости измените свойства эмулятора (впрочем, свойства существующего эмулятора также можно изменить позднее). Пока

присвойте своему эмулятору имя, по которому вы сможете узнать его в будущем, и щелкните на кнопке Finish (рис. 1.17).

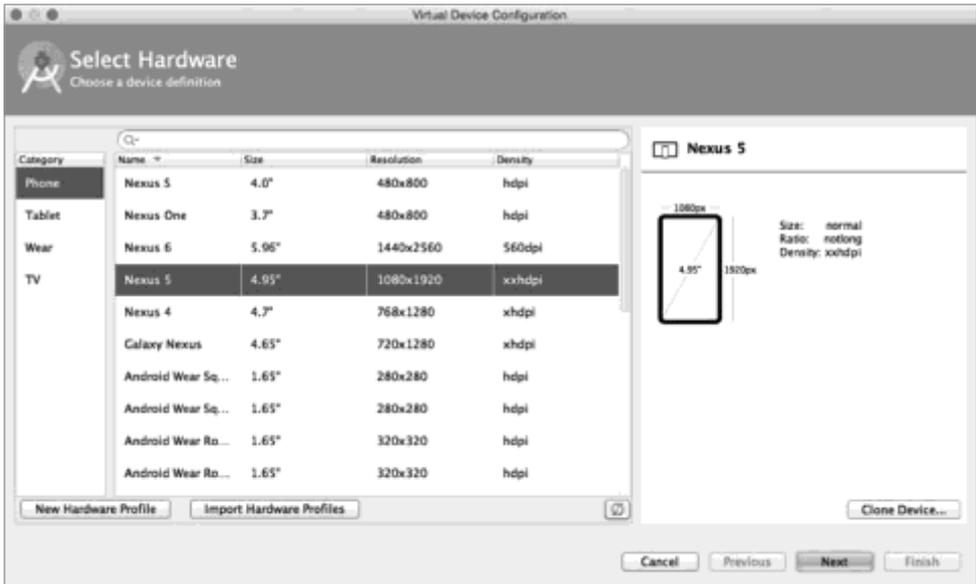


Рис. 1.15. Выбор виртуального устройства

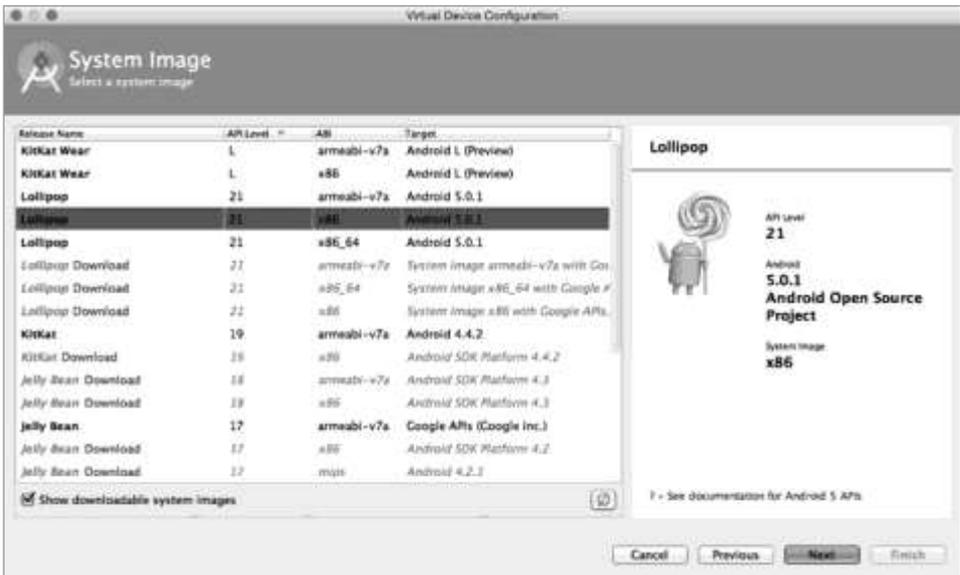


Рис. 1.16. Выбор образа системы



Рис. 1.17. Настройка свойств эмулятора

Когда виртуальное устройство будет создано, в нем можно запустить приложение GeoQuiz. На панели инструментов Android Studio щелкните на кнопке Run (зеленый символ «воспроизведение») или нажмите Ctrl+R. Android Studio находит созданное виртуальное устройство, запускает его, устанавливает на нем пакет приложения и запускает приложение.

Возможно, запуск эмулятора потребует некоторого времени, но вскоре приложение GeoQuiz запустится на созданном вами виртуальном устройстве. Понажимайте кнопки и оцените уведомления. (Если приложение запускается, когда вас нет поблизости, возможно, вам придется разблокировать AVD после возвращения. AVD работает как настоящее устройство и блокируется после некоторого времени бездействия.)

Если при запуске GeoQuiz или нажатии кнопки происходит сбой, в нижней части представления LogCat панели Android DDMS выводится полезная информация. (Если представление LogCat не открылось автоматически при запуске GeoQuiz, его можно открыть при помощи кнопки Android в нижней части окна Android Studio.) Ищите исключения в журнале; они будут выделяться красным цветом (рис. 1.18).

Text
<pre> at dalvik.system.NativeStart.main(Native Method) Caused by: java.lang.NullPointerException at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:21) at android.app.Activity.performCreate(Activity.java:5008) at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079) </pre>

Рис. 1.18. Исключение NullPointerException в строке 21

Сравните свой код с кодом в книге и попытайтесь найти источник проблемы. Затем снова запустите приложение. (LogCat и процесс отладки более подробно рассматриваются в следующих двух главах.)

Не закрывайте эмулятор; не стоит ждать, пока он загружается, при каждом запуске. Вы можете остановить приложение кнопкой Back (U-образная стрелка), а затем снова запустить приложение из Android Studio, чтобы протестировать изменения.

Эмулятор полезен, но тестирование на реальном устройстве дает более точные результаты. В главе 2 мы запустим приложение GeoQuiz на физическом устройстве, а также расширим набор вопросов по географии.

Для любознательных: процесс построения приложений Android

Вероятно, у вас уже накопились неотложные вопросы относительно того, как работает процесс построения приложений Android. Вы уже видели, что Android Studio строит проект автоматически в процессе его изменения, а не по команде. Во время построения инструментарий Android берет ваши ресурсы, код и файл `AndroidManifest.xml` (содержащий метаданные приложения) и преобразует их в файл `.apk`. Полученный файл подписывается отладочным ключом, что позволяет запускать его в эмуляторе. (Чтобы распространять файл `.apk` среди пользователей, необходимо подписать его ключом публикации. Дополнительную информацию об этом процессе можно найти в документации разработчика Android по адресу <http://developer.android.com/tools/publishing/preparing.html>.)

Общая схема процесса построения представлена на рис. 1.19.

Как содержимое `activity_quiz.xml` преобразуется в объекты `View` в приложении? В процессе построения утилита `aapt` (Android Asset Packaging Tool) компилирует ресурсы файлов макетов в более компактный формат. Откомпилированные ресурсы упаковываются в файл `.apk`. Затем, когда метод `setContentView(...)` вызывается в методе `onCreate(...)` класса `QuizActivity`, `QuizActivity` использует класс `LayoutInflater` для создания экземпляров всех объектов `View`, определенных в файле макета (рис. 1.20).

(Также классы представлений можно создать на программном уровне в классе активности вместо определения их в XML. Однако отделение презентационной логики от логики приложения имеет свои преимущества, главное из которых — использование изменений конфигурации, встроенное в SDK; мы подробнее поговорим об этом в главе 3.)

За дополнительной информацией о том, как работают различные атрибуты XML и как происходит отображение представлений на экране, обращайтесь к главе 8.

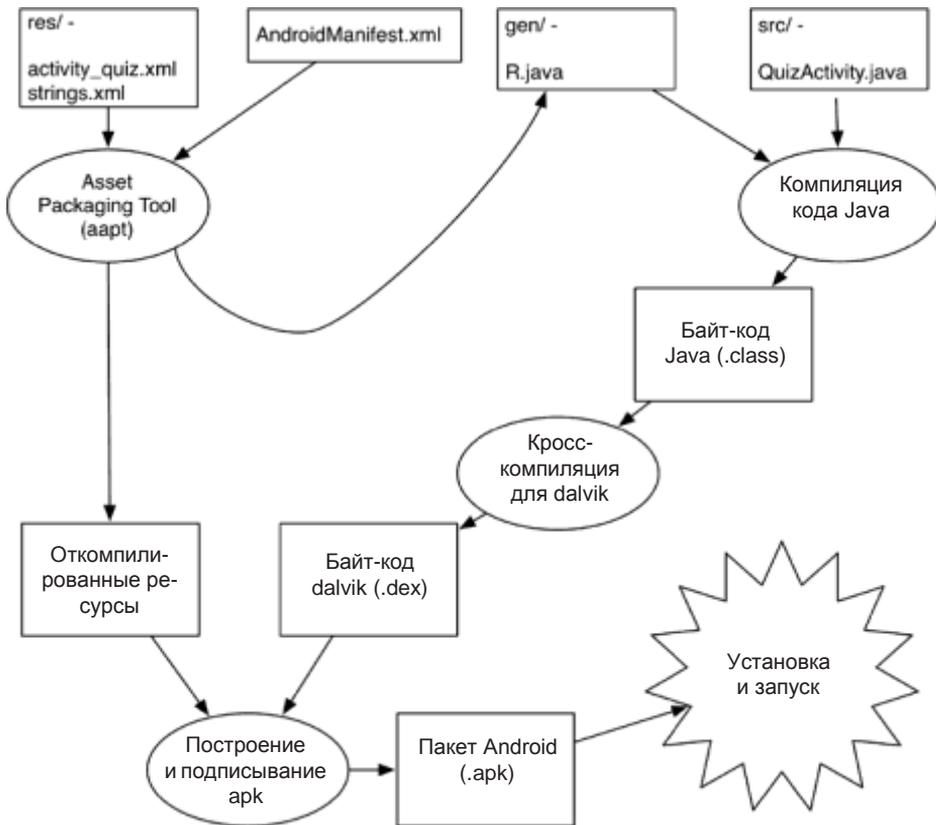


Рис. 1.19. Построение GeoQuiz

Средства построения программ Android

Все программы, которые мы запускали до настоящего времени, исполнялись из среды Android Studio. Этот процесс интегрирован в среду разработки — он вызывает стандартные средства построения программ Android (такие, как *aapt*), но сам процесс построения проходит под управлением Android Studio.

Может оказаться, что вам по каким-то своим причинам потребуется провести построение за пределами среды Android Studio. Для этого проще всего воспользоваться программой командной строки. В современной системе построения приложений Android используется программа Gradle.

(Вы и сами поймете, представляет ли эта информация для вас интерес. Если не представляет, просто бегло просмотрите ее, не беспокоясь о том, зачем это нужно и что делать, если что-то не работает. Рассмотрение всех тонкостей использования командной строки выходит за рамки книги.)

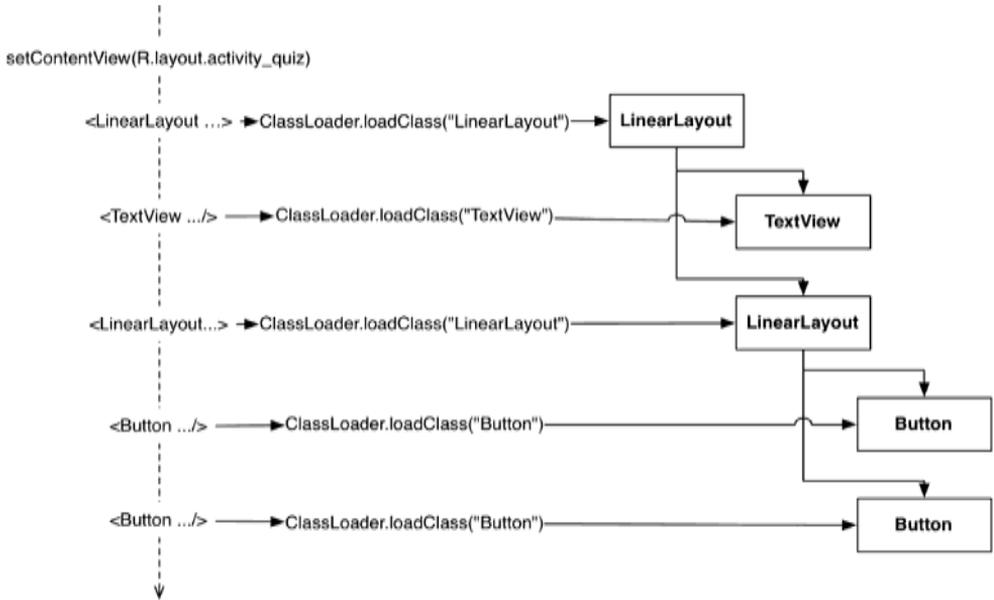


Рис. 1.20. Заполнение activity_quiz.xml

Чтобы использовать Gradle в режиме командной строки, перейдите в каталог своего проекта и выполните следующую команду:

```
$ ./gradlew tasks
```

В системе Windows команда выглядит немного иначе:

```
> gradlew.bat tasks
```

Команда выводит список задач, которые можно выполнить. Та задача, которая вам нужна, называется `installDebug`.

Запустите ее следующей командой:

```
$ ./gradlew installDebug
```

Или в системе Windows:

```
> gradlew.bat installDebug
```

Команда устанавливает приложение на подключенное устройство, но не запускает его — вы должны сделать это вручную.

1.2. Создание нового класса

В окне инструментов Project щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.geoquiz` и выберите команду `New ▶ Java Class`. Введите имя класса `Question` и щелкните на кнопке `OK` (рис. 2.2).

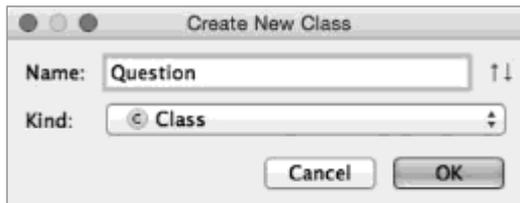


Рис. 2.2. Создание класса `Question`

Добавьте в файл `Question.java` два поля и конструктор:

Листинг 2.1. Добавление класса `Question` (`Question.java`)

```
public class Question {  
  
    private int mTextResId;  
    private boolean mAnswerTrue;  
  
    public Question(int textResId, boolean answerTrue) {  
        mTextResId = textResId;  
        mAnswerTrue = answerTrue;  
    }  
}
```

Класс `Question` содержит два вида данных: текст вопроса и правильный ответ (да/нет).

Почему поле `mTextResId` объявлено с типом `int`, а не `String`? В нем будет храниться идентификатор ресурса (всегда `int`) строкового ресурса с текстом вопроса. Мы займемся созданием этих строковых ресурсов в следующем разделе.

Для переменных необходимо определить `get-` и `set-` методы. Вводить их самостоятельно не нужно — проще приказать Android Studio сгенерировать реализации.

Генерирование `get-` и `set-`методов

Прежде всего следует настроить Android Studio на распознавание префикса `m` в полях классов.

Откройте окно настроек Android Studio (меню `Android Studio` на Mac, команда `File ▶ Settings` в системе Windows). Откройте раздел `Editor`, затем раздел `Code Style`. Выберите категорию `Java` и перейдите на вкладку `Code Generation`.

В таблице Naming найдите строку Field (рис. 2.3) и в поле Naming Prefix введите префикс m для полей. Затем добавьте префикс s для статических полей в строке Static field. (В проекте GeoQuiz префикс s не используется, но он пригодится в будущих проектах.)

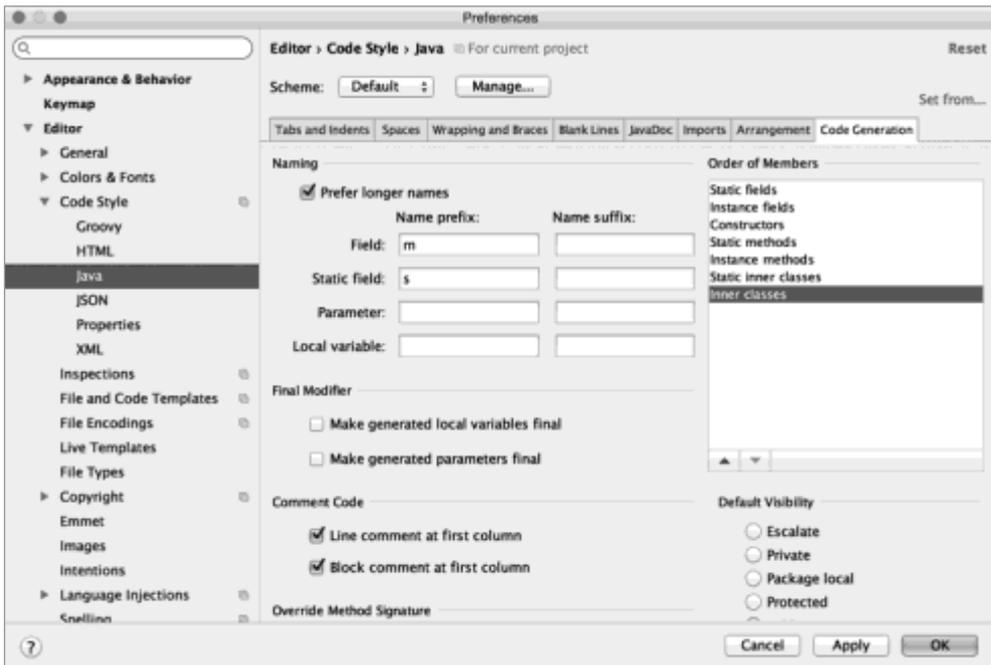


Рис. 2.3. Настройка стиля оформления кода Java

Щелкните на кнопке ОК.

Зачем мы задавали эти префиксы? Если теперь приказать Android Studio сгенерировать get-метод для mTextResId, среда сгенерирует методы с именем getTextResId() вместо getMTextResId() и isAnswerTrue() вместо isMAnswerTrue().

Вернитесь к файлу Question.java, щелкните правой кнопкой мыши после конструктора и выберите команду Generate... ▶ Getter And Setter. Выберите поля mTextResId и mAnswerTrue, затем щелкните на кнопке ОК, чтобы сгенерировать get- и set-метод для каждой переменной.

Листинг 2.2. Сгенерированные get- и set-методы (Question.java)

```
public class Question {

    private int mTextResId;
    private boolean mAnswerTrue;
    ...

    public int getTextResId() {
```

```
    return mTextResId;
}

public void setTextResId(int textResId) {
    mTextResId = textResId;
}

public boolean isAnswerTrue() {
    return mAnswerTrue;
}

public void setAnswerTrue(boolean answerTrue) {
    mAnswerTrue = answerTrue;
}
}
```

Класс Question готов. Вскоре мы внесем изменения в QuizActivity для работы с Question, но для начала посмотрим, как фрагменты GeoQuiz будут работать вместе. Класс QuizActivity должен создать массив объектов Question. В процессе работы он взаимодействует с TextView и тремя виджетами Button для вывода вопроса и предоставления обратной связи на ответы пользователя. Отношения между этими объектами изображены на рис. 2.4.

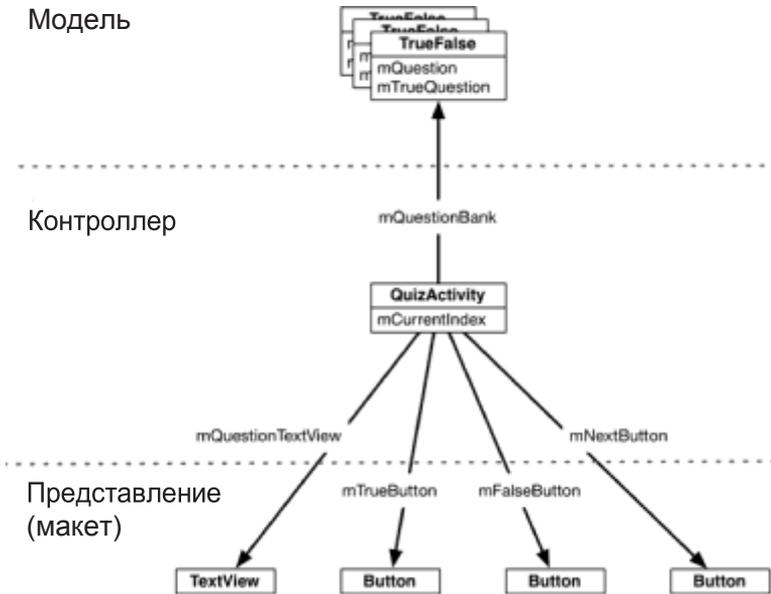


Рис. 2.4. Диаграмма объектов GeoQuiz

Архитектура «Модель-Представление-Контроллер» и Android

Вероятно, вы заметили, что объекты на рис. 2.4 разделены на три области: «Модель», «Контроллер» и «Представление». Приложения Android строятся на базе архитектуры, называемой «Модель-Представление-Контроллер», или сокращенно MVC (Model-View-Controller). Согласно канонам MVC, каждый объект приложения должен быть объектом модели, объектом представления или объектом контроллера.

❶ *Объект модели* содержит данные приложения и «бизнес-логику». Классы модели обычно проектируются для моделирования сущностей, с которыми имеет дело приложение, — пользователь, продукт в магазине, фотография на сервере, вопрос «да/нет» и т. д. Объекты модели ничего не знают о пользовательском интерфейсе; их единственной целью является хранение и управление данными.

В приложениях Android классы моделей обычно создаются разработчиком для конкретной задачи. Все объекты модели в вашем приложении составляют его *уровень модели*.

Уровень модели GeoQuiz состоит из класса Question.

❶ *Объекты представлений* умеют отображать себя на экране и **принимают** ввод пользователя, например касания. Простейшее правило: если вы видите что-то на экране, значит, это представление.

Android предоставляет широкий набор настраиваемых классов представлений. Разработчик также может создавать собственные классы представлений. Объекты представления в приложении образуют *уровень представления*.

В GeoQuiz уровень представления состоит из виджетов, заполненных по содержанию файла activity_quiz.xml.

❶ *Объекты контроллеров* связывают объекты представления и модели; они **держат** «логику приложения». Контроллеры реагируют на различные события, инициируемые объектами представлений, и управляют потоками данных между объектами модели и уровнем представления.

В Android контроллер обычно представляется субклассом Activity, Fragment или Service. (Фрагменты рассматриваются в главе 7, а службы — в главе 26.)

Уровень контроллера GeoQuiz в настоящее время состоит только из класса QuizActivity.

На рис. 2.5 показана логика передачи управления между объектами в ответ на пользовательское событие, такое как нажатие кнопки. Обратите внимание: объекты модели и представлений не взаимодействуют друг с другом напрямую; в любом взаимодействии участвуют «посредники»-контроллеры, получающие сообщения от одних объектов и передающие инструкции другим.



Рис. 2.5. Взаимодействия MVC при получении ввода от пользователя

Преимущества MVC

Приложение может обрастать функциональностью до тех пор, пока не станет слишком сложным для понимания. Разделение кода на классы упрощает проектирование и понимание приложения в целом; разработчик мыслит в контексте классов, а не отдельных переменных и методов.

Аналогичным образом разделение классов на уровни модели, представления и контроллера упрощает проектирование и понимание приложения; вы можете мыслить в контексте уровней, а не отдельных классов.

Приложение GeoQuiz нельзя назвать сложным, но преимущества разделения уровней проявляются и здесь. Вскоре мы обновим уровень представления GeoQuiz и добавим в него кнопку Next. При этом нам совершенно не нужно помнить о только что созданном классе Question.

MVC также упрощает повторное использование классов. Класс с ограниченными обязанностями лучше подходит для повторного использования, чем класс, который пытается заниматься всем сразу.

Например, класс модели Question ничего не знает о виджетах, используемых для вывода вопроса «да/нет». Это упрощает использование Question в приложении для разных целей. Например, если потребуется вывести полный список всех вопросов, вы можете использовать тот же объект, который используется для вывода всего одного вопроса.

Обновление уровня представления

После теоретического знакомства с MVC пора переходить к практике: обновим уровень представления GeoQuiz и включим в него кнопку Next.

В Android объекты уровня представления обычно заполняются на основе разметки XML в файле макета. Весь макет GeoQuiz определяется в файле `activity_quiz.xml`. В него следует внести изменения, представленные на рис. 2.6. (Для экономии места на рисунке не показаны атрибуты виджетов, оставшихся без изменений.)

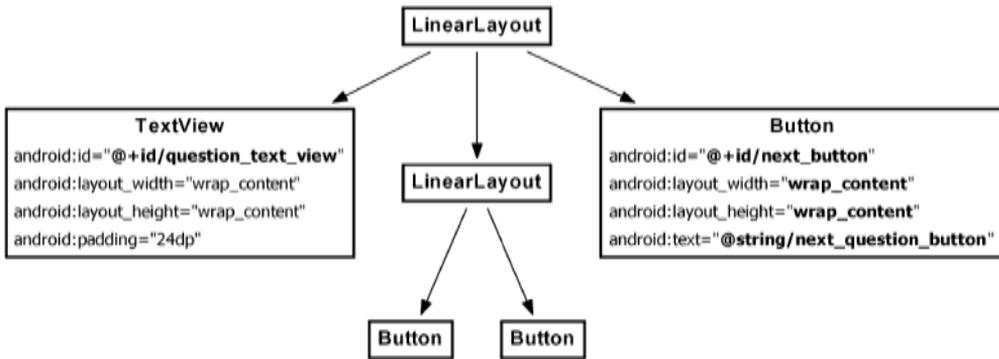


Рис. 2.6. Новая кнопка!

Итак, на уровне представления необходимо внести следующие изменения:

- ❶ Удалите атрибут `android:text` из `TextView`. Жестко ~~з~~уфиксированный текст вопроса не должен присутствовать в определении.
- ❶ Назначьте `TextView` атрибут `android:id`. Идентификатор ресурса ~~id~~ виджету для того, чтобы мы могли задать его текст в коде `QuizActivity`.
- ❶ Добавьте новый виджет `Button` как потомка корневого элемента `LinearLayout`.

Вернитесь к файлу `activity_quiz.xml` и выполните все перечисленные действия.

Листинг 2.3. Новая кнопка и изменения в `TextView` (`activity_quiz.xml`)

```

<LinearLayout
... >

<TextView
  android:id="@+id/question_text_view"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:padding="24dp"
  android:text="@string/question_text"
/>

<LinearLayout
... >

...
</LinearLayout>

<Button

```

```

    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button" />

```

```
</LinearLayout>
```

Сохраните файл `activity_quiz.xml`. Возможно, на экране появится знакомая ошибка с сообщением об отсутствующем строковом ресурсе.

Вернитесь к файлу `res/values/strings.xml`. Удалите строку вопроса и добавьте строку для новой кнопки.

Листинг 2.4. Обновленные строки (strings.xml)

```

...
<string name="app_name">GeoQuiz</string>
<del><string name="question_text">Constantinople is the largest city
in Turkey.</string></del>
<string name="true_button">True</string>
<string name="false_button">False</string>
<string name="next_button">Next</string>
<string name="correct_toast">Correct!</string>
...

```

Пока файл `strings.xml` открыт, добавьте строки с вопросами по географии, которые будут предлагаться пользователю.

Листинг 2.5. Добавление строк вопросов (strings.xml)

```

...
<string name="incorrect_toast">Incorrect!</string>
<string name="action_settings">Settings</string>
<string name="question_oceans">The Pacific Ocean is larger than
the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.
</string>
<string name="question_americas">The Amazon River is the longest river
in the Americas.</string>
<string name="question_asia">Lake Baikal is the world's oldest and deepest
freshwater lake.</string>
...

```

Обратите внимание на использование служебной последовательности `\'` для включения апострофа в последнюю строку. В строковых ресурсах могут использоваться все стандартные служебные последовательности, включая `\n` для обозначения новой строки.

Сохраните файлы. Вернитесь к файлу `activity_quiz.xml` и ознакомьтесь с изменениями макета в графическом конструкторе.

Пока это все, что относится к уровню представления GeoQuiz. Пора связать все воедино в классе контроллера QuizActivity.

Обновление уровня контроллера

В предыдущей главе в единственном контроллере GeoQuiz — QuizActivity — не происходило почти ничего. Он отображал макет, определенный в файле activity_quiz.xml, назначал слушателей для двух кнопок и организовывал выдачу уведомлений.

Теперь, когда у нас появились дополнительные вопросы, классу QuizActivity придется приложить дополнительные усилия для связывания уровней модели и представления GeoQuiz.

Откройте файл QuizActivity.java. Добавьте переменные для TextView и новой кнопки Button. Также создайте массив объектов Question и переменную для индекса массива.

Листинг 2.6. Добавление переменных и массива Question (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {  
  
    private Button mTrueButton;  
    private Button mFalseButton;  
    private Button mNextButton; private  
    TextView mQuestionTextView;  
  
    private Question[] mQuestionBank = new Question[] {  
        new Question(R.string.question_oceans, true),  
        new Question(R.string.question_mideast, false),  
        new Question(R.string.question_africa, false),  
        new Question(R.string.question_americas, true),  
        new Question(R.string.question_asia, true),  
    };  
  
    private int mCurrentIndex = 0;  
    ---
```

Программа несколько раз вызывает конструктор Question и создает массив объектов Question.

(В более сложном проекте этот массив создавался бы и хранился в другом месте. Позднее мы рассмотрим более правильные варианты хранения данных модели. А пока для простоты массив будет создаваться в контроллере.)

Мы собираемся использовать mQuestionBank, mCurrentIndex и методы доступа Question для вывода на экран серии вопросов.

Начнем с получения ссылки на TextView и задания тексту виджета вопроса с текущим индексом.

Листинг 2.7. Подключение виджета TextView (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ---
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_quiz);

        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);

        mTrueButton = (Button)findViewById(R.id.true_button);
        ---
    }
}
```

Сохраните файлы и проверьте возможные ошибки. Запустите программу GeoQuiz. Первый вопрос из массива должен отображаться в виджете TextView.

Теперь разберемся с кнопкой Next. Получите ссылку на кнопку, назначьте ей слушателя View.OnClickListener. Этот слушатель будет увеличивать индекс и обновлять текст TextView.

Листинг 2.8. Подключение новой кнопки (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ---

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

        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
        ---
        mFalseButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(QuizActivity.this,
                    R.string.correct_toast,
                    Toast.LENGTH_SHORT).show();
            }
        });

        mNextButton = (Button)findViewById(R.id.next_button);
        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
```

```

        mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
    }
});
---
}
}

```

Обновление переменной `mQuestionTextView` осуществляется в двух разных местах. Лучше выделить этот код в закрытый метод, как показано в листинге 2.9. Далее остается лишь вызвать этот метод в слушателе `mNextButton` и в конце `onCreate(Bundle)` для исходного заполнения текста в представлении активности.

Листинг 2.9. Инкапсуляция в методе `updateQuestion()` (`QuizActivity.java`)

```

public class QuizActivity extends AppCompatActivity {
    ---
    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ---
        mQuestionTextView = (TextView)findViewById(R.id.question_text_view);
        int question = mQuestionBank[mCurrentIndex].getQuestion();
        mQuestionTextView.setText(question);

        mNextButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
                int question = mQuestionBank[mCurrentIndex].getQuestion();
                mQuestionTextView.setText(question);
                updateQuestion();
            }
        });

        updateQuestion();
        ---
    }
}

```

Запустите `GeoQuiz` и протестируйте новую кнопку `Next`.

Итак, с вопросами мы разобрались, пора обратиться к ответам. В текущем состоянии приложение `GeoQuiz` считает, что на все вопросы ответ должен быть отрицательным; исправим этот недостаток. И снова мы реализуем закрытый метод для инкапсуляции кода, вместо того чтобы вставлять одинаковый код в двух местах.

Сигнатура метода, который будет добавлен в QuizActivity, выглядит так:

```
private void checkAnswer(boolean userPressedTrue)
```

Метод получает логическую переменную, которая указывает, какую кнопку нажал пользователь: True или False. Ответ пользователя проверяется по ответу текущего объекта Question. Наконец, после определения правильности ответа метод создает уведомление для вывода соответствующего сообщения.

Включите в файл QuizActivity.java реализацию checkAnswer(boolean), приведенную в листинге 2.10.

Листинг 2.10. Добавление метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private void updateQuestion() {
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
    }

    private void checkAnswer(boolean userPressedTrue) {
        boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();

        int messageResId = 0;

        if (userPressedTrue == answerIsTrue) {
            messageResId = R.string.correct_toast;
        } else {
            messageResId = R.string.incorrect_toast;
        }

        Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
            .show();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

Включите в слушателя кнопки вызов checkAnswer(boolean), как показано в листинге 2.11.

Листинг 2.11. Вызов метода checkAnswer(boolean) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

```
mTrueButton = (Button)findViewById(R.id.true_button);
mTrueButton.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
        R.string.incorrect_toast,
        Toast.LENGTH_SHORT).show();
        checkAnswer(true);
    }
});

mFalseButton = (Button)findViewById(R.id.false_button);
mFalseButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(QuizActivity.this,
        R.string.correct_toast,
        Toast.LENGTH_SHORT).show();
        checkAnswer(false);
    }
});

mNextButton = (Button)findViewById(R.id.next_button);
...
}
```

Программа GeoQuiz снова готова к работе. Давайте запустим ее на реальном устройстве.

Запуск на устройстве

В этом разделе мы займемся настройкой системы, устройства и приложения для выполнения GeoQuiz на физическом устройстве.

Подключение устройства

Прежде всего подключите устройство к системе. Если разработка ведется на Mac, ваша система должна немедленно распознать устройство. В системе Windows может потребоваться установка драйвера *adb* (Android Debug Bridge). Если Windows не может найти драйвер *adb*, загрузите его с сайта производителя устройства.

Настройка устройства для разработки

Чтобы вы могли тестировать приложения на своем устройстве, необходимо разрешить для устройства отладку USB.

- ① На устройствах с Android 4.2 и выше флажок Developer options по умолчанию отображается. Чтобы включить его, откройте меню Settings ▶ About Tablet/Phone и нажмите Build Number семь раз. После этого вернитесь в меню Settings, найдите раздел Developer options и установите флажок USB debugging.
- ① На устройствах с Android 4.0 или 4.1 откройте меню Settings ▶ Developer options.
- ① На устройствах с версиями Android до 4.0 откройте меню Settings ▶ Applications ▶ Development и найдите флажок USB debugging.

Как видите, настройки серьезно различаются между устройствами. Если у вас возникнут проблемы с включением отладки на вашем устройстве, обратитесь за помощью по адресу <http://developer.android.com/tools/device.html>.

Чтобы убедиться в том, что устройство было успешно опознано, откройте режим представления Devices. Для этого проще всего выбрать панель Android в нижней части окна Android Studio. На панели должен находиться раскрывающийся список подключенных устройств (рис. 2.7). В списке должны присутствовать как AVD, так и физическое устройство.

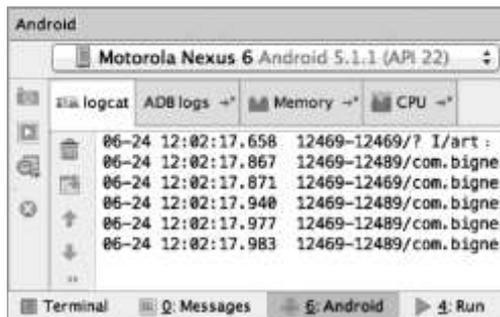


Рис. 2.7. Просмотр списка подключенных устройств

Если у вас возникнут трудности с распознаванием устройства, прежде всего убедитесь в том, что устройство включено и для него включена отладка USB.

Если устройство так и не появилось в режиме представления Devices, дополнительную информацию можно найти на сайте разработчиков Android. Начните со страницы <http://developer.android.com/tools/device.html> или обратитесь на форум книги forums.bignerdranch.com за дополнительной информацией о решении проблемы.

Запустите GeoQuiz так, как это делалось ранее. Android Studio предлагает выбор между запуском на виртуальном устройстве и на физическом устройстве, подключенном к системе. Выберите физическое устройство; GeoQuiz запускается на выбранном устройстве.

Если Android Studio не предлагает выбрать устройство, а GeoQuiz запускается в эмуляторе, проверьте описанную ранее процедуру и убедитесь в том, что устройство подключено. Затем проверьте правильность конфигурации запуска; чтобы изменить параметры конфигурации, выберите раскрывающийся список в верхней части окна (рис. 2.8).



Рис. 2.8. Конфигурации запуска

Выберите в списке строку Edit Configurations; открывается новое окно с подробной информацией о текущей конфигурации (рис. 2.9). Выберите на левой панели строку app и убедитесь в том, что в разделе Target Device установлен переключатель Show chooser dialog. Щелкните на кнопке OK и запустите приложение заново. На этот раз вам будет предложено выбрать устройство для запуска.

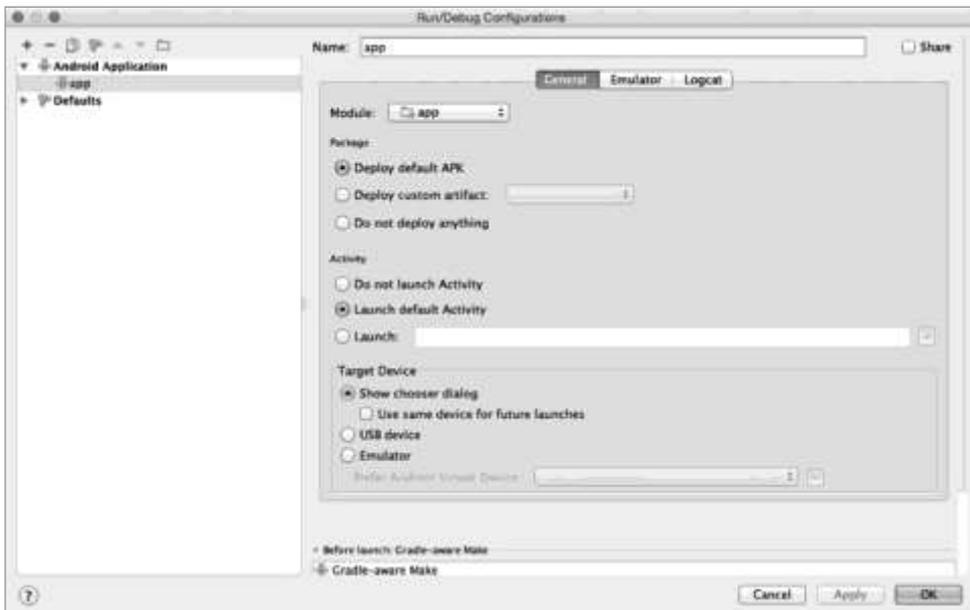


Рис. 2.9. Свойства конфигурации запуска

Добавление значка

Приложение GeoQuiz работает, но пользовательский интерфейс смотрелся бы более привлекательно, если бы на кнопке Next была изображена стрелка, обращенная направо.

Изображение такой стрелки можно найти в файле решений этой книги (<http://www.bignerdranch.com/solutions/AndroidProgramming2e.zip>). Файл решений представляет собой набор проектов Android Studio для всех глав книги.

Загрузите файл и откройте каталог `02_MVC/GeoQuiz/app/src/main/res`. Найдите в нем подкаталоги `drawable-hdpi`, `res/drawable-mdpi`, `drawable-xhdpi` и `drawable-xxhdpi`.

Суффиксы имен каталогов обозначают экранную плотность пикселей устройства.

mdpi	Средняя плотность (~160 dpi)
hdpi	Высокая плотность (~240 dpi)
xhdpi	Сверхвысокая плотность (~320 dpi)
xxhdpi	Сверхсверхвысокая плотность (~480 dpi)

(Также существуют другие категории устройств, включая `ldpi` и `xxxhdpi`, но в решениях они не используются.)

Каждый каталог содержит два графических файла — `arrow_right.png` и `arrow_left.png`. Эти файлы адаптированы для плотности пикселей, указанной в имени каталога.

В полноценных приложениях важно включить в проект графику для разных плотностей пикселей, поскольку это приводит к сокращению артефактов от масштабирования изображений. Вся графика в проекте устанавливается с приложением, а ОС выбирает наиболее подходящий вариант для конкретного устройства.

Мы включим в решения GeoQuiz все файлы изображений. При запуске ОС выбирает файл изображения, наиболее подходящий для конкретного устройства, на котором выполняется приложение. Следует учитывать, что дублирование изображений увеличивает размер приложения. В данном примере это не создает серьезных проблем, потому что GeoQuiz — очень простое приложение.

Если приложение выполняется на устройстве, экранная плотность которого не соответствует ни одному признаку в именах каталогов, Android автоматически масштабирует изображение к подходящему размеру. Благодаря этому обстоятельству не обязательно предоставлять изображения для всех категорий плотности. Для сокращения размера приложения можно сосредоточиться на одной или нескольких категориях высокой плотности и выборочно оптимизировать графику для уменьшенного разрешения, когда автоматическое масштабирование Android создает артефакты на устройствах с низким разрешением.

(Альтернативные варианты дублирования изображений с разной плотностью и каталог `mirrar` рассматриваются в главе 21.)

Добавление ресурсов в проект

Следующим шагом станет включение графических файлов в ресурсы приложения GeoQuiz.

Для начала следует убедиться в том, что проект содержит все необходимые папки `drawable`. Проверьте, что в окне инструментов Project отображается представление Project (выберите в раскрывающемся списке в верхней части окна инструментов Project строку Project, как показано на рис. 1.13 в главе 1). Раскройте содержимое узла `GeoQuiz/app/src/main/res`. В нем должны присутствовать папки с именами `drawable-`

hdpi, drawable-mdpi, drawable-xhdpi и drawable-xxhdpi, изображенные на рис. 2.10. (Также в списке присутствуют другие папки; пока не обращайтесь на них внимания.)



Рис. 2.10. Проверка существования каталогов drawable

Если какие-либо из перечисленных папок drawable отсутствуют, создайте их перед добавлением графических ресурсов. Щелкните правой кнопкой мыши на каталоге res и выберите команду **New** ▶ **Directory**. Введите имя отсутствующего каталога (например, drawable-mdpi) и щелкните на кнопке **OK** (рис. 2.11).

Созданный каталог drawable-mdpi должен появиться в представлении Project окна инструментов Project. (Если новый каталог не виден, вероятно, вы работаете в режиме представления Android. Переключитесь в режим Project так, как описано выше.)

Повторите описанные действия для создания каталогов drawable-hdpi, drawable-xhdpi и drawable-xxhdpi.

Когда все каталоги будут созданы, для каждого каталога drawable в файле решений скопируйте файлы arrow_left.png и arrow_right.png и вставьте их в соответствующий каталог drawable проекта.

После того как все изображения будут скопированы, файлы arrow_left.png и arrow_right.png появляются в окне инструментов Project (рис. 2.12).



Рис. 2.11. Создание каталога drawable

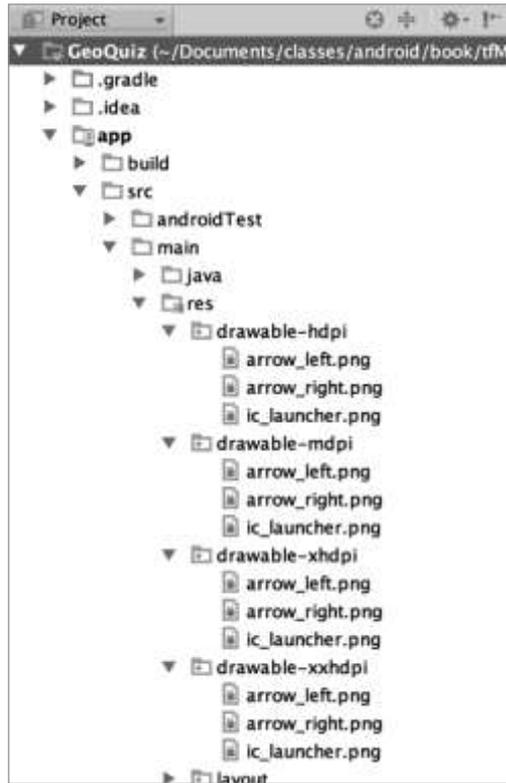


Рис. 2.12. Изображения стрелок в каталогах drawable проекта GeoQuiz

Переключив окно инструментов Project обратно в режим Android, вы увидите сводку добавленных файлов (рис. 2.13).

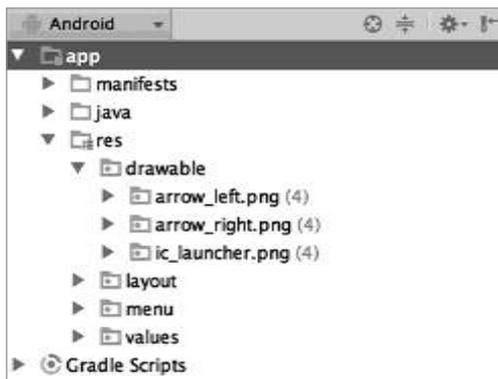


Рис. 2.13. Сводка изображений со стрелками в каталогах drawable приложения GeoQuiz

Процесс включения графики в приложение чрезвычайно прост. Любому файлу .png, .jpg или .gif, добавленному в папку res/drawable, автоматически назначается идентификатор ресурса. (Учтите, что имена файлов должны быть записаны в нижнем регистре и не могут содержать пробелов.)

Эти идентификаторы ресурсов не уточняются плотностью пикселей, так что вам не нужно определять плотность пикселей экрана во время выполнения; просто используйте идентификатор ресурса в коде. При запуске приложения ОС автоматически выберет изображение, подходящее для конкретного устройства.

Система ресурсов Android более подробно рассматривается в главе 3 и далее. А пока давайте заставим работать нашу стрелку.

Ссылки на ресурсы в XML

Для ссылок на ресурсы в коде используются идентификаторы ресурсов. Но мы хотим настроить кнопку Next так, чтобы в определении макета отображалась стрелка. Как включить ссылку на ресурс в разметку XML?

Да почти так же, только с немного измененным синтаксисом. Откройте файл activity_quiz.xml и добавьте два атрибута в определение виджета Button.

Листинг 2.12. Включение графики в кнопку Next (activity_quiz.xml)

```
<LinearLayout
  ... >
  ...

  <LinearLayout
    ... >
    ...
  </LinearLayout>

  <Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"
  />

</LinearLayout>
```

В ресурсах XML вы ссылаетесь на другой ресурс по его типу и имени. Ссылка на строку начинается с префикса @string/. Ссылка на графический ресурс начинается с префикса @drawable/.

Имена ресурсов и структура каталогов res более подробно рассматриваются начиная с главы 3.

Сохраните приложение GeoQuiz, запустите его и насладитесь новым внешним видом кнопки. Протестируйте ее и убедитесь, что кнопка работает точно так же, как прежде.

Впрочем, в программе GeoQuiz скрывается ошибка. Во время выполнения GeoQuiz нажмите кнопку Next, чтобы перейти к следующему вопросу, а затем поверните устройство. (Если программа выполняется в эмуляторе, нажмите Fn+Control+F12/Ctrl+F12.)

После поворота мы снова видим первый вопрос. Почему это произошло и как исправить ошибку?

Ответы на эти вопросы связаны с жизненным циклом активности, — этой теме посвящена глава 3.

Упражнения

Упражнения в конце главы предназначены для самостоятельной работы. Некоторые из них просты и рассчитаны на повторение материала главы. Другие, более сложные упражнения требуют логического мышления и навыков решения задач.

Трудно переоценить важность упражнений. Они закрепляют усвоенный материал, повышают уверенность в обретенных навыках и сокращают разрыв между изучением программирования Android и умением самостоятельно писать программы для Android.

Если у вас возникнут затруднения с каким-либо упражнением, сделайте перерыв, потом вернитесь и попробуйте еще раз. Если и эта попытка окажется безуспешной, обратитесь на форум книги forums.bignerdranch.com. Здесь вы сможете просмотреть вопросы и решения, полученные от других читателей, а также опубликовать свои вопросы и решения.

Чтобы избежать случайного повреждения текущего проекта, мы рекомендуем создать копию и практиковаться на ней.

В файловом менеджере компьютера перейдите в корневой каталог своего проекта. Скопируйте папку GeoQuiz и вставьте новую копию рядом с оригиналом (в OS X воспользуйтесь функцией дублирования). Переименуйте новую папку и присвойте ей имя GeoQuiz Challenge. В Android Studio выполните команду File ▶ Import Project.... В окне импорта перейдите к папке GeoQuiz Challenge и нажмите кнопку OK. Скопированный проект появляется в новом окне готовым к работе.

Упражнение. Добавление слушателя для TextView

Кнопка Next удобна, но было бы неплохо сделать так, чтобы пользователь мог перейти к следующему вопросу простым нажатием на виджете TextView.

Подсказка. Для TextView можно использовать слушателя View.OnClickListener, который использовался с Button, потому что класс TextView также является производным от View.

Упражнение. Добавление кнопки возврата

Добавьте кнопку для возвращения к предыдущему вопросу. Пользовательский интерфейс должен выглядеть примерно так, как показано на рис. 2.14.

Это очень полезное упражнение. В нем вам придется вспомнить многое из того, о чем говорилось в двух предыдущих главах.

Упражнение. От Button к ImageButton

Возможно, пользовательский интерфейс будет смотреться еще лучше, если на кнопках будут отображаться *только* значки, как на рис. 2.15.



Рис. 2.14. Теперь с кнопкой возврата!

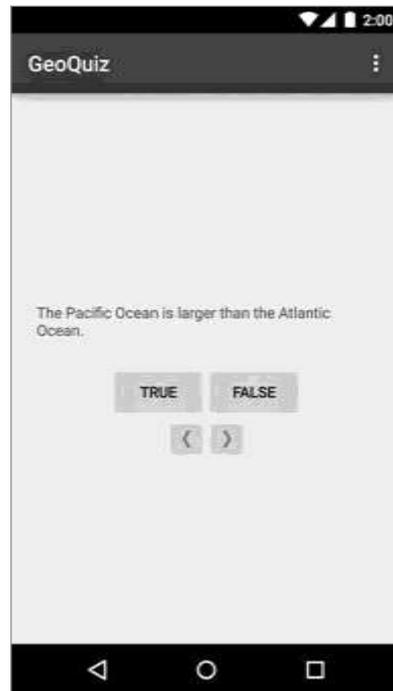


Рис. 2.15. Кнопки только со значками

Для этого оба виджета должны относиться к типу `ImageButton` (вместо обычного `Button`).

Виджет `ImageButton` является производным от `ImageView` — в отличие от виджета `Button`, производного от `TextView`. Диаграммы их наследования изображены на рис. 2.16.

Атрибуты `text` и `drawable` кнопки `Next` можно заменить одним атрибутом `ImageView`:

```
<Button ImageButton  
  android:id="@+id/next_button"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:text="@string/next_question_button"  
  android:drawableRight="@drawable/arrow_right"  
  android:drawablePadding="4dp"  
  android:src="@drawable/arrow_right"  
>
```

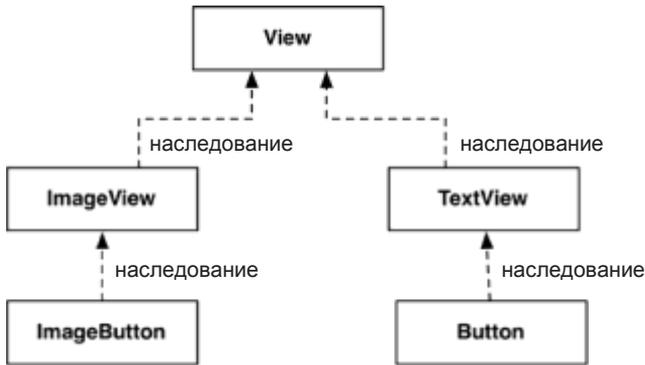


Рис. 2.16. Диаграмма наследования ImageButton и Button

Конечно, вам также придется внести изменения в QuizActivity, чтобы этот класс работал с ImageButton.

После того как вы замените кнопки кнопками ImageButton, Android Studio выдаст предупреждение об отсутствии атрибута android:contentDescription. Этот атрибут обеспечивает доступность контента для читателей с ослабленным зрением. Строка, заданная этому атрибуту, читается экранным диктором (при включении соответствующих настроек в системе пользователя).

Наконец, добавьте атрибут android:contentDescription в каждый элемент ImageButton.

1.3. Жизненный цикл активности

У каждого экземпляра Activity имеется жизненный цикл. Во время этого жизненного цикла активность переходит между тремя возможными состояниями: выполнение, приостановка и остановка. Для каждого перехода у Activity существует метод, который оповещает активность об изменении состояния. На рис. 3.1 изображен жизненный цикл активности, состояния и методы.



Рис. 3.1. Диаграмма состояний Activity

Субклассы `Activity` могут использовать методы, представленные на рис. 3.1, для выполнения необходимых действий во время критических переходов жизненного цикла активности.

Один из таких методов вам уже знаком — это метод `onCreate(Bundle)`. ОС вызывает этот метод после создания экземпляра активности, но до его отображения на экране.

Как правило, активность переопределяет `onCreate(...)` для подготовки пользовательского интерфейса:

- ① заполнение виджетов и их вывод на экран (вызов `setContentView(int)`);
- ① получение ссылок на заполненные виджеты;
- ① назначение слушателей виджетам для обработки взаимодействия с пользователем;
- ① подключение к внешним данным модели.

Важно понимать, что вы никогда не вызываете `onCreate(...)` или другие методы жизненного цикла `Activity` в своих приложениях: вы переопределяете их в субклассах активности, а Android вызывает их в нужный момент времени.

Регистрация событий жизненного цикла Activity

В этом разделе мы переопределим методы жизненного цикла, чтобы отслеживать основные переходы жизненного цикла `QuizActivity`. Реализации ограничиваются регистрацией в журнале сообщения о вызове метода.

Создание сообщений в журнале

В Android класс `android.util.Log` отправляет журнальные сообщения в общий журнал системного уровня. Класс `Log` предоставляет несколько методов регистрации сообщений. Следующий метод чаще всего встречается в этой книге:

```
public static int d(String tag, String msg)
```

Имя «d» означает «debug» (отладка) и относится к уровню регистрации сообщений. (Уровни `Log` более подробно рассматриваются в последнем разделе этой главы.) Первый параметр определяет источник сообщения, а второй — его содержимое.

Первая строка обычно содержит константу TAG, значением которой является имя класса. Это позволяет легко определить источник конкретного сообщения.

В файле `QuizActivity.java` добавьте константу TAG в `QuizActivity`.

Листинг 3.1. Добавление константы TAG (`QuizActivity.java`)

```
public class QuizActivity extends AppCompatActivity {
    private static final String TAG = "QuizActivity";
    ...
}
```

Включите в `onCreate(...)` вызов `Log.d(...)` для регистрации сообщения.

Листинг 3.2. Включение команды регистрации сообщения в onCreate(...) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate(Bundle) called");
        setContentView(R.layout.activity_quiz);
        ...
    }
}
```

Теперь переопределите еще пять методов в QuizActivity; для этого добавьте следующие определения после onCreate(Bundle), но до onCreateOptionsMenu(Menu):

Листинг 3.3. Переопределение методов жизненного цикла (QuizActivity.java)

```
    @Override
    public void onStart() {
        super.onStart();
        Log.d(TAG, "onStart() called");
    }

    @Override
    public void onPause() {
        super.onPause();
        Log.d(TAG, "onPause() called");
    }

    @Override
    public void onResume() {
        super.onResume();
        Log.d(TAG, "onResume() called");
    }

    @Override
    public void onStop() {
        super.onStop();
        Log.d(TAG, "onStop() called");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(TAG, "onDestroy() called");
    }
    ...
}
```

Обратите внимание на вызовы реализаций суперкласса перед регистрацией сообщений. Эти вызовы являются обязательными, причем в onCreate(...) реализация

суперкласса должна вызываться до выполнения каких-либо критических операций; в других методах порядок менее важен.

Возможно, вы заметили аннотацию `@Override`. Она приказывает компилятору проследить за тем, чтобы класс действительно содержал переопределяемый метод. Например, компилятор сможет предупредить вас об опечатке в имени метода:

```
public class QuizActivity extends AppCompatActivity {  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_quiz);  
    }  
}
```

...

Класс `Activity` не содержит метод `onCreat(Bundle)`, поэтому компилятор выдает предупреждение. Это позволит вам исправить опечатку, вместо того чтобы случайно реализовать `QuizActivity.onCreat(Bundle)`.

Использование LogCat

Чтобы просмотреть системный журнал во время работы приложения, используйте `LogCat` — программу, включенную в инструментарий `Android SDK`.

При запуске `GeoQuiz` данные `LogCat` появляются в нижней части окна `Android Studio` (рис. 3.2). Если панель `LogCat` не видна, выберите панель `Android` в нижней части окна и перейдите на вкладку `Devices|logcat`.

Запустите `GeoQuiz`; на панели `LogCat` начнут быстро появляться сообщения. По умолчанию выводятся журнальные сообщения, сгенерированные с именем пакета вашего приложения. Вы увидите сообщения своей программы, а также некоторые системные сообщения.

Чтобы упростить поиск сообщений, можно отфильтровать вывод по константе `TAG`. В `LogCat` щелкните на раскрывающемся списке фильтра в правом верхнем углу панели. Обратите внимание на существующий фильтр, настроенный на вывод сообщений только от вашего приложения. Если выбрать вариант `No Filters`, будут выводиться журнальные сообщения, сгенерированные в масштабах всей системы.

Выберите в раскрывающемся списке пункт `Edit Filter Configuration`. Нажмите кнопку `+` для создания нового фильтра. Введите имя фильтра `QuizActivity` и введите строку `QuizActivity` в поле `by Log Tag`: (рис. 3.3).

Щелкните на кнопке `OK`; после этого будут видны только сообщения с тегом `QuizActivity` (рис. 3.4). Как видите, после запуска `GeoQuiz` были вызваны три метода жизненного цикла и был создан исходный экземпляр `QuizActivity`.

(Если вы не видите отфильтрованного списка, выберите фильтр `QuizActivity` на левой панели `LogCat`.)

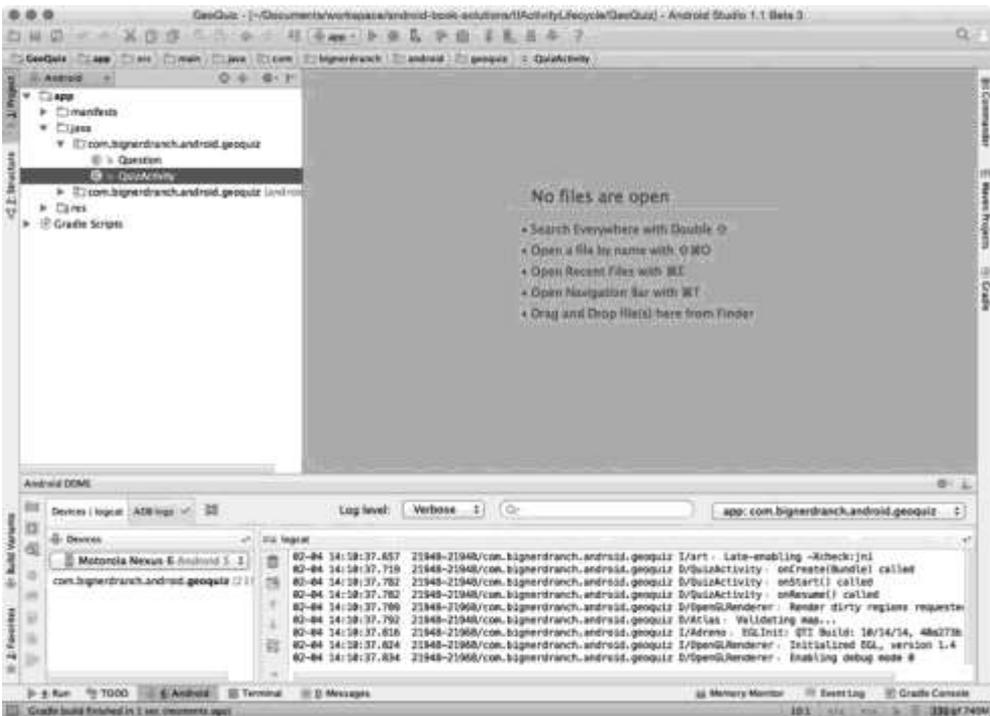


Рис. 3.2. Android Studio с выводом LogCat

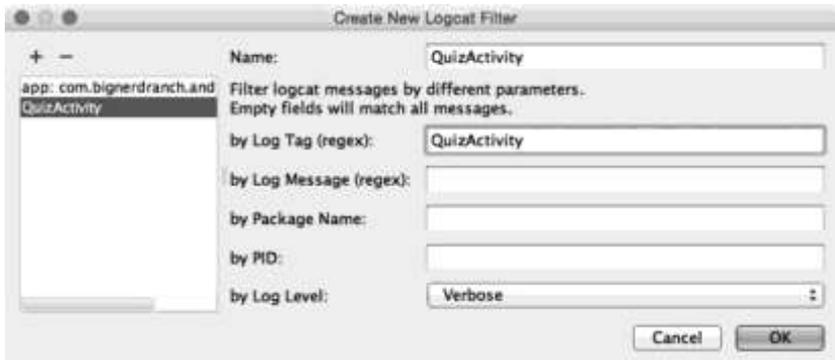


Рис. 3.3. Создание фильтра в LogCat

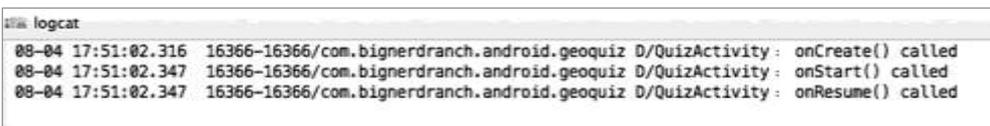


Рис. 3.4. При запуске GeoQuiz происходит создание, запуск и продолжение активности

А теперь немного поэкспериментируем. Нажмите на устройстве кнопку Back, а затем проверьте вывод LogCat. Активность приложения получила вызовы onPause(), onStop() и onDestroy() (рис. 3.5).

```
logcat
08-04 17:51:02.316 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:51:02.347 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 17:54:35.463 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 17:54:35.811 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
08-04 17:54:35.811 16366-16366/com.bignerdranch.android.geoquiz D/QuizActivity: onDestroy() called
```

Рис. 3.5. Нажатие кнопки Back приводит к уничтожению активности

Нажимая кнопку Back, вы сообщаете Android: «Я завершил работу с активностью, и она мне больше не нужна». Android уничтожает активность, чтобы избежать неэффективного расходования ограниченных ресурсов устройства.

Перезапустите приложение GeoQuiz. Нажмите кнопку Home и проверьте вывод LogCat. Ваша активность получила вызовы onPause() и onStop(), но не вызов onDestroy() (рис. 3.6).

```
logcat
08-04 17:56:05.477 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 17:56:05.533 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 17:56:10.851 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 17:56:11.191 18475-18475/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
```

Рис. 3.6. Нажатие кнопки Back приводит к уничтожению активности

Вызовите на устройстве диспетчер задач. На новых устройствах для этого следует нажать кнопку Recents рядом с кнопкой Home (рис. 3.7). На устройствах без кнопки Recents выполните долгое нажатие кнопки Home.

В диспетчере задач нажмите на приложении GeoQuiz и проверьте вывод LogCat. Активность запускается и продолжает работу, но создавать ее не нужно.

Нажатие кнопки Home сообщает Android: «Я сейчас займусь другим делом, но потом могу вернуться». Android приостанавливает активность, но старается не уничтожать ее на случай возвращения.

Тем не менее существование остановленной активности не гарантировано. Если системе потребуется занятая память, она уничтожает остановленные активности.

Наконец, представьте маленькое временное окно, которое только частично закрывает активность. При появлении такого окна находящаяся за ним активность приостанавливается, и осуществить взаимодействие с ней невозможно. Выполнение активности будет продолжено, когда временное окно будет закрыто.

Далее в этой книге мы будем переопределять различные методы жизненного цикла активности для решения реальных задач. При этом применение каждого метода будет рассматриваться более подробно.



Рис. 3.7. Кнопки Home, Back и Recents

Повороты и жизненный цикл активности

А теперь вернемся к ошибке, обнаруженной в конце главы 2. Запустите GeoQuiz, нажмите кнопку Next для перехода к следующему вопросу, а затем поверните устройство. (Чтобы имитировать поворот в эмуляторе, нажмите Fn+Control+F12/Ctrl+F12.)

После поворота GeoQuiz снова выводит первый вопрос. Чтобы понять, почему это произошло, просмотрите вывод LogCat. Он выглядит примерно так, как показано на рис. 3.8.

```
logcat
08-04 18:01:32.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:01:32.555 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
08-04 18:08:53.557 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onPause() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStop() called
08-04 18:08:53.558 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onDestroy() called
08-04 18:08:53.585 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onCreate() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onStart() called
08-04 18:08:53.605 20706-20706/com.bignerdranch.android.geoquiz D/QuizActivity: onResume() called
```

Рис. 3.8. QuizActivity умирает и возрождается

Когда вы поворачиваете устройство, экземпляр QuizActivity, который вы видели, уничтожается, и вместо него создается новый экземпляр. Снова поверните устройство — происходит еще один цикл уничтожения и возрождения.

Из-за этого и возникает ошибка. При каждом создании нового экземпляра QuizActivity переменная `mCurrentIndex` инициализируется 0, а пользователь начинает с первого вопроса. Вскоре мы исправим ошибку, но сначала повнимательнее разберемся, почему это происходит.

Конфигурации устройств и альтернативные ресурсы

Поворот приводит к изменению *конфигурации устройства*. Конфигурация устройства представляет собой набор характеристик, описывающих текущее состояние конкретного устройства. К числу характеристик, определяющих конфигурацию, относится ориентация экрана, плотность пикселей, размер экрана, тип клавиатуры, режим стыковки, язык и многое другое.

Как правило, приложения предоставляют альтернативные ресурсы для разных конфигураций устройств. Пример такого рода нам уже встречался: вспомните, как мы включали в проект несколько изображений стрелки для разной плотности пикселей.

Плотность пикселей является фиксированным компонентом конфигурации устройства; она не может измениться во время выполнения. Напротив, некоторые компоненты (такие, как ориентация) *могут* изменяться при выполнении.

При изменении конфигурации во время выполнения может оказаться, что приложение содержит ресурсы, лучше подходящие для новой конфигурации. Чтобы увидеть, как работает этот механизм, мы создадим альтернативный ресурс, который Android найдет и использует при изменении ориентации экрана.

Создание макета для альбомной ориентации

В окне инструментов Project щелкните правой кнопкой мыши на каталоге `res` и выберите команду `New ▶ Android resource directory`. Открывается окно со списками типов ресурсов и квалификаторами этих типов (наподобие изображенного на рис. 3.9). Выберите в раскрывающемся списке `Resource type` строку `layout`. Оставьте в списке `Source set` значение `main`.

Теперь необходимо выбрать способ уточнения ресурсов макета. Выберите в списке `Available qualifiers` строку `Orientation` и щелкните на кнопке `>>`, чтобы переместить значение `Orientation` в поле `Chosen qualifiers`.

Наконец, убедитесь в том, что в списке `Screen Orientation` выбрано значение `Landscape` (рис. 3.10). Проверьте, что в поле `Directory name` теперь указано имя каталога `layout-land`. Хотя окно выглядит нетривиально, его цель — задать имя каталога. Щелкните на кнопке `OK`; Android Studio создает папку `res/layout-land/`.

Суффикс `-land` — еще один пример конфигурационного квалификатора. По конфигурационным квалификаторам подкаталогов `res` Android определяет, какие ресурсы лучше всего подходят для текущей конфигурации устройства. Список конфигурационных квалификаторов, поддерживаемых Android, и обозначаемых

ими компонентов конфигурации устройств находится по адресу <http://developer.android.com/guide/topics/resources/providing-resources.html>.



Рис. 3.9. Создание нового каталога ресурсов

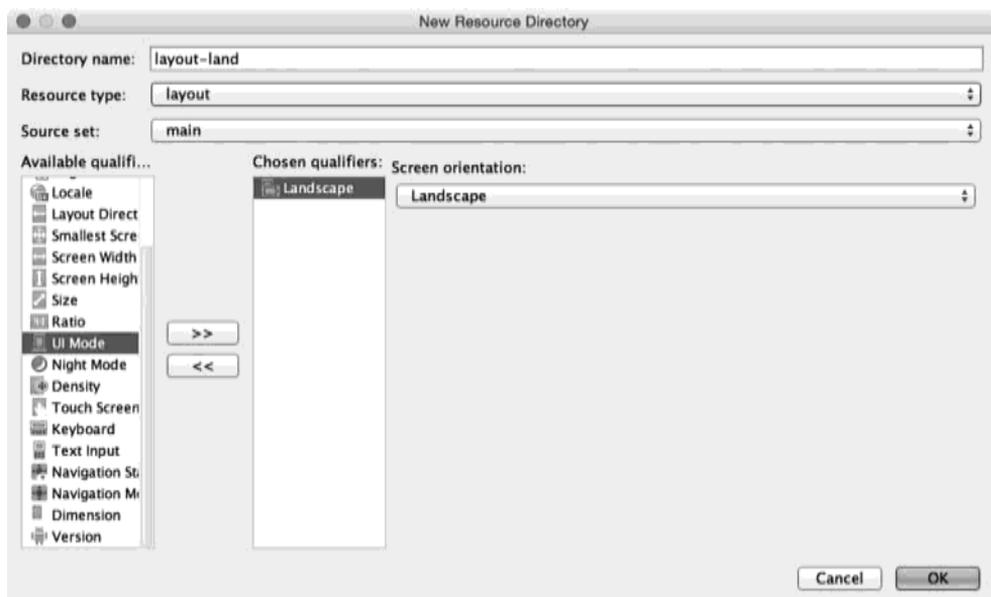


Рис. 3.10. Создание каталога res/layout-land/

Когда устройство находится в альбомной ориентации, Android находит и использует ресурсы в каталоге `res/layout-land`. В противном случае используются ресурсы по умолчанию из каталога `res/layout/`. Впрочем, на данный момент каталог `res/layout-land` не содержит ресурсов; этот недостаток нужно исправить.

Скопируйте файл `activity_quiz.xml` из `res/layout/` в `res/layout-land/`. Теперь в приложении имеются два макета: макет для альбомной ориентации и макет по умолчанию. Оставьте имя файла без изменения. Два файла макетов должны иметь одинаковые имена, чтобы на них можно было ссылаться по одному идентификатору ресурса.

Теперь внесем некоторые изменения в альбомный макет, чтобы он отличался от макета по умолчанию. Сводка этих изменений представлена на рис. 3.11.

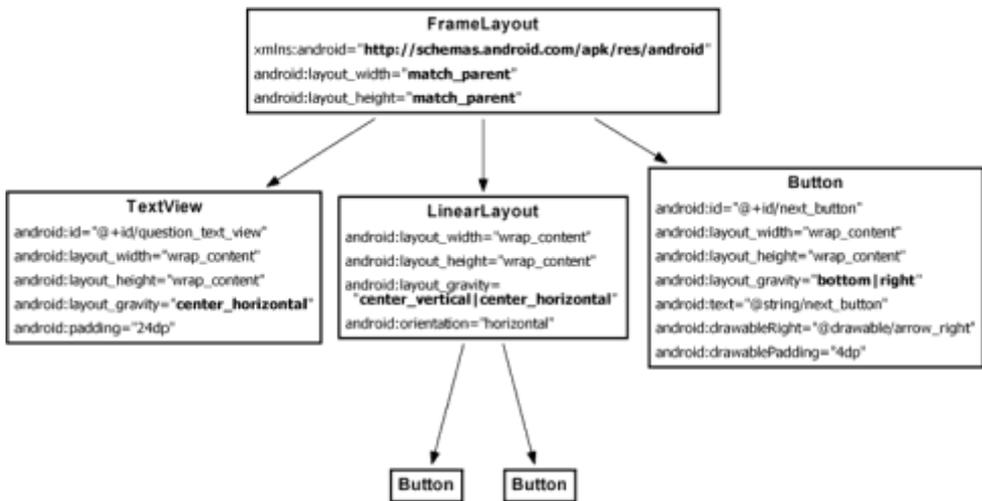


Рис. 3.11. Альтернативный макет для альбомной ориентации

Вместо `LinearLayout` будет использоваться `FrameLayout` — простейшая разновидность `ViewGroup` без определенного способа размещения потомков. В этом макете дочерние представления будут размещаться в соответствии с атрибутом `android:layout_gravity`.

Атрибут `android:layout_gravity` необходим виджетам `TextView`, `LinearLayout` и `Button`. Потомки `Button` виджета `LinearLayout` остаются без изменений.

Откройте файл `layout-land/activity_quiz.xml` и внесите необходимые изменения, руководствуясь рис. 3.11. Проверьте результат своей работы по листингу 3.4.

Листинг 3.4. Настройка альбомного макета (`layout-land/activity_quiz.xml`)

```

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical">
  
```

```

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

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical|center_horizontal"
        android:orientation="horizontal" >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|right"
        android:text="@string/next_button"
        android:drawableRight="@drawable/arrow_right"
        android:drawablePadding="4dp"
    />

</LinearLayout>
</FrameLayout>

```

Снова запустите GeoQuiz. Поверните устройство в альбомную ориентацию, чтобы увидеть новый макет (рис. 3.12). Конечно, в программе используется не только новый макет, но и новый экземпляр QuizActivity.



Рис. 3.12. QuizActivity в альбомной ориентации

Поверните устройство, чтобы переключиться в книжную ориентацию. Вы увидите новый макет — и новый экземпляр QuizActivity.

Android выбирает наиболее подходящий ресурс за вас, но для этого он создает новую активность с нуля. Чтобы класс QuizActivity вывел новый макет, необходимо снова вызвать метод `setContentView(R.layout.activity_quiz)`. А это не произойдет без повторного вызова `QuizActivity.onCreate(...)`. Соответственно Android при повороте уничтожает текущий экземпляр QuizActivity и начинает все заново, чтобы обеспечить оптимальный подбор ресурсов для новой конфигурации.

Учтите, что Android уничтожает текущую активность и создает новую при каждом изменении конфигурации времени выполнения. Также во время выполнения могут происходить и другие изменения (например, изменение языка или доступности клавиатуры), но изменение в ориентации экрана является самым частым.

Сохранение данных между поворотами

Android очень старается предоставить альтернативные ресурсы в нужный момент. Тем не менее уничтожение и повторное создание активностей при поворотах может создать проблемы, как, например, в случае с возвратом к первому вопросу в приложении GeoQuiz.

Чтобы исправить эту ошибку, экземпляр QuizActivity, созданный после поворота, должен знать старое значение `mCurrentIndex`. Нам необходим механизм сохранения данных при изменении конфигурации времени выполнения (например, при поворотах). Одно из возможных решений заключается в переопределении метода `Activity`:

```
protected void onSaveInstanceState(Bundle outState)
```

Обычно этот метод вызывается системой перед `onPause()`, `onStop()` и `onDestroy()`.

Реализация по умолчанию `onSaveInstanceState(...)` приказывает всем представлениям активности сохранить свое состояние в данных объекта Bundle — структуры, связывающей строковые ключи со значениями некоторых ограниченных типов.

Тип Bundle нам уже встречался. Он передается методу `onCreate(Bundle)`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...
}
```

При переопределении `onCreate(...)` вы вызываете реализацию `onCreate(...)` суперкласса активности и передаете ей только что полученный объект Bundle. В реализации суперкласса сохраненное состояние представлений извлекается и используется для воссоздания иерархии представлений активности.

Переопределение onSaveInstanceState(Bundle)

Метод onSaveInstanceState(...) можно переопределить так, чтобы он сохранял дополнительные данные в Bundle, а затем снова загружал их в onCreate(...). Именно так мы организуем сохранение значения mCurrentIndex между поворотами.

Для начала добавьте в QuizActivity.java константу, которая станет ключом в сохраняемой паре «ключ-значение».

Листинг 3.5. Добавление ключа для сохраняемого значения (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";

    Button mTrueButton;
    ...
}
```

Теперь переопределим onSaveInstanceState(...) для записи значения mCurrentIndex в Bundle с использованием константы в качестве ключа.

Листинг 3.6. Переопределение onSaveInstanceState(...) (QuizActivity.java)

```
mNextButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
        updateQuestion();
    }
});

updateQuestion();
}

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    Log.i(TAG, "onSaveInstanceState");
    savedInstanceState.putInt(KEY_INDEX, mCurrentIndex);
}
```

Наконец, в методе onCreate(...) следует проверить это значение, и если оно присутствует, присвоить его mCurrentIndex.

Листинг 3.7. Проверка сохраненных данных в onCreate(...) (QuizActivity.java)

```
...
if (savedInstanceState != null) {
    mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
}

updateQuestion();
}
```

Запустите GeoQuiz и нажмите кнопку Next. Сколько бы поворотов устройства вы ни выполнили, вновь созданный экземпляр QuizActivity «вспоминает» текущий вопрос.

Учтите, что для сохранения и восстановления из Bundle подходят примитивные типы и объекты, реализующие интерфейс Serializable или Parcelable. Впрочем, обычно сохранение объектов пользовательских типов в Bundle считается нежелательным, потому что данные могут потерять актуальность к моменту их извлечения. Лучше организовать для данных другой тип хранилища и сохранить в Bundle примитивный идентификатор.

Реализацию onSaveInstanceState(...) желательно протестировать, особенно если вы сохраняете и восстанавливаете объекты. Повороты тестируются легко; стесированием ситуаций нехватки памяти дело обстоит сложнее. В конце этой главы приведена информация о том, как имитировать уничтожение вашей активности системой Android для освобождения памяти.

Снова о жизненном цикле Activity

Переопределение onSaveInstanceState(Bundle) используется не только при поворотах. Активность также может уничтожаться и в том случае, если пользователь займется другими делами, а Android понадобится освободить память.

Android никогда не уничтожает для освобождения памяти выполняемую активность. Чтобы активность была уничтожена, она должна находиться в приостановленном или остановленном состоянии. Если активность приостановлена или остановлена, значит, был вызван ее метод onSaveInstanceState(...).

При вызове onSaveInstanceState(...) данные сохраняются в объекте Bundle. Операционная система заносит объект Bundle в *запись активности* (activity record).

Чтобы понять, что собой представляет запись активности, добавим *сохраненное* состояние на схему жизненного цикла активности (рис. 3.13).

Когда ваша активность сохранена, объект Activity не существует, но объект записи активности «живет» в ОС. При необходимости операционная система может воссоздать активность по записи активности.

Следует учесть, что активность может перейти в сохраненное состояние без вызова onDestroy(). Однако вы всегда можете рассчитывать на то, что методы onPause() и onSaveInstanceState(...) были вызваны. Обычно метод onSaveInstanceState(...) переопределяется для сохранения небольших, временных состояний текущей активности данных в Bundle, а onPause() — для всех прочих операций.

В некоторых ситуациях Android не только уничтожает активность, но и полностью завершает процесс приложения. Такая ситуация может возникнуть только в том случае, если пользователь не смотрит на приложение, но она возможна (и вполне реальна). Даже в этом случае запись активности продолжает существовать и позволяет быстро перезапустить активность при возвращении пользователя.



Рис. 3.13. Полный жизненный цикл активности

Когда же запись приложения пропадает? Когда пользователь нажимает кнопку Back, активность уничтожается — раз и навсегда. При этом запись активности теряется. Записи активности также обычно уничтожаются при перезагрузке; также возможно их уничтожение в том случае, если они слишком долго не используются.

Для любознательных: тестирование onSaveInstanceState(Bundle)

Переопределяя onSaveInstanceState(Bundle), необходимо убедиться в том, что состояние сохраняется и восстанавливается так, как предполагалось. Это легко делается в эмуляторе.

Запустите виртуальное устройство. В списке приложений на устройстве найдите приложение Settings (рис. 3.14). Оно присутствует в большинстве образов системы, используемых в эмуляторе.

Запустите приложение Settings и выберите категорию Development options. В нее входит много разных настроек; установите флажок Don't keep activities (рис. 3.15).

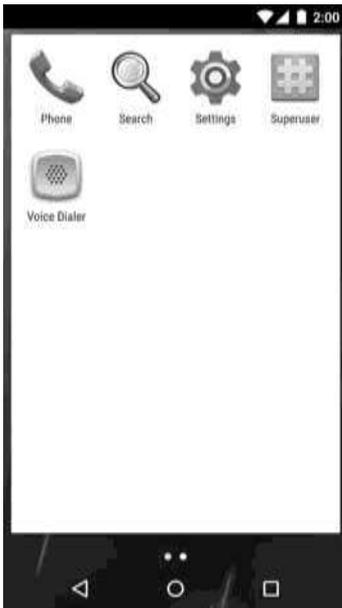


Рис. 3.14. Приложение Settings

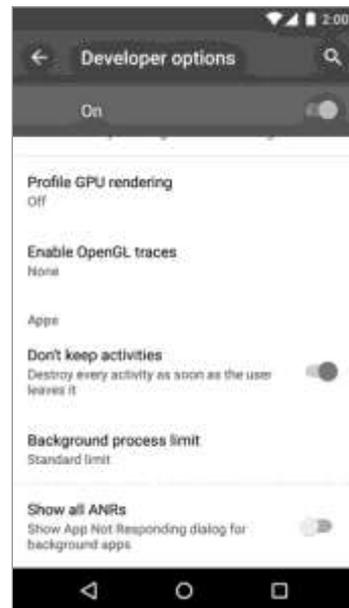


Рис. 3.15. Запрет сохранения активностей

Запустите приложение и нажмите кнопку Home. Это приведет к приостановке и остановке активности. Затем остановленная активность будет уничтожена так, как если бы ОС Android решила освободить занимаемую ею память. Восстановите приложение и посмотрите, было ли состояние сохранено так, как ожидалось. *Обязательно отключите этот режим после завершения тестирования, так как он приводит к снижению быстродействия и некорректному поведению некоторых приложений.*

Нажатие кнопки Back (вместо Home) всегда приводит к уничтожению активности независимо от того, установлен флажок в настройках разработчика или нет. Нажатие кнопки Back сообщает ОС, что пользователь завершил работу с активностью.

Чтобы выполнить тот же тест на физическом устройстве, необходимо установить на нем пакет Dev Tools. За дополнительной информацией обращайтесь по адресу <http://developer.android.com/tools/debugging/debugging-devtools.html>.

Для любознательных: методы и уровни регистрации

Когда вы используете класс `android.util.Log` для регистрации сообщений в журнале, вы задаете не только содержимое сообщения, но и *уровень регистрации*, определяющий важность сообщения. Android поддерживает пять уровней реги-

страции (рис. 3.16). Каждому уровню соответствует свой метод класса `Log`. Регистрация данных в журнале сводится к вызову соответствующего метода `Log`.

Уровень регистрации	Метод	Примечания
ERROR	<code>Log.e(...)</code>	Ошибки
WARNING	<code>Log.w(...)</code>	Предупреждения
INFO	<code>Log.i(...)</code>	Информационные сообщения
DEBUG	<code>Log.d(...)</code>	Отладочный вывод (может фильтроваться)
VERBOSE	<code>Log.v(...)</code>	Только для разработчиков!

Рис. 3.16. Методы и уровни регистрации

Каждый метод регистрации существует в двух вариантах: один получает строковый *тег* и строку сообщения, а второй получает эти же аргументы и экземпляр `Throwable`, упрощающий регистрацию информации о конкретном исключении, которое может быть выдано вашим приложением. В листинге 3.8 представлены примеры сигнатуры методов журнала. Для сборки строк сообщений используйте стандартные средства конкатенации строк Java — или `String.format`, если их окажется недостаточно.

Листинг 3.8. Различные способы регистрации в Android

```
// Регистрация сообщения с уровнем отладки "debug"
Log.d(TAG, "Current question index: " + mCurrentIndex);

Question question;
try {
    question = mQuestionBank[mCurrentIndex];
} catch (ArrayIndexOutOfBoundsException ex) {
    // Регистрация сообщения с уровнем отладки "error"
    // вместе с трассировкой стека исключений
    Log.e(TAG, "Index was out of bounds", ex);
}
```

1.4. Исключения и трассировка стека

Вернемся к проблеме с нашим приложением. Чтобы понять, что произошло, разверните панель AndroidDDMS. Прокрутите содержимое LogCat и найдите текст, выделенный красным шрифтом (рис. 4.2). Это стандартный отчет об исключениях Android. Если информация об исключении отсутствует в LogCat, возможно, необходимо изменить фильтры LogCat: выберите в раскрывающемся списке фильтров вариант No Filters. Также можно выбрать в списке Log Level значение Error, при котором отображаются только наиболее критические сообщения.

В отчете приводится исключение верхнего уровня и данные трассировки стека; затем исключение, которое привело к этому исключению, и его трассировка стека; и так далее, пока не будет найдено исключение, не имеющее причины.

Как правило, в написанном вами коде интерес представляет именно последнее исключение. В данном примере это исключение `java.lang.NullPointerException`. Строка непосредственно под именем исключения содержит начало трассировки стека. В ней указываются класс и метод, в котором произошло исключение, а также имя файла и номер строки кода. Щелкните на синей ссылке; Android Studio открывает указанную строку кода.


```

09-03 12:44:08.523 5458-5458/? E/AndroidRuntime: FATAL EXCEPTION: main
Process: com.bignerdranch.android.geoquiz, PID: 5458
java.lang.RuntimeException: Unable to start activity ComponentInfo{com.bignerdranch.android.ge
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2184)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
    at dalvik.system.NativeStart.main(Native Method)
Caused by: java.lang.NullPointerException
    at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:35)
    at com.bignerdranch.android.geoquiz.QuizActivity.onCreate(QuizActivity.java:90)
    at android.app.Activity.performCreate(Activity.java:5231)
    at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1087)
    at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2148)
    at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2233)
    at android.app.ActivityThread.access$800(ActivityThread.java:135)
    at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1196)
    at android.os.Handler.dispatchMessage(Handler.java:102)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invokeNative(Native Method)
    at java.lang.reflect.Method.invoke(Method.java:515) <3 more...>

```

Рис. 4.2. Исключения и трассировка стека в LogCat

В открывшейся строке программа впервые обращается к переменной `mQuestionTextView` в методе `updateQuestion()`. Имя исключения `NullPointerException` подсказывает суть проблемы: переменная не была инициализирована.

Раскомментируйте строку с инициализацией `mQuestionTextView`, чтобы исправить ошибку.

Помните: когда в вашей программе возникают исключения времени выполнения, следует искать последнее исключение в LogCat и первую строку трассировки стека со ссылкой на написанный вами код. Именно здесь возникает проблема, и именно здесь следует искать ответы.

Даже если сбой происходит на неподключенном устройстве, не все потеряно. Устройство сохраняет последние строки, выводимые в журнал. Длина и срок хранения журнала зависят от устройства, но обычно можно рассчитывать на то, что результаты будут храниться минимум 10 минут. Подключите устройство и выберите его на панели Devices. LogCat заполняется данными из сохраненного журнала.

Диагностика ошибок поведения

Проблемы с приложениями не всегда приводят к сбоям — в некоторых случаях приложение просто начинает некорректно работать. Допустим, пользователь нажимает кнопку Next, а в приложении ничего не происходит. Такие ошибки относятся к категории ошибок поведения.

В файле QuizActivity.java внесите изменение в слушателя mNextButton и закоментируйте код, увеличивающий mCurrentIndex.

Листинг 4.2. Из программы исключается важная строка (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            //mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Запустите GeoQuiz и нажмите кнопку Next. Ничего не происходит.

Эта ошибка коварнее предыдущей. Она не приводит к выдаче исключения, поэтому исправление ошибки не сводится к простому устранению исключения. Кроме того, некорректное поведение может быть вызвано разными причинами: то ли в программе не изменяется индекс, то ли не вызывается метод updateQuestion().

Если вы не знаете причину происходящего, необходимо ее выяснить. Далее мы покажем два основных приема диагностики: сохранение трассировки стека и использование отладчика для назначения точки прерывания.

Сохранение трассировки стека

Включите команду сохранения отладочного вывода в метод updateQuestion() класса QuizActivity:

Листинг 4.3. Использование Exception

```
public class QuizActivity extends AppCompatActivity {
    ...

    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
            new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
    }
}
```

Версия Log.d с сигнатурой Log.d(String, String, Throwable) регистрирует в журнале все данные трассировки стека — как уже встречавшееся ранее исключение AndroidRuntime. По данным трассировки стека вы сможете определить, в какой момент произошел вызов updateQuestion().

При вызове Log.d(...) вовсе не обязательно передавать перехваченное исключение. Вы можете создать новый экземпляр Exception() и передать его методу без инициализации исключения. В журнале будет сохранена информация о том, где было создано исключение.

Запустите приложение GeoQuiz, нажмите кнопку Next и проверьте вывод в LogCat (рис. 4.3).

```
09-04 12:47:37.733 30612-30612/com.bignerdranch.android.geoquiz D/QuizActivity: Updating question text
java.lang.Exception
    at com.bignerdranch.android.geoquiz.QuizActivity.updateQuestion(QuizActivity.java:34)
    at com.bignerdranch.android.geoquiz.QuizActivity.access$100(QuizActivity.java:12)
    at com.bignerdranch.android.geoquiz.QuizActivity$3.onClick(QuizActivity.java:82)
    at android.view.View.performClick(View.java:4438)
    at android.view.View$PerformClick.run(View.java:18422)
    at android.os.Handler.handleCallback(Handler.java:733)
    at android.os.Handler.dispatchMessage(Handler.java:95)
    at android.os.Looper.loop(Looper.java:136)
    at android.app.ActivityThread.main(ActivityThread.java:5001)
    at java.lang.reflect.Method.invokeNative(Native Method) <1 internal calls>
    at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:785)
    at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:601)
    at dalvik.system.NativeStart.main(Native Method)
```

Рис. 4.3. Результаты

Верхняя строка трассировки стека соответствует строке, в которой в журнале были зарегистрированы данные Exception. Следующая строка показывает, где был вызван updateQuestion(), — в реализации onClick(...). Щелкните на ссылке в этой строке; открывается позиция, в которой была закоментирована строка с увеличением индекса заголовка. Но пока не торопитесь исправлять ошибку; сейчас мы найдем ее повторно при помощи отладчика.

Регистрация в журнале данных трассировки стека — мощный инструмент отладки, но он выводит довольно большой объем данных. Оставьте в программе несколько таких команд, и вскоре вывод LogCat превращается в невразумительную мешанину. Кроме того, по трассировке стека конкуренты могут разобраться, как работает ваш код, и похитить ваши идеи.

С другой стороны, в некоторых ситуациях нужна именно трассировка стека с информацией, показывающей, что делает ваш код. Если вы обратитесь за помощью на сайты <http://stackoverflow.com> или forums.bignerdranch.com, в вопрос часто желательнее включить трассировку стека. Информацию можно скопировать прямо из LogCat.

Прежде чем продолжать, удалите команду log из QuizActivity.java.

Листинг 4.4. Прощай, друг (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...

    public void updateQuestion() {
        Log.d(TAG, "Updating question text for question #" + mCurrentIndex,
            new Exception());
        int question = mQuestionBank[mCurrentIndex].getTextResID();
        mQuestionTextView.setText(question);
    }
}
```

Установка точек прерывания

Попробуем найти ту же ошибку при помощи отладчика среды Android Studio. Мы установим *точку прерывания* в `updateQuestion()`, чтобы увидеть, был ли вызван этот метод. Точка прерывания останавливает выполнение программы в заданной позиции, чтобы вы могли в пошаговом режиме проверить, что происходит далее.

В файле `QuizActivity.java` вернитесь к методу `updateQuestion()`. В первой строке метода щелкните на серой полосе слева от кода. На месте щелчка появляется красный кружок; он обозначает точку прерывания (рис. 4.4).

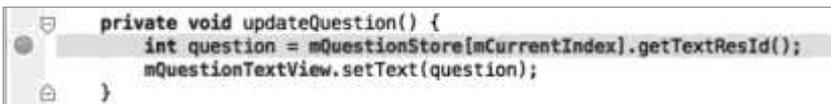


Рис. 4.4. Точка прерывания

Чтобы задействовать отладчик и активизировать точку прерывания, необходимо запустить приложение в отладочном режиме (в отличие от обычного запуска). Для этого щелкните на кнопке отладки (кнопка с зеленым жуком) рядом с кнопкой выполнения. Также можно выполнить команду меню `Run ▶ Debug 'app'`. Устройство сообщает, что оно ожидает подключения отладчика, а затем продолжает работу как обычно.

Когда приложение запустится и заработает под управлением отладчика, его выполнение прерывается. Запуск `GeoQuiz` привел к вызову метода `QuizActivity.onCreate(Bundle)`, который вызвал `updateQuestion()`, что привело к срабатыванию точки прерывания. На рис. 4.5 видно, что редактор открывает файл `QuizActivity.java` и выделяет строку с точкой прерывания, в которой было остановлено выполнение.

В нижней части экрана теперь отображается панель `Debug`; она содержит области `Frames` и `Variables` (рис. 4.6).

Кнопки со стрелками в верхней части панели используются для пошагового выполнения программы. Из трассировки стека видно, что метод `updateQuestion()` был вызван из `onCreate(Bundle)`. Так как нас интересует поведение кнопки `Next`, нажмите кнопку `Resume`, чтобы продолжить выполнение программы. Затем снова нажмите кнопку `Next`, чтобы увидеть, активизируется ли точка прерывания (она должна активизироваться).

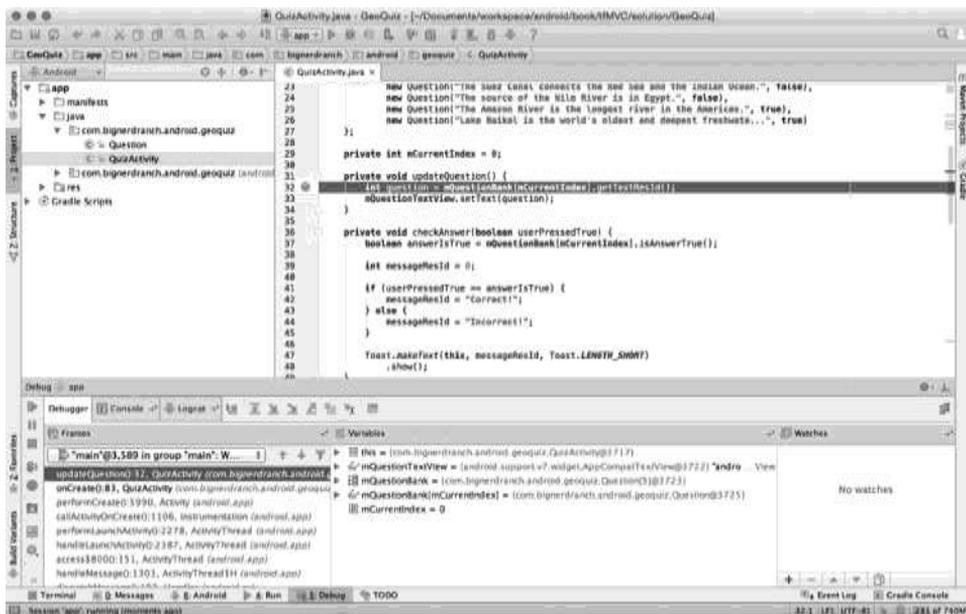


Рис. 4.5. Стоять!

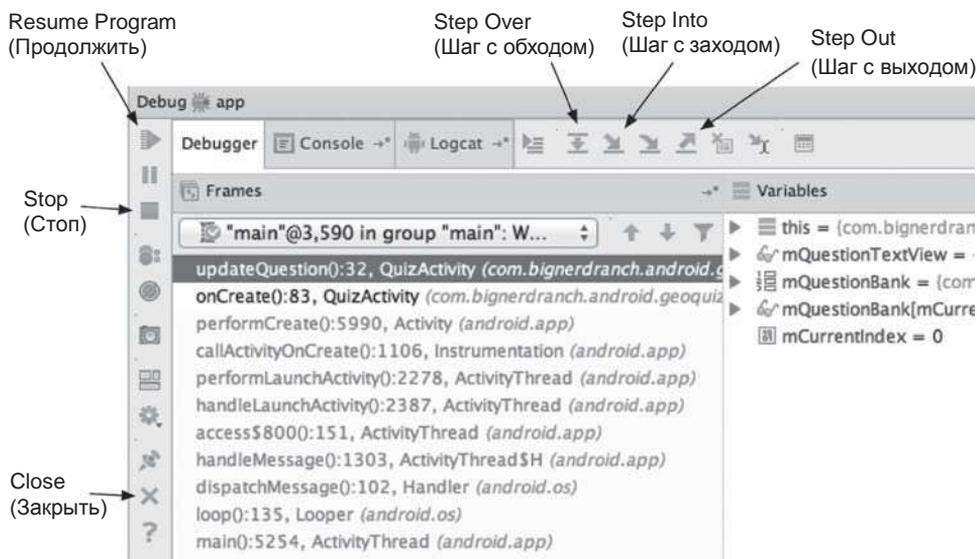


Рис. 4.6. Панель отладки

Теперь, когда мы добрались до интересного момента выполнения программы, можно немного осмотреться. Представление Variables используется для просмотра текущих значений объектов вашей программы. На ней должны отображаться

переменные, созданные в QuizActivity, а также еще одно значение: this (сам объект QuizActivity).

Переменную this можно было бы развернуть, чтобы увидеть все переменные, объявленные в суперклассе класса QuizActivity — Activity, в суперклассе класса Activity, в его суперклассе и т. д. Тем не менее пока мы ограничимся теми переменными, которые создали в программе.

Сейчас нас интересует только одно значение: mCurrentIndex. Прокрутите список и найдите в нем mCurrentIndex. Разумеется, переменная равна 0.

Код выглядит вполне нормально. Чтобы продолжить расследование, необходимо выйти из метода. Щелкните на кнопке шага с выходом Step Out.

Взгляните на панель редактора — управление передано слушателю OnClickListener кнопки mNextButton, в точку непосредственно после вызова updateQuestion(). Удобно, что и говорить.

Ошибку нужно исправить, но прежде чем вносить какие-либо изменения в код, необходимо прервать отладку приложения. Это можно сделать двумя способами: либо остановив программу, либо простым отключением отладчика. Чтобы остановить программу, щелкните на кнопке Stop на рис. 4.6. Вариант с отключением обычно проще: щелкните на кнопке Close, также обозначенной на рис. 4.6.

Верните OnClickListener в прежнее состояние.

Листинг 4.5. Возвращение к исходному состоянию (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            //mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Мы рассмотрели два способа поиска проблемной строки кода: сохранение в журнале трассировки стека и установка точки прерывания в отладчике. Какой способ лучше? Каждый находит свои применения, и скорее всего, один из них станет для вас основным.

Преимущество трассировки стека заключается в том, что трассировки из нескольких источников просматриваются в одном журнале. С другой стороны, чтобы получить новую информацию о программе, вы должны добавить новые команды

регистрации, заново построить приложение, развернуть его и добраться до нужной точки. С отладчиком работать проще. Запустив приложение с подключенным отладчиком (команда Debug As ► Android Application), вы сможете установить точку прерывания во время работы приложения, а потом поэкспериментировать для получения информации сразу о разных проблемах.

Прерывания по исключениям

Если вам недостаточно этих решений, вы также можете использовать отладчик для перехвата исключений. Вернитесь к методу onCreate() класса QuizActivity и снова закомментируйте строку кода, которая вызывала сбой приложения.

Листинг 4.6. Возвращение к нерабочей версии GeoQuiz (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ...

    mNextButton = (Button) findViewById(R.id.next_button);
    // mNextButton = (Button) findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            updateQuestion();
        }
    });
    ...
}
```

Выполните команду Run ► View Breakpoints, чтобы вызвать диалоговое окно прерываний по исключениям (рис. 4.7).

В диалоговом окне перечислены все текущие точки прерывания. Удалите точку прерывания, добавленную ранее, — выделите ее и щелкните на кнопке со знаком «-».

Диалоговое окно также позволяет установить точку прерывания, которая срабатывает при инициировании исключения, где бы оно ни произошло. Прерывания можно ограничить только неперехваченными исключениями или же применить их как к перехваченным, так и к неперехваченным исключениям.

Щелкните на кнопке со знаком «+», чтобы добавить новую точку прерывания. Выберите в раскрывающемся списке строку Java Exception Breakpoints. Теперь можно выбрать тип перехватываемых исключений. Введите *RuntimeException* и выберите в списке предложенных вариантов RuntimeException (java.lang).

RuntimeException является суперклассом NullPointerException, ClassCastException и других проблем времени выполнения, с которыми вы можете столкнуться, поэтому этот вариант удобен своей универсальностью.

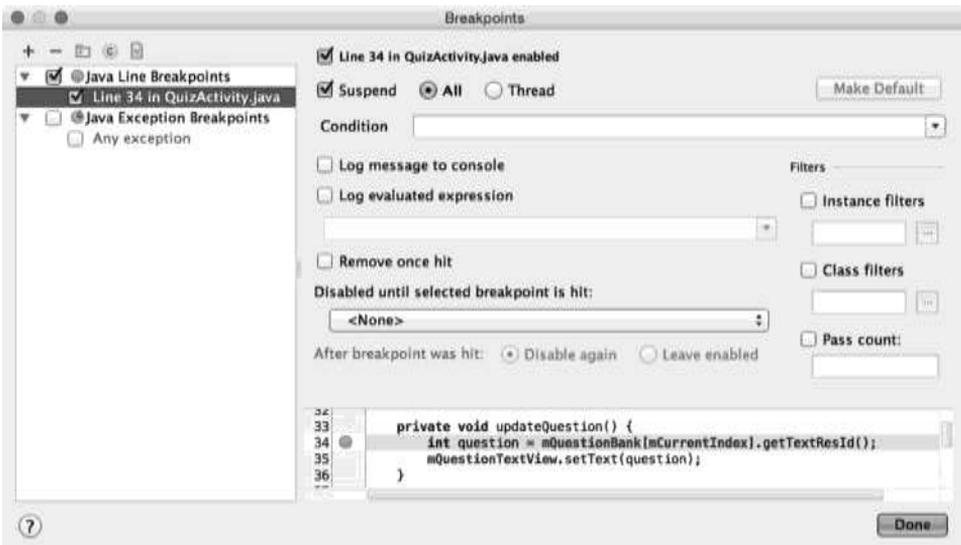


Рис. 4.7. Установка точки прерывания по исключению

Щелкните на кнопке Done и запустите GeoQuiz с присоединенным отладчиком. На этот раз отладчик переходит прямо к строке, в которой было инициировано исключение, — замечательно.

Учтите, что если эта точка прерывания останется включенной во время отладки, она может сработать на инфраструктурном коде или в других местах, на которые вы не рассчитывали. Не забудьте отключить ее, если она не используется. Для удаления точки прерывания снова вернитесь к диалоговому окну Run ▸ View Breakpoints....

Отмените изменения из листинга 4.6 и верните приложение GeoQuiz в работоспособное состояние.

Особенности отладки Android

Как правило, процесс отладки в Android не отличается от обычной отладки кода Java. Тем не менее у вас могут возникнуть проблемы в областях, специфических для Android (например, ресурсы), о которых компилятор Java ничего не знает.

Android Lint

На помощь приходит Android Lint — *статический анализатор* кода Android. Статические анализаторы проверяют код на наличие дефектов, не выполняя его. Android Lint использует свое знание инфраструктуры Android для проверки кода и выявления проблем, которые компилятор обнаружить не может. Как правило, к рекомендациям Android Lint стоит прислушиваться.

В главе 6 вы увидите, как Android Lint выдает предупреждение о проблеме совместимости. Кроме того, Android Lint может выполнять проверку типов для объектов, определенных в XML. Попробуйте включить в QuizActivity следующую ошибку преобразования типов.

Листинг 4.7. Ошибка в указании типа (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.true_button);
    mTrueButton = (Button)findViewById(R.id.question_text_view);
    ...
}
```

Из-за указания неверного идентификатора ресурса код попытается преобразовать TextView в Button во время выполнения, что приведет к исключению неправильного преобразования типа. Компилятор Java не видит проблем в этом коде, но Android Lint обнаружит ошибку еще до запуска приложения.

Вы можете запустить Lint вручную, чтобы получить список всех потенциальных проблем в приложении (в том числе и не столь серьезных, как эта). Выполните команду меню Analyze ▶ Inspect Code... Вам будет предложено выбрать анализируемые части проекта. Выберите вариант Whole project. Android Studio запускает Lint, а также несколько других статических анализаторов кода.

После завершения анализа выводится список потенциальных проблем, разбитый на несколько категорий. Раскройте категорию Android Lint, чтобы просмотреть информацию Lint о вашем проекте (рис. 4.8).

Выберите проблему в списке, чтобы получить более подробную информацию и узнать ее местонахождение в проекте. Ошибка несоответствия типов уже известна: вы только что сами ее создали. Исправьте преобразование типа в onCreate(Bundle).

Листинг 4.8. Исправление простой ошибки (QuizActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(TAG, "onCreate() called");
    setContentView(R.layout.activity_quiz);

    mQuestionTextView = (TextView)findViewById(R.id.question_text_view);

    mTrueButton = (Button)findViewById(R.id.question_text_view);
    mTrueButton = (Button)findViewById(R.id.true_button);
    ...
}
```

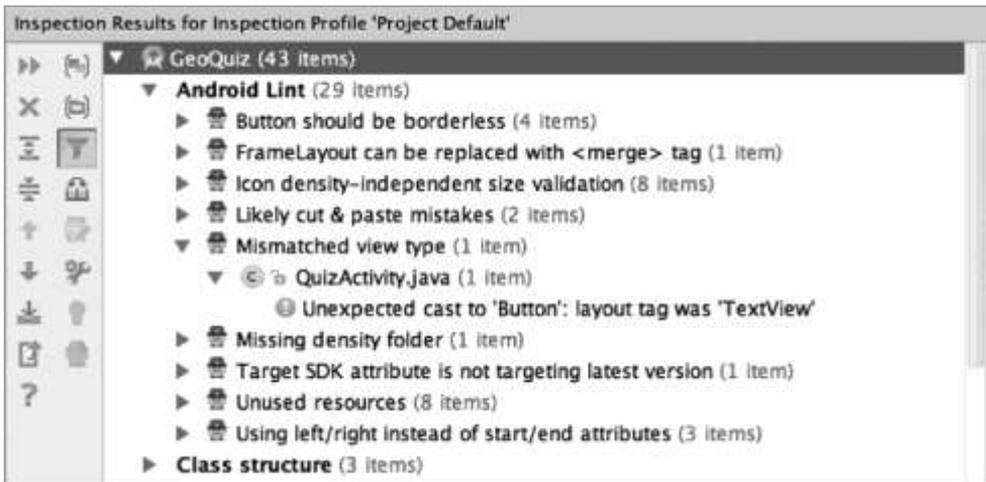


Рис. 4.8. Предупреждения Lint

Запустите приложение GeoQuiz и убедитесь в том, что оно снова работает нормально.

Проблемы с классом R

Всем известны ошибки построения, которые возникают из-за ссылок на ресурсы до их добавления (или удаления ресурсов, на которые ссылаются другие файлы). Обычно повторное сохранение файла после добавления ресурса или удаления ссылки приводит к тому, что Android Studio проводит построение заново, а проблемы исчезают.

Однако иногда такие ошибки остаются или появляются ниоткуда. Если вы столкнетесь с подобной ситуацией, попробуйте принять следующие меры.

Проверьте разметку XML в файлах ресурсов

Если файл R.java не был сгенерирован при последнем построении, то все ссылки на ресурс будут сопровождаться ошибками. Часто ошибки вызваны опечатками в разметке одного из файлов XML. Разметка макета не проверяется, поэтому среда не сможет привлечь ваше внимание к опечаткам в таких файлах. Если вы найдете ошибку и заново сохраните файл, код R.java будет сгенерирован заново.

Выполните чистку проекта

Выполните команду Build ▶ Clean Project. Android Studio строит проект заново, а результат построения нередко избавляется от ошибок.

Синхронизируйте проект с Gradle

Если вы вносите изменения в файл build.gradle, эти изменения необходимо синхронизировать с настройками построения вашего проекта. Выполните команду Tools ▶ Android ▶ Sync Project with Gradle Files. Android Studio строит проект заново

с правильными настройками; при этом проблемы, связанные с изменением конфигурации Gradle, могут исчезнуть.

Запустите Android Lint

Обратите внимание на предупреждения, полученные от Android Lint. При запуске этой программы нередко выявляются совершенно неожиданные проблемы.

Если у вас все еще остаются проблемы с ресурсами (или иные проблемы), отдохните и просмотрите сообщения об ошибках и файлы макета на свежую голову. В запале легко пропустить ошибку. Также проверьте все ошибки и предупреждения Android Lint. При спокойном повторном рассмотрении сообщений нередко выявляются ошибки или опечатки.

Наконец, если вы зашли в тупик или у вас возникли другие проблемы с Android Studio, обратитесь к архивам <http://stackoverflow.com> или посетите форум книги по адресу <http://forums.bignerdranch.com>.

1.5. Создание второй активности

В этой главе нам предстоит много сделать. К счастью, часть рутинной работы будет выполнена за вас мастером New Activity среды Android Studio.

Но сначала откройте файл `strings.xml` и добавьте строки, необходимые для этой главы.

Листинг 5.1. Добавление строк (`strings.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  ...
  <string name="question_asia">Lake Baikal is the world's oldest and deepest
    freshwater lake.</string>
  <string name="warning_text">Are you sure you want to do this?</string>
  <string name="show_answer_button">Show Answer</string>
  <string name="cheat_button">Cheat!</string>
  <string name="judgment_toast">Cheating is wrong.</string>
</resources>
```

Создание нового макета

При создании активности обычно изменяются по крайней мере три файла: файл класса Java, макет XML и манифест приложения. Но если эти файлы будут изменены некорректно, Android будет протестовать. Чтобы избежать возможных проблем, воспользуйтесь мастером New Activity среды Android Studio.

Запустите мастер New Activity: щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.geoquiz` в окне инструментов Project и выберите команду New ▶ Activity ▶ Blank Activity (рис. 5.3).

На экране появляется диалоговое окно, изображенное на рис. 5.4. Введите в поле Activity Name строку `CheatActivity` — это имя вашего subclasses Activity. В поле Layout Name автоматически генерируется имя `activity_cheat`. Оно становится базовым именем файла макета, создаваемого мастером. Поле Title также автоматически заполняется текстом `CheatActivity`, но поскольку этот текст будет виден пользователю, оставьте в поле только «Cheat».

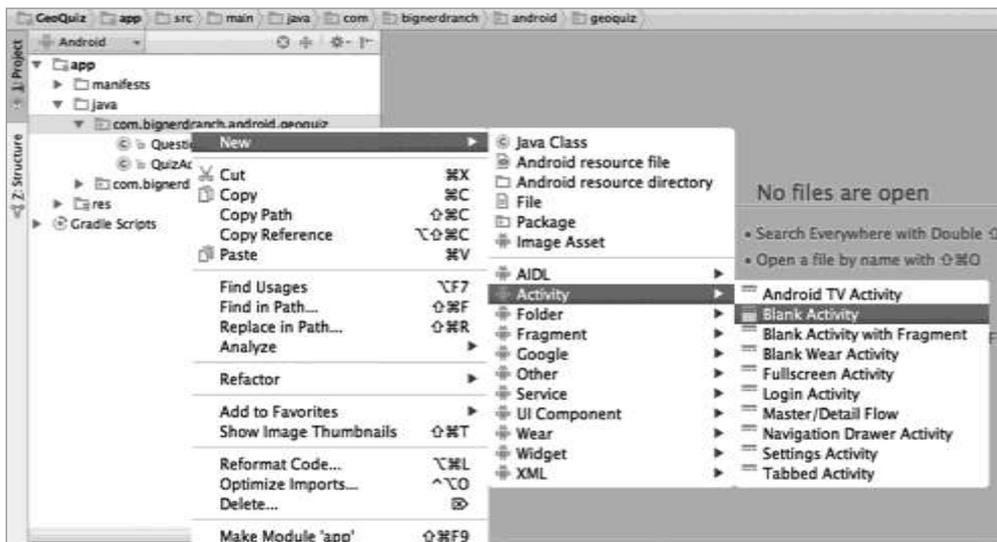


Рис. 5.3. Меню запуска мастера New Activity

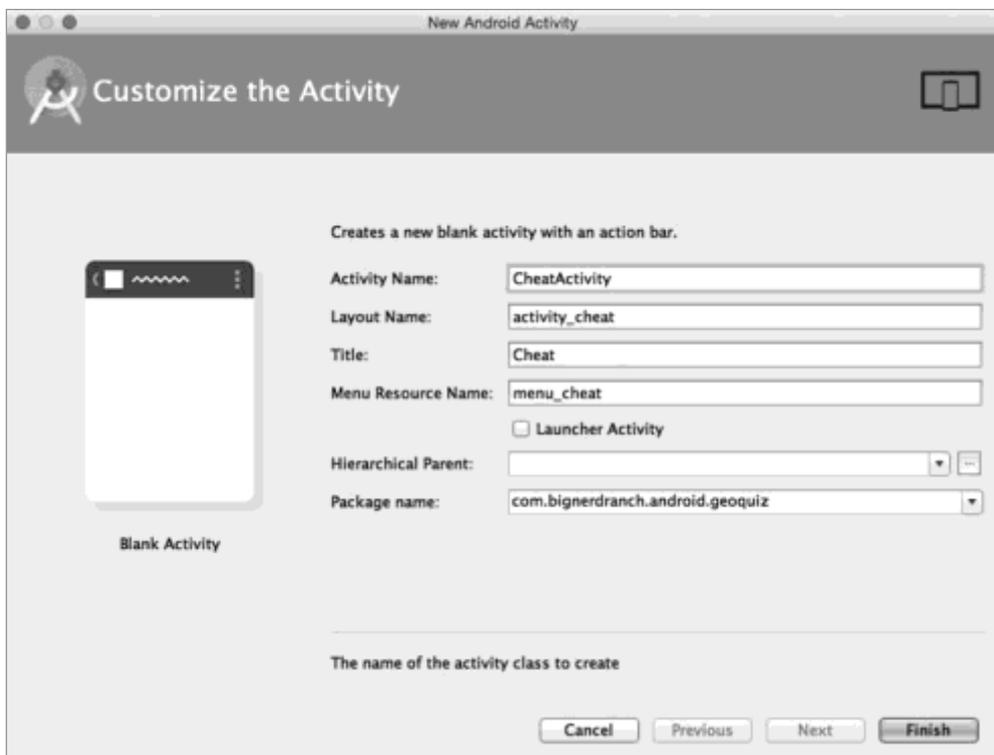


Рис. 5.4. Мастер New Blank Activity

Остальным полям можно оставить значения по умолчанию, но не забудьте убедиться в том, что имя пакета выглядит так, как предполагается. Оно определяет местонахождение CheatActivity.java в файловой системе. Щелкните на кнопке Finish, чтобы мастер завершил свою работу.

Теперь можно обратиться к пользовательскому интерфейсу. Снимок экрана в начале главы показывает, как должна выглядеть активность CheatActivity. На рис. 5.5 приведены определения виджетов.

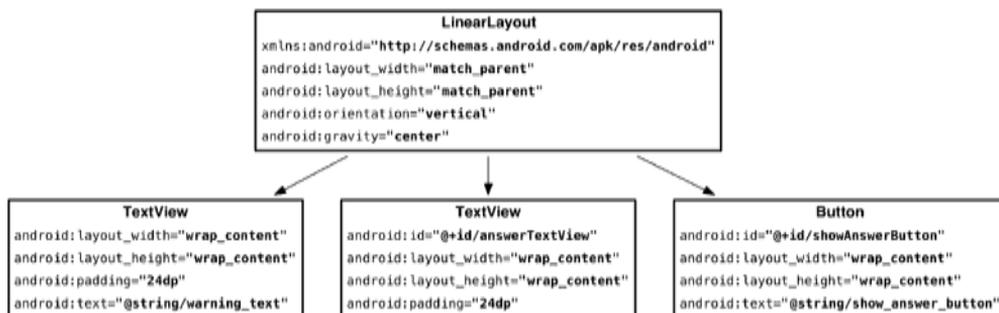


Рис. 5.5. Схема макета CheatActivity

После завершения мастера должен открыться файл activity_cheat.xml из каталога layout. Если этого не произошло, откройте файл самостоятельно и перейдите в режим представления Text (XML).

Попробуйте создать разметку XML для макета по образцу рис. 5.5. Замените простой макет элементом LinearLayout, и так далее по дереву представлений. После главы 8 в тексте вместо длинных фрагментов XML будут приводиться только схемы макетов вроде рис. 5.5, так что вам стоит освоить самостоятельное создание XML-макетов. Сравните свою работу с листингом 5.2.

Листинг 5.2. Заполнение макета второй активности (activity_cheat.xml)

```

<LinearLayout 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"
    android:gravity="center"
    android:orientation="vertical"
    tools:context="com.bignerdranch.android.geoquiz.CheatActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp"
        android:text="@string/warning_text"/>
  
```

```
<TextView
```

```
    android:id="@+id/answer_text_view"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="24dp"  
    tools:text="Answer"/>
```

```
<Button
```

```
    android:id="@+id/show_answer_button"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/show_answer_button"/>
```

```
</LinearLayout>
```

Обратите внимание на специальное пространство имен XML для `tools` и атрибута `tools:text` виджета `TextView`, в котором будет выводиться ответ. Это пространство имен позволяет переопределить любой атрибут виджета для того, чтобы он иначе отображался в режиме предварительного просмотра Android Studio. Так как у `TextView` есть атрибут `text`, вы можете присвоить ему фиктивное литеральное значение, чтобы знать, как виджет будет выглядеть во время выполнения. Значение «Answer» никогда не появится в реальном приложении. Удобно, верно?

Мы не будем создавать для `activity_cheat.xml` альтернативный макет с альбомной ориентацией, однако существует возможность увидеть, как макет по умолчанию будет отображаться в альбомном режиме.

В окне предварительного просмотра найдите на панели инструментов над панелью предварительного просмотра кнопку, на которой изображено устройство с изогнутой стрелкой. Щелкните на этой кнопке, чтобы изменить ориентацию макета (рис. 5.6).

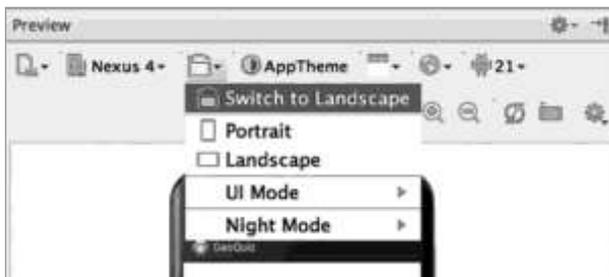


Рис. 5.6. Просмотр макета `activity_cheat.xml` в альбомной ориентации

Макет по умолчанию достаточно хорошо смотрится в обеих ориентациях, можно переходить к созданию subclasses активности.

Создание нового subclasses активности

В окне инструментов Project найдите пакет `com.bignerdranch.android.geoquiz` и откройте класс `CheatActivity` (он находится в файле `CheatActivity.java`).

Класс уже содержит простейшую реализацию `onCreate(...)`, которая передает идентификатор ресурса макета, определенного в `activity_cheat.xml`, при вызове `setContentView(...)`.

Со временем функциональность метода `onCreate(...)` в `CheatActivity` будет расширена. А пока давайте перейдем к следующему шагу: объявлению `CheatActivity` в манифесте приложения.

Объявление активностей в манифесте

Манифест (manifest) представляет собой файл XML с метаданными, описывающими ваше приложение для ОС Android. Файл манифеста всегда называется `AndroidManifest.xml` и располагается в каталоге `app/manifests` вашего проекта.

В окне инструментов Project найдите и откройте `AndroidManifest.xml`. Также можно воспользоваться диалоговым окном `Android Studio Quick Open` — нажмите `Command+Shift+O` (`Ctrl+Shift+N`) и начинайте вводить имя файла. Когда в окне будет предложен правильный файл, нажмите `Return` (`Enter`), чтобы открыть этот файл.

Каждая активность приложения должна быть объявлена в манифесте, чтобы она стала доступной для ОС.

Когда вы использовали мастер новых приложений для создания `QuizActivity`, мастер объявил активность за вас. Аналогичным образом мастер `New Activity` объявил `CheatActivity`, добавив разметку XML, выделенную в листинге 5.3.

Листинг 5.3. Объявление `CheatActivity` в манифесте (`AndroidManifest.xml`)

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

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

```
        android:label="@string/title_activity_cheat" >
    </activity>
</application>

</manifest>
```

Атрибут `android:name` является обязательным. Точка в начале значения атрибута сообщает ОС, что класс этой активности находится в пакете, который задается атрибутом `package` в элементе `manifest` в начале файла.

Иногда встречается полностью уточненный атрибут `android:name` вида `android:name="com.bignerdranch.android.geoquiz.CheatActivity"`. Длинная форма записи идентична версии в листинге 5.3.

Манифест содержит много интересной информации, но сейчас мы хотим как можно быстрее организовать работу `CheatActivity`. Другие части манифеста будут рассматриваться позднее.

Добавление кнопки Cheat в QuizActivity

Итак, пользователь должен нажать кнопку в `QuizActivity`, чтобы вызвать на экран экземпляр `CheatActivity`. Следовательно, мы должны включить новые кнопки в `layout/activity_quiz.xml` и `layout-land/activity_quiz.xml`.

В макете по умолчанию добавьте новую кнопку как прямого потомка корневого элемента `LinearLayout`. Ее определение должно непосредственно предшествовать кнопке `Next`.

Листинг 5.4. Добавление кнопки Cheat! в макет по умолчанию (`layout/activity_quiz.xml`)

```
...
</LinearLayout>

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

<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp"/>

</LinearLayout>
```

В альбомном макете новая кнопка размещается внизу и по центру корневого элемента `FrameLayout`.

Листинг 5.5. Добавление кнопки Cheat! в альбомный макет (layout-land/activity_quiz.xml)

```

...
</LinearLayout>

<Button
    android:id="@+id/cheat_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|center"
    android:text="@string/cheat_button" />

<Button
    android:id="@+id/next_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|right"
    android:text="@string/next_button"
    android:drawableRight="@drawable/arrow_right"
    android:drawablePadding="4dp" />

</FrameLayout>

```

Сохраните файлы макетов и откройте QuizActivity.java. Добавьте переменную, получите ссылку и назначьте заглушку View.OnClickListener для кнопки Cheat!.

Листинг 5.6. Подключение кнопки Cheat! (QuizActivity.java)

```

public class QuizActivity extends AppCompatActivity {
    ...
    private Button mNextButton;
    private Button mCheatButton;
    ...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...

        mCheatButton = (Button)findViewById(R.id.cheat_button);
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Запуск CheatActivity
            }
        });

        if (savedInstanceState != null) {
            mCurrentIndex = savedInstanceState.getInt(KEY_INDEX, 0);
        }

        updateQuestion();
    }
    ...
}

```

Теперь можно переходить к запуску CheatActivity.

Запуск активности

Чтобы запустить одну активность из другой, проще всего воспользоваться методом `Activity`:

```
public void startActivity(Intent intent)
```

Напрашивается предположение, что `startActivity(...)` является статическим методом, который должен вызываться для запускаемого subclasses `Activity`. Тем не менее это не так. Когда активность вызывает `startActivity(...)`, этот вызов передается ОС.

А точнее, он передается компоненту ОС, который называется `ActivityManager`. `ActivityManager` создает экземпляры `Activity` и вызывает его метод `onCreate(...)` (рис. 5.7).

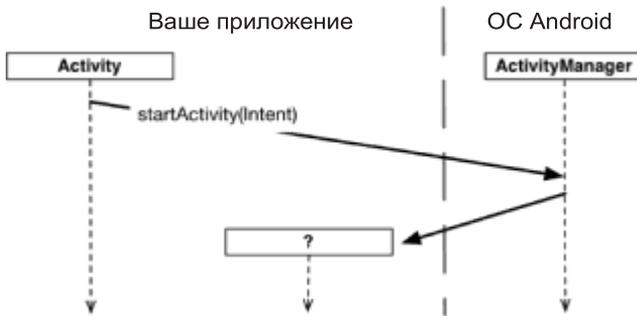


Рис. 5.7. Запуск активности

Откуда `ActivityManager` узнает, какую активность следует запустить? Эта информация передается в параметре `Intent`.

Передача информации через интенды

Интенд (`intent`) — объект, который может использоваться *компонентом* для взаимодействия с ОС. Пока что из компонентов нам встречались только активности, но еще существуют службы (`services`), широкопередаточные приемники (`broadcast receivers`) и поставщики контента (`content providers`).

Интенды представляют собой многоцелевые средства передачи информации, а класс `Intent` предоставляет разные конструкторы в зависимости от того, для чего должен использоваться интенд.

В данном случае интенд сообщает `ActivityManager`, какую активность следует запустить, поэтому мы используем следующий конструктор:

```
public Intent(Context packageContext, Class<?> cls)
```

Объект Class задает класс активности, которая должна быть запущена ActivityManager. Аргумент Context сообщает ActivityManager, в каком пакете находится объект Class (рис. 5.8).

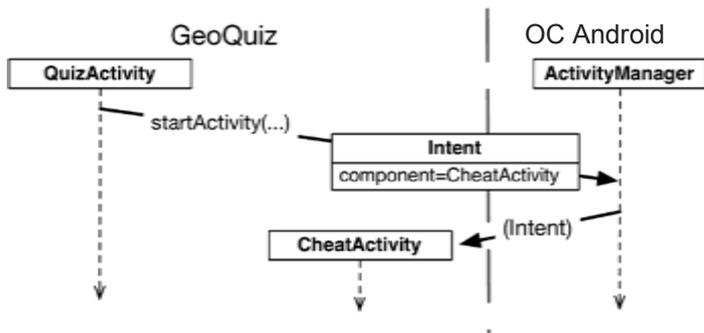


Рис. 5.8. Интент: передача ActivityManager информации о том, что нужно сделать

В слушателе mCheatButton создайте объект Intent, включающий класс CheatActivity, а затем передайте его при вызове startActivity(Intent) (листинг 5.7).

Листинг 5.7. Запуск CheatActivity (QuizActivity.java)

```
...
mCheatButton = (Button)findViewById(R.id.cheat_button);
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Запуск CheatActivity
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        startActivity(i);
    }
});
...
```

Прежде чем запускать активность, ActivityManager ищет в манифесте пакета объявление с именем, соответствующим заданному объекту Class. Если такое объявление будет найдено, активность запускается, — все хорошо. Если объявление не найдено, выдается неприятное исключение ActivityNotFoundException, которое может привести к аварийному завершению приложения. Вот почему все активности должны объявляться в манифесте.

Запустите приложение GeoQuiz. Нажмите кнопку Cheat!; на экране появляется новая активность. Теперь нажмите кнопку Back. Активность CheatActivity уничтожается, а приложение возвращается к QuizActivity.

Интенты явные и неявные

При создании объекта Intent с объектом Context и Class вы создаете *явный* (explicit) интент. Явные интенты используются для запуска активностей в приложениях.

Может показаться странным, что две активности внутри приложения должны взаимодействовать через компонент ActivityManager, находящийся вне приложения. Тем не менее такая схема позволяет активности одного приложения легко работать с активностью другого приложения.

Когда активность в вашем приложении должна запустить активность в другом приложении, вы создаете *неявный* (implicit) интент. Неявные интенты будут использоваться в главе 15.

Передача данных между активностями

Итак, в нашем приложении действуют активности QuizActivity и CheatActivity, и мы можем подумать о передаче данных между ними. На рис. 5.9 показано, какие данные будут передаваться между двумя активностями.

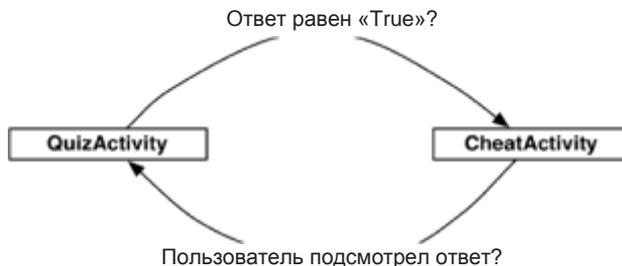


Рис. 5.9. Обмен данными между QuizActivity и CheatActivity

QuizActivity передает CheatActivity ответ на текущий вопрос при запуске CheatActivity.

Когда пользователь нажимает кнопку Back, чтобы вернуться к QuizActivity, экземпляр CheatActivity уничтожается. В последний момент он передает QuizActivity информацию о том, посмотрел ли пользователь правильный ответ.

Начнем с передачи данных от QuizActivity к CheatActivity.

Дополнения интентов

Чтобы сообщить CheatActivity ответ на текущий вопрос, мы будем передавать значение

```
mQuestionBank[mCurrentIndex].isAnswerTrue();
```

Значение будет послано в виде *дополнения* (extra) объекта Intent, передаваемого startActivity(Intent).

Дополнения представляют собой произвольные данные, которые вызывающая активность может передать вместе с интендом. Их можно рассматривать как аналоги аргументов конструктора, несмотря на то что вы не можете использовать произвольный конструктор с субклассом активности (Android создает экземпляры активностей и несет ответственность за их жизненный цикл). ОС направляет интенд активностям-получателям, которая обращается к дополнению и извлекает данные.

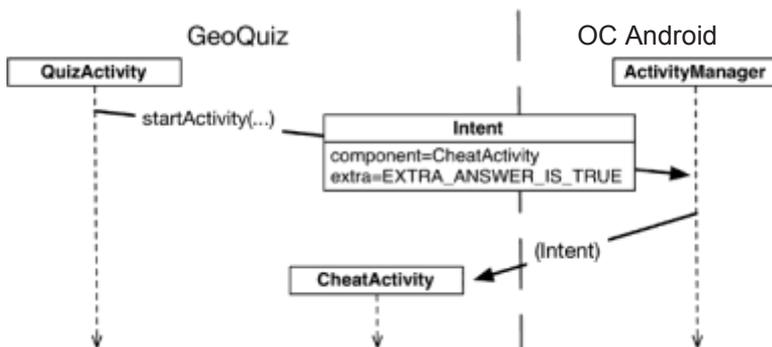


Рис. 5.10. Дополнения интендов: взаимодействие с другими активностями

Дополнение представляет собой пару «ключ-значение» наподобие той, которая использовалась для сохранения значения mCurrentIndex в QuizActivity. onSaveInstanceState(Bundle).

Для включения дополнений в интенд используется метод Intent.putExtra(...), а точнее, метод

```
public Intent putExtra(String name, boolean value)
```

Метод Intent.putExtra(...) существует в нескольких разновидностях, но он всегда получает два аргумента. В первом аргументе всегда передается ключ String, а во втором — значение того или иного типа. Метод всегда возвращает сам объект Intent, так что при необходимости можно использовать цепочки из сцепленных вызовов.

Добавьте в CheatActivity.java ключ для дополнения.

Листинг 5.8. Добавление константы (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    ...
}
```

Активность может запускаться из нескольких разных мест, поэтому ключи для дополнений должны определяться в активностях, которые читают и используют их.

Как видно из листинга 5.8, уточнение дополнения именем пакета предотвращает конфликты имен с дополнениями других приложений.

Теперь можно вернуться к QuizActivity и включить дополнение в интент, но это не лучший вариант. Нет никаких причин, по которым QuizActivity или любому другому коду приложения было бы нужно знать подробности реализации тех данных, которые CheatActivity ожидает получить в дополнении интента. Эту работу правильнее инкапсулировать в методе newIntent(...).

Создайте этот метод в CheatActivity:

Листинг 5.9. Метод newIntent(...) (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";

    public static Intent newIntent(Context packageContext, boolean
                                answersIsTrue) {
        Intent i = new Intent(packageContext, CheatActivity.class);
        i.putExtra(EXTRA_ANSWER_IS_TRUE, answersIsTrue);
        return i;
    }

    ...
}
```

Этот статический метод позволяет создать объект Intent, настроенный дополнениями, необходимыми для CheatActivity. Логический аргумент answersIsTrue помещается в интент под закрытым именем с использованием константы EXTRA_ANSWER_IS_TRUE. Вскоре мы прочитаем это значение. Подобное использование метода newIntent(...) для subclasses активностей упрощает настройку интентов в других местах кода.

Разужречь зашла о других местах, используем этот метод в слушателе кнопки Cheat!:

Листинг 5.10. Запуск CheatActivity.java с дополнением (QuizActivity.java)

```
...
mCheatButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // Запуск CheatActivity
        Intent i = new Intent(QuizActivity.this, CheatActivity.class);
        boolean answersIsTrue = mQuestionBank[mCurrentIndex].
            isAnswerTrue();
        Intent i = CheatActivity.newIntent(QuizActivity.this,
            answersIsTrue);
        startActivity(i);
    }
});
updateQuestion();
}
```

В нашей ситуации достаточно одного дополнения, но при необходимости можно добавить в Intent несколько дополнений. В этом случае добавьте в `newIntent(...)` дополнительные аргументы в соответствии с используемой схемой.

Для чтения значения из дополнения используется метод

```
public boolean getBooleanExtra(String name, boolean defaultValue)
```

Первый аргумент `getBooleanExtra(...)` содержит имя дополнения, а второй — ответ по умолчанию, если ключ не найден.

В `CheatActivity` прочитайте значение из дополнения в `onCreate(Bundle)` и сохраните его в переменной.

Листинг 5.11. Использование дополнения (`CheatActivity.java`)

```
public class CheatActivity extends AppCompatActivity {
    ---
    private static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    private boolean mAnswersIsTrue;
    ---
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_cheat);

        mAnswersIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE,
            false);
    }
    ---
}
```

Обратите внимание: `Activity.getIntent()` возвращает объект `Intent`, который был передан в `startActivity(Intent)`.

Наконец, добавьте в `CheatActivity` код, обеспечивающий использование прочитанного значения в виджете `TextView` ответа и кнопке `Show Answer`.

Листинг 5.12. Добавление функциональности подсматривания ответов (`CheatActivity.java`)

```
public class CheatActivity extends AppCompatActivity {
    ---
    private boolean mAnswersIsTrue;

    private TextView mAnswerTextView;
    private Button mShowAnswer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

setContentView(R.layout.activity_cheat);

mAnswerIsTrue = getIntent().getBooleanExtra(EXTRA_ANSWER_IS_TRUE, false);

mAnswerTextView = (TextView)findViewById(R.id.answer_text_view);

mShowAnswer = (Button)findViewById(R.id.show_answer_button);
mShowAnswer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
    }
});
}
}
}

```

Код достаточно тривиален: мы задаем текст `TextView` при помощи метода `TextView.setText(int)`. Метод `TextView.setText(...)` существует в нескольких вариантах; здесь используется вариант, получающий идентификатор строкового ресурса.

Запустите приложение `GeoQuiz`. Нажмите кнопку `Cheat!`, чтобы перейти к `CheatActivity`. Затем нажмите кнопку `Show Answer`, чтобы открыть ответ на текущий вопрос.

Получение результата от дочерней активности

В текущей версии приложения пользователь может беспрепятственно жульничать. Сейчас мы исправим этот недостаток; для этого `CheatActivity` будет сообщать `QuizActivity`, подсмотрел ли пользователь ответ.

Чтобы получить информацию от дочерней активности, вызовите следующий метод `Activity`:

```
public void startActivityForResult(Intent intent, int requestCode)
```

Первый параметр содержит тот же интент, что и прежде. Во втором параметре передается *код запроса* — определяемое пользователем целое число, которое передается дочерней активности, а затем принимается обратно родителем. Оно используется тогда, когда активность запускает сразу несколько типов дочерних активностей и ей необходимо определить, кто из потомков возвращает данные. `QuizActivity` запускает только один тип дочерней активности, но передача константы в коде запроса — полезная практика, с которой вы будете готовы к возможным будущим изменениям.

В классе `QuizActivity` измените слушателя `mCheatButton` и включите в него вызов `startActivityForResult(Intent, int)`.

Листинг 5.13. Вызов `startActivityForResult(...)` (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    private static final String TAG = "QuizActivity";
    private static final String KEY_INDEX = "index";
    private static final int REQUEST_CODE_CHEAT = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mCheatButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                boolean answersTrue = mQuestionBank[mCurrentIndex].
                    isAnswerTrue();
                Intent i = CheatActivity.newIntent(QuizActivity.this,
                    answersTrue);
                startActivity(i);
                startActivityForResult(i, REQUEST_CODE_CHEAT);
            }
        });
        ...
    }
}
```

Передача результата

Существует два метода, которые могут вызываться в дочерней активности для возвращения данных родителю:

```
public final void setResult(int resultCode)
public final void setResult(int resultCode, Intent data)
```

Как правило, `resultCode` содержит одну из двух predefined констант: `Activity.RESULT_OK` или `Activity.RESULT_CANCELED`. (Также можно использовать другую константу `RESULT_FIRST_USER` как смещение при определении собственных кодов результатов.)

Назначение кода результата полезно в том случае, когда родитель должен выполнить разные действия в зависимости от того, как завершилась дочерняя активность.

Например, если в дочерней активности присутствуют кнопки `OK` и `Cancel`, то дочерняя активность может назначать разные коды результата в зависимости от того, какая кнопка была нажата. Родительская активность будет выбирать разные варианты действий в зависимости от полученного кода.

Вызов `setResult(...)` не является обязательным для дочерней активности. Если вам не нужно различать результаты или получать произвольные данные по интенту, просто разрешите ОС отправить код результата по умолчанию. Код результата всегда возвращается родителю, если дочерняя активность была запущена методом `startActivityForResult(...)`. Если метод `setResult(...)` не вызывался, то при нажатии пользователем кнопки `Back` родитель получит код `Activity.RESULT_CANCELED`.

Возвращение интента

В нашей реализации дочерняя активность должна вернуть QuizActivity некоторые данные. Соответственно мы создадим объект Intent, поместим в него дополнение, а затем вызовем Activity.setResult(int, Intent) для передачи этих данных QuizActivity.

Добавьте в CheatActivity константу для ключа дополнения и закрытый метод, который выполняет необходимую работу. Затем включите вызов этого метода в слушателя кнопки Show Answer.

Листинг 5.14. Назначение результата (CheatActivity.java)

```
public class CheatActivity extends AppCompatActivity {

    public static final String EXTRA_ANSWER_IS_TRUE =
        "com.bignerdranch.android.geoquiz.answer_is_true";
    public static final String EXTRA_ANSWER_SHOWN =
        "com.bignerdranch.android.geoquiz.answer_shown";
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
        mShowAnswer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (mAnswerIsTrue) {
                    mAnswerTextView.setText(R.string.true_button);
                } else {
                    mAnswerTextView.setText(R.string.false_button);
                }
                setAnswerShownResult(true);
            }
        });
    }

    private void setAnswerShownResult(boolean isAnswerShown) {
        Intent data = new Intent();
        data.putExtra(EXTRA_ANSWER_SHOWN, isAnswerShown);
        setResult(RESULT_OK, data);
    }

}
```

Когда пользователь нажимает кнопку Show Answer, CheatActivity упаковывает код результата и интент в вызове setResult(int, Intent).

Затем, когда пользователь нажимает кнопку Back для возвращения к QuizActivity, ActivityManager вызывает следующий метод родительской активности:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data)
```

В параметрах передается исходный код запроса от QuizActivity, код результата и интент, переданный setResult(...).

Последовательность взаимодействий изображена на рис. 5.11.

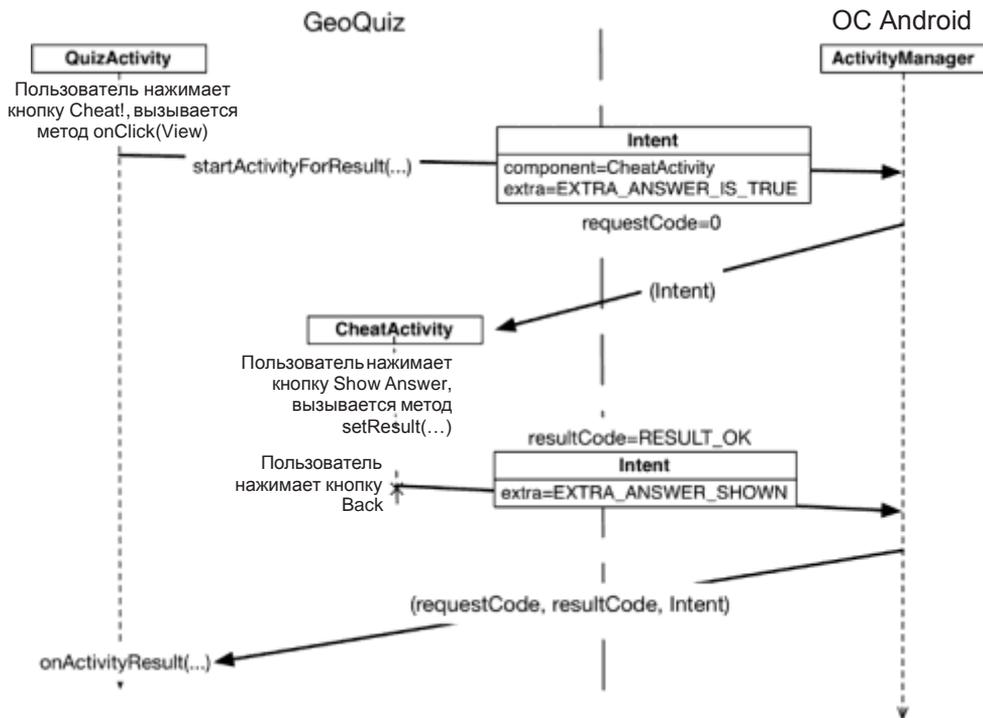


Рис. 5.11. Диаграмма последовательности для GeoQuiz

Остается последний шаг: переопределение onActivityResult(int, int, Intent) в QuizActivity для обработки результата. Но поскольку содержимое интента результата также относится к подробностям реализации CheatActivity, добавьте еще один метод для упрощения декодирования дополнения к виду, который может использоваться QuizActivity.

Листинг 5.15. Декодирование интента результата (CheatActivity.java)

```

public static Intent newIntent(Context packageContext, boolean answersTrue) {
    Intent i = new Intent(packageContext, CheatActivity.class);
    i.putExtra(EXTRA_ANSWER_IS_TRUE, answersTrue);
    return i;
}

public static boolean wasAnswerShown(Intent result) {
    return result.getBooleanExtra(EXTRA_ANSWER_SHOWN, false);
}

```

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
}
```

Обработка результата

Добавьте в QuizActivity.java новую переменную для хранения значения, возвращаемого CheatActivity. Затем включите в переопределение onActivityResult(...) код его получения, проверки кода запроса и кода результата, чтобы быть уверенным в том, что они соответствуют ожиданиям. Как и в предыдущем случае, это полезная практика, которая упростит возможные будущие изменения.

Листинг 5.16. Реализация onActivityResult(...) (QuizActivity.java)

```
public class QuizActivity extends AppCompatActivity {
    ...
    private int mCurrentIndex = 0;
    private boolean mIsCheater;
    ...

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        ...
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent
        data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }

        if (requestCode == REQUEST_CODE_CHEAT) {
            if (data == null) {
                return;
            }
            mIsCheater = CheatActivity.wasAnswerShown(data);
        }
    }
    ...
}
```

Измените метод checkAnswer(boolean) в QuizActivity. Он должен проверять, посмотрел ли пользователь ответ, и реагировать соответствующим образом.

Листинг 5.17. Изменение уведомления в зависимости от значения mIsCheater (QuizActivity.java)

```
private void checkAnswer(boolean userPressedTrue) {
    boolean answerIsTrue = mQuestionBank[mCurrentIndex].isAnswerTrue();
```

```

int messageResId = 0;

if (mIsCheater) {
    messageResId = R.string.judgment_toast;
} else {
    if (userPressedTrue == answerIsTrue) {
        messageResId = R.string.correct_toast;
    } else {
        messageResId = R.string.incorrect_toast;
    }
}

Toast.makeText(this, messageResId, Toast.LENGTH_SHORT)
    .show();
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    mNextButton = (Button)findViewById(R.id.next_button);
    mNextButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mCurrentIndex = (mCurrentIndex + 1) % mQuestionBank.length;
            mIsCheater = false;
            updateQuestion();
        }
    });
    ...
}

```

Запустите приложение GeoQuiz. Попробуйте подсмотреть ответ и увидите, что произойдет.

Ваши активности с точки зрения Android

Давайте посмотрим, что происходит при переключении между активностями с точки зрения ОС. Прежде всего, когда вы щелкаете на приложении GeoQuiz в лаунчере, ОС запускает не приложение, а активность в приложении. А если говорить точнее, запускается *активность лаунчера* приложения. Для GeoQuiz активностью лаунчера является QuizActivity.

Когда мастер New Application создавал приложение GeoQuiz, класс QuizActivity был назначен активностью лаунчера по умолчанию. Статус активности лаунчера задается в манифесте элементом `intent-filter` в объявлении QuizActivity (листинг 5.18).

Листинг 5.18. QuizActivity объявляется активностью лаунчера (AndroidManifest.xml)

```

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

```

```
... >
...
<application
  ... >
  <activity
    android:name="com.bignerdranch.android.geoquiz.QuizActivity"
    android:label="@string/app_name" >
    <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
  <activity
    android:name=".CheatActivity"
    android:label="@string/app_name" />
</application>

</manifest>
```

Когда экземпляр QuizActivity окажется на экране, пользователь может нажать кнопку Cheat!. При этом экземпляр CheatActivity запускается поверх QuizActivity. Эти активности образуют стек (рис. 5.12).

При нажатии кнопки Back в CheatActivity этот экземпляр выводится из стека, а QuizActivity занимает свою позицию на вершине, как показано на рис. 5.12.

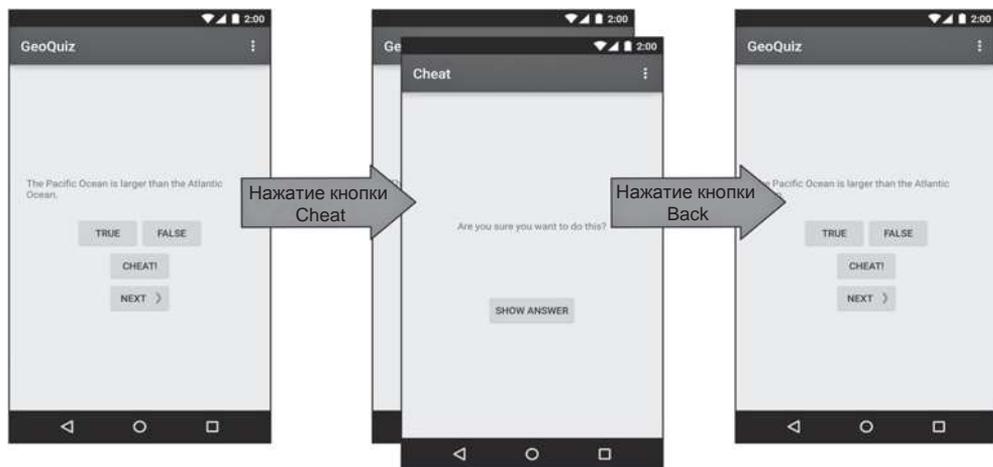


Рис. 5.12. Стек GeoQuiz

Вызов Activity.finish() в CheatActivity также выводит CheatActivity из стека.

Если запустить GeoQuiz и нажать кнопку Back в QuizActivity, то активность QuizActivity будет извлечена из стека и вы вернетесь к последнему экрану, который просматривался перед запуском GeoQuiz (рис. 5.13).

Если вы запустили GeoQuiz из лаунчера (launcher), то нажатие кнопки Back из QuizActivity вернет вас обратно (рис. 5.14).

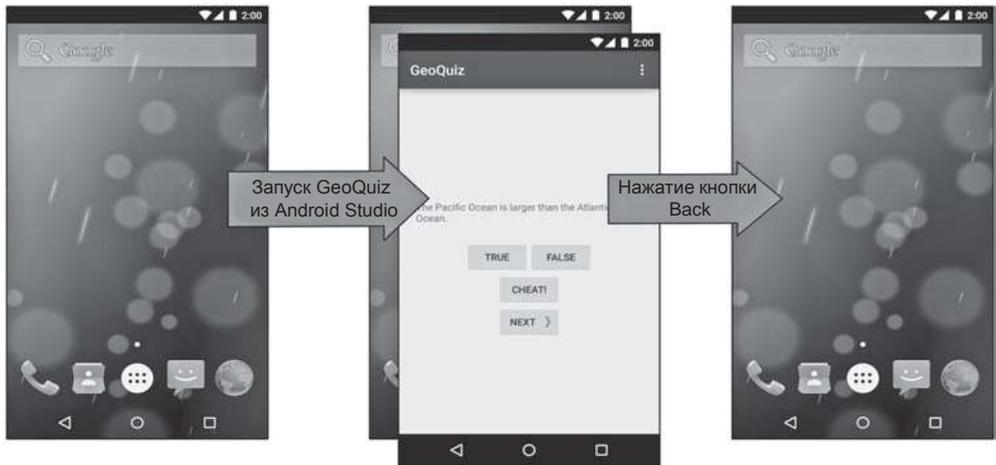


Рис. 5.13. Экран Home

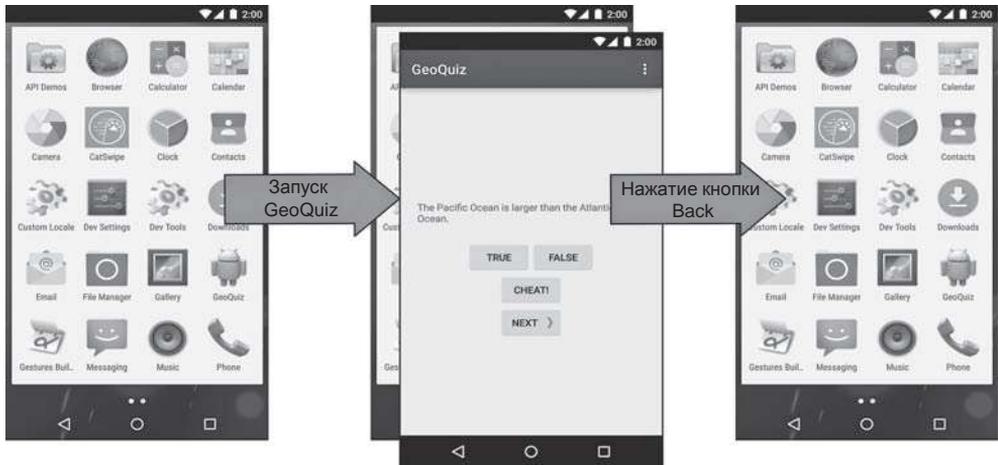


Рис. 5.14. Запуск GeoQuiz из лаунчера

Нажатие Back в запущенном лаунчере вернет вас к экрану, который был открыт перед его запуском.

Мы видим, что ActivityManager поддерживает стек возврата (back stack) и что этот стек не ограничивается активностями вашего приложения. Он совместно используется активностями всех приложений; это одна из причин, по которым ActivityManager участвует в запуске активностей и находится под управлением

ОС, а не вашего приложения. Стек представляет использование ОС и устройства в целом, а не использование одного приложения.

Упражнение

Мошенники никогда не выигрывают... Если, конечно, им не удастся обойти вашу защиту от мошенничества. А скорее всего, они так и сделают — именно потому, что они мошенники.

Приложение GeoQuiz содержит ряд «лазеек», которыми могут воспользоваться мошенники. В этом упражнении вы должны устранить эти лазейки. Основные дефекты приложения перечислены ниже по возрастанию сложности, от самых простых к самым сложным:

- ① Подсмотрев ответ, пользователь может повернуть CheatActivity, чтобы сбросить результат.
- ① После возвращения пользователь может повернуть QuizActivity, чтобы сбросить флаг mlsCheater.
- ① Пользователь может нажимать кнопку Next до тех пор, пока вопрос, ответ на который был подсмотрен, снова не появится на экране.

Удачи!

Модуль 2. Языковые средства реализации мобильных приложений

2.1. Версии Android SDK

В табл. 6.1 перечислены версии SDK, соответствующие версии прошивки Android, и процент устройств, использующих их, по состоянию на июнь 2015 года.

Таблица 6.1. Уровни API Android, версии прошивки и процент устройств

Уровень API	Кодовое название	Версия прошивки устройства	Процент использующих устройств
22	Lollipop	5.1	0,8
21		5.0	11,6
19	KitKat	4.4	39,2
18	Jelly Bean	4.3	5,2
17		4.2	17,5
16		4.1	14,7
15	Ice Cream Sandwich (ICS)	4.0.3, 4.0.4	5,1
10	Gingerbread	2.3.3–2.3.7	5,6
8	Froyo	2.2.x	0,3

(Версии Android, используемые менее чем на 0,1% устройств, в таблице не приводятся.)

За каждым выпуском с кодовым названием следуют инкрементные выпуски. Например, платформа Ice Cream Sandwich was изначально выпущена как Android 4.0 (API уровня 14) и почти немедленно заменена инкрементными выпусками, которые в конечном итоге привели к появлению Android 4.0.3 и 4.0.4 (API уровня 15).

Конечно, проценты из табл. 6.1 будут изменяться, но в них проявляется важная закономерность: устройства Android со старыми версиями не подвергаются немедленному обновлению или замене при появлении новой версии. По состоянию на июнь 2015 года более 10% устройств все еще работали под управлением Ice Cream Sandwich или Gingerbread. Версия Android 4.0.4 (последнее обновление ICS) была выпущена в мае 2012 года.

(Обновленные данные из таблицы доступны по адресу <http://developer.android.com/about/dashboards/index.html>.)

Почему на многих устройствах продолжают работать старые версии Android? В основном из-за острой конкуренции между производителями устройств Android и операторами сотовой связи. Операторы стремятся иметь возможности и телефоны, которых нет у других сетей. Производители устройств тоже испытывают давление — все их телефоны базируются на одной ОС, но им нужно выделяться на фоне конкурентов. Сочетание этого давления со стороны рынка и операторов сотовой связи привело к появлению многочисленных устройств со специализированными модификациями Android.

Устройство со специализированной версией Android не сможет перейти на новую версию, выпущенную Google. Вместо этого ему придется ждать совместимого «фирменного» обновления, которое может выйти через несколько месяцев после выпуска версии Google... А может вообще не выйти — производители часто предпочитают расходовать ресурсы на новые устройства, а не на обновление старых устройств.

Совместимость и программирование Android

Из-за задержек обновления в сочетании с регулярным выпуском новых версий совместимость становится важной проблемой в программировании Android. Чтобы привлечь широкую аудиторию, разработчики Android должны создавать приложения, которые хорошо работают на устройствах с разными версиями Android: Jelly Bean, KitKat, Lollipop и более современными версиями, а также на устройствах различных форм-факторов.

Поддержка разных размеров не столь сложна, как может показаться. Смартфоны выпускаются с экранами различных размеров, но система макетов Android хорошо приспособляется к ним. С планшетами дело обстоит сложнее, но в этом случае на помощь приходят квалификаторы конфигураций (как будет показано в главе 17). Впрочем, для устройств Android TV и Android Wear (также работающих под управлением Android) различия в пользовательском интерфейсе настолько серьезны, что разработчику приходится переосмысливать организацию взаимодействия с пользователем и дизайн приложения.

Разумный минимум

Самой старой версией Android, поддерживаемой в примерах книги, является API уровня 16 (Jelly Bean). Также встречаются упоминания более старых версий, но мы сосредоточимся на тех версиях, которые мы считаем современными (API уровня 16+). Доля устройств с версиями FroYo, Gingerbread и Ice Cream Sandwich падает от месяца к месяцу, и объем работы, необходимой для поддержки старых версий, перевешивает пользу от них.

Инкрементные выпуски не создают особых проблем с обратной совместимостью. Изменение основной версии — совсем другое дело. Работа, необходимая для поддержки только устройств 4.x, не так уж велика, но если вам также необходимо поддерживать устройства 2.x, придется потратить время на анализ различий между версиями. (За подробной информацией о поддержке Android версий 2.x обращайтесь к первому изданию этой книги.) Поддержка Android 5.0 (Lollipop) наряду с версиями 4.x потребует определенных усилий, но компания Google предоставила специальные библиотеки, которые упростят задачу. Вы узнаете о них в следующих главах.

Почему же поддержка устройств 2.x создает столько проблем? Выпуск Honeycomb, Android 3.0, стал поворотным моментом в Android, ознаменовавшим появление нового пользовательского интерфейса и новых архитектурных компонентов. Версия Honeycomb предназначалась только для планшетов, так что новые разработки получили массовое распространение только с выходом Ice Cream Sandwich. Версии, вышедшие после этого, были менее революционными.



Рис. 6.1. Еще не забыли?

Система Android поддерживает обеспечение обратной совместимости. Также существуют сторонние библиотеки, которые помогают в этом. С другой стороны, поддержание совместимости усложняет изучение программирования Android.

При создании проекта GeoQuiz в мастере нового приложения вы выбирали минимальную версию SDK (рис. 6.1). (Учтите, что в Android термины «версия SDK» и «уровень API» являются синонимами.)

Кроме минимальной поддерживаемой версии, также можно задать *целевую версию* (target version) и *версию для построения* (build version). Давайте разберемся, чем отличаются эти версии и как изменить их.

Все эти свойства задаются в файле build.gradle в модуле app. Версия для построения задается исключительно в этом файле. Минимальная и целевая версии SDK задаются в файле build.gradle, но используются для замены или инициализации значений из файла AndroidManifest.xml.

Откройте файл build.gradle, находящийся в модуле app. Обратите внимание на значения compileSdkVersion, minSdkVersion и targetSdkVersion.

Листинг 6.1. Конфигурация построения (app/build.gradle)

```
---
compileSdkVersion 22
buildToolsVersion "23.0.0"
defaultConfig {
    applicationId "com.bignerdranch.android.geoquiz"
    minSdkVersion 16
    targetSdkVersion 22
    ---
}
---
```

Минимальная версия SDK

Значение minSdkVersion определяет нижнюю границу, за которой ОС отказывается устанавливать приложение.

Выбирая API уровня 16 (Jelly Bean), вы разрешаете Android устанавливать GeoQuiz на устройствах с версией Jelly Bean и выше. Android откажется устанавливать GeoQuiz на устройстве, допустим, с системой Froyo.

Снова обращаясь к табл. 6.1, мы видим, почему API 16 является хорошим выбором для минимальной версии SDK: в этом случае приложение можно будет устанавливать на 88 % используемых устройств.

Целевая версия SDK

Значение targetSdkVersion сообщает Android, для какого уровня API *проектировалось* ваше приложение. Чаще всего в этом поле указывается новейшая версия Android.

Когда следует понижать целевую версию SDK? Новые выпуски SDK могут изменять внешний вид вашего приложения на устройстве и даже поведение ОС «за кулисами». Если ваше приложение уже спроектировано, убедитесь в том, что

в новых версиях оно работает так, как ожидалось. За информацией о возможных проблемах обращайтесь к документации по адресу http://developer.android.com/reference/android/os/Build.VERSION_CODES.html. Далее либо измените свое приложение, чтобы оно работало с новым поведением, либо понизьте целевую версию SDK. Понижение целевой версии SDK гарантирует, что приложение будет работать с внешним видом и поведением целевой версии, в которой оно хорошо работало. Этот параметр существует для обеспечения совместимости с новыми версиями Android, так как все изменения последующих версий игнорируются до увеличения `targetSdkVersion`.

Версия SDK для построения

Последний параметр SDK обозначен в листинге 6.1 именем `compileSdkVersion`. В файле `AndroidManifest.xml` этот параметр отсутствует. Если минимальная и целевая версии SDK помещаются в манифест и сообщаются ОС, версия SDK для построения относится к закрытой информации, известной только вам и компилятору.

Функциональность Android используется через классы и методы SDK. Версия SDK, используемая для построения, также называемая *целью построения* (`build target`), указывает, какая версия должна использоваться при построении вашего кода. Когда Android Studio ищет классы и методы, на которые вы ссылаетесь в директивах импорта, версия SDK для построения определяет, в какой версии SDK будет осуществляться поиск.

Лучшим вариантом цели построения является новейший уровень API (на момент написания книги 21, Lollipop). Тем не менее при необходимости цель построения существующего приложения можно изменить, например, при выпуске очередной версии Android, чтобы вы могли использовать новые методы и классы, появившиеся в этой версии.

Все эти параметры — минимальную версию SDK, целевую версию SDK, версию SDK для построения — можно изменить в файле `build.gradle`, однако следует помнить, что изменения в этом файле проявятся только после синхронизации проекта с измененными файлами. Выберите команду **Tools** ▶ **Android** ▶ **Sync Project with Gradle Files**. Проект строится заново с обновленными параметрами.

Безопасное добавление кода для более поздних версий API

Различия между минимальной версией SDK и версией SDK для построения создают проблемы совместимости, которыми необходимо управлять. Например, что произойдет в GeoQuiz при выполнении кода, рассчитанного на версию SDK после минимальной версии Jelly Bean (API уровня 16)? Если установить такое приложение и запустить его на устройстве с Jelly Bean, произойдет сбой.

Раньше тестирование в подобных ситуациях было сущим кошмаром. Однако благодаря совершенствованию Android Lint проблемы, порожденные вызовом нового кода на старых устройствах, теперь успешно обнаруживаются. Если вы использу-

ете код версии, превышающей минимальную версию SDK, Android Lint сообщит об ошибке построения.

Весь код текущей версии GeoQuiz совместим с API уровня 16 и ниже. Добавим код API уровня 21 (Lollipop) и посмотрим, что произойдет.

Откройте файл CheatActivity.java. Включите в код метод OnClickListener кнопки Show Answer следующий фрагмент для создания круговой анимации на время сокрытия кнопки:

Листинг 6.2. Добавление кода анимации (CheatActivity.java)

```
mShowAnswer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        int cx = mShowAnswer.getWidth() / 2;
        int cy = mShowAnswer.getHeight() / 2;
        float radius = mShowAnswer.getWidth();
        Animator anim = ViewAnimationUtils
            .createCircularReveal(mShowAnswer, cx, cy, radius, 0);
        anim.addListener(new AnimatorListenerAdapter() {
            @Override
            public void onAnimationEnd(Animator animation) {
                super.onAnimationEnd(animation);
                mAnswerTextView.setVisibility(View.VISIBLE);
                mShowAnswer.setVisibility(View.INVISIBLE);
            }
        });
        anim.start();
    }
});
```

Метод `createCircularReveal` создает объект `Animator` по нескольким параметрам. Сначала вы задаете объект `View`, который будет скрываться или отображаться в ходе анимации. Затем указывается центральная позиция, начальный и конечный радиус анимации. Кнопка `Show Answer` скрывается, поэтому радиус изменяется в диапазоне от ширины кнопки до 0.

Перед запуском только что созданной анимации назначается слушатель, который помогает узнать о завершении анимации. После завершения выводится ответ, а кнопка скрывается.

Наконец, анимация запускается, и начинается круговая анимация. Тема анимации подробно рассматривается в главе 30.

Класс `ViewAnimationUtils` и его метод `createCircularReveal` были добавлены в Android SDK в API level 21, поэтому при попытке выполнения этого кода на устройстве с более низкой версией произойдет сбой.

После ввода кода в листинге 6.2 Android Lint немедленно выдает предупреждение о том, что выполнение кода небезопасно при вашей минимальной версии SDK. Если предупреждение не появилось, вы можете вручную активизировать Lint командой `Analyze ▶ Inspect Code...`. Так как для построения используется API level 21, у компилятора не возникнет проблем с этим кодом. С другой стороны, Android Lint знает минимальную версию SDK и будет протестовать.

Сообщения об ошибках будут выглядеть примерно так: «Call requires API level 21 (current min is 16)». Запуск кода с этим предупреждением возможен, но Lint знает, что это небезопасно.

Как избавиться от ошибок? Первый способ — поднять минимальную версию SDK до 21. Однако тем самым вы не столько решаете проблему совместимости, сколько обходите ее. Если ваше приложение не может устанавливаться на устройствах с API уровня 16 и более старых устройствах, то проблема совместимости исчезает.

Другое, более правильное решение — заключить код более высокого уровня API в условную конструкцию, которая проверяет версию Android на устройстве.

Листинг 6.3. Предварительная проверка версии Android на устройстве

```
mShowAnswer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (mAnswerIsTrue) {
            mAnswerTextView.setText(R.string.true_button);
        } else {
            mAnswerTextView.setText(R.string.false_button);
        }
        setAnswerShownResult(true);

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
            int cx = mShowAnswer.getWidth() / 2;
            int cy = mShowAnswer.getHeight() / 2;
            float radius = mShowAnswer.getWidth();
            Animator anim = ViewAnimationUtils
                .createCircularReveal(mShowAnswer, cx, cy, radius, 0);
            anim.addListener(new AnimatorListenerAdapter() {
                @Override
                public void onAnimationEnd(Animator animation) {
                    super.onAnimationEnd(animation);
                    mAnswerTextView.setVisibility(View.VISIBLE);
                    mShowAnswer.setVisibility(View.INVISIBLE);
                }
            });
            anim.start();
        } else {
            mAnswerTextView.setVisibility(View.VISIBLE);
        }
    }
});
```

```
        mShowAnswer.setVisibility(View.INVISIBLE);
    }
});
```

Константа `Build.VERSION.SDK_INT` определяет версию Android на устройстве. Она сравнивается с константой, соответствующей Lollipop. (Коды версий доступны по адресу http://developer.android.com/reference/android/os/Build.VERSION_CODES.html.)

Теперь код анимации будет выполняться только в том случае, если приложение работает на устройстве с API уровня 21 и выше. Код стал безопасным для API уровня 16, Android Lint не на что жаловаться.

Запустите GeoQuiz на устройстве с версией Lollipop и выше и проверьте, как работает новая анимация при запуске `CheatActivity`.

Также можно запустить GeoQuiz на устройстве Jelly Bean или KitKat (виртуальном или физическом). Анимация в этом случае не отображается, но вы можете убедиться в том, что приложение работает нормально.

Документация разработчика Android

Ошибки Android Lint сообщают, к какому уровню API относится несовместимый код. Однако вы также можете узнать, к какому уровню API относятся конкретные классы и методы, — эта информация содержится в документации разработчика Android.

Постарайтесь поскорее научиться работать с документацией. Информация Android SDK слишком обширна, чтобы держать ее в голове, а с учетом регулярного появления новых версий разработчик должен знать, что нового появилось, и уметь пользоваться этими новшествами.

Документация разработчика Android — превосходный и объемный источник информации. Ее главная страница находится по адресу <http://developer.android.com/>. Документация делится на три части: проектирование (Design), разработка (Develop) и распространение (Distribute). В разделе проектирования описаны паттерны и принципы проектирования пользовательского интерфейса приложений. В разделе разработки содержится основная документация и учебные материалы. В разделе распространения объясняется, как готовить и публиковать приложения в Google Play или через механизм открытого распространения. Все это стоит просмотреть, когда у вас появится возможность.

Раздел разработки дополнительно разбит на шесть секций.

Training	Учебные модули для начинающих и опытных разработчиков, включая загружаемый код примеров
API Guides	Тематические описания компонентов приложений, функций и полезных приемов

Reference	Гипертекстовая документация по всем классам, методам, интерфейсам, константам атрибутов и т. д. в SDK, с возможностью поиска
Tools	Описания и ссылки на инструменты разработчика
Google Services	Информация о собственных API компании Google, включая Google Maps и Google Cloud Messaging
Samples	Примеры кода, демонстрирующие использование API

Для работы с документацией не обязательно иметь доступ к Интернету. Откройте в файловой системе каталог, в который были загружены SDK; в нем расположен каталог docs с полной документацией.

Чтобы определить, к какому уровню API относится ViewAnimationUtils, проведите поиск этого метода при помощи строки поиска в правом верхнем углу браузера. Вы получите результаты из нескольких категорий. Нам нужны результаты из справочной секции. Отфильтруйте результаты поиска, щелкнув на категории Reference в левой части окна.

Выберите первый результат; открывается справочная страница с описанием класса ViewAnimationUtils (рис. 6.2). В верхней части страницы находятся ссылки на разные секции.

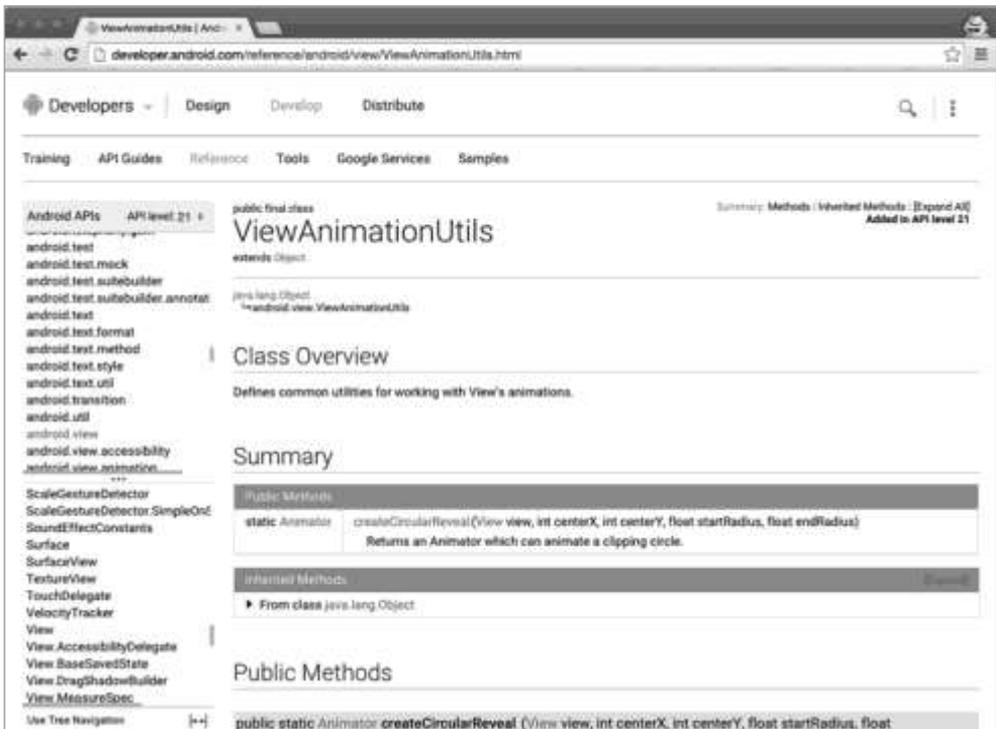


Рис. 6.2. Страница со справочным описанием ViewAnimationUtils

Прокрутите список, найдите метод `createCircularReveal(...)` и щелкните на имени метода, чтобы увидеть описание. Справа от сигнатуры метода видно, что метод `createCircularReveal(...)` появился в API уровня 21.

Если вы хотите узнать, какие методы `ViewAnimationUtils` доступны, скажем, в API уровня 16, отфильтруйте справочник по уровню API. В левой части страницы, где классы индексируются по пакету, найдите категорию `API level:21`. Щелкните на близлежащем элементе управления и выберите в списке 16. Все, что появилось в Android после API уровня 16, выделяется в списке серым цветом. В данном случае класс `ViewAnimationUtils` появился в API уровня 21, поэтому вы увидите предупреждение о том, что весь класс полностью недоступен в API уровня 16.

Фильтр уровня API приносит намного больше пользы для классов, доступных на используемом вами уровне API. Найдите в документации страницу класса `Activity`. Верните в фильтре уровня API значение 16; обратите внимание, как много методов появилось с момента выхода API 16: например, метод `onEnterAnimationComplete`, включенный в SDK в Lollipop, позволяет создавать интересные переходы между активностями.

Продолжая чтение книги, постарайтесь почаще возвращаться к документации. Она наверняка понадобится вам для выполнения упражнений, однако не стесняйтесь заниматься самостоятельными исследованиями каждый раз, когда вам захочется побольше узнать о некотором классе, методе и т. д. Создатели Android постоянно обновляют и совершенствуют свою документацию, и вы всегда узнаете из нее что-то новое.

Упражнение. Вывод версии построения

Добавьте в макет `GeoQuiz` виджет `TextView` для вывода уровня API устройства, на котором работает программа. На рис. 6.3 показано, как должен выглядеть результат.

Задать текст `TextView` в макете невозможно, потому что версия построения устройства не известна до момента выполнения. Найдите метод `TextView` для задания текста в справочной странице `TextView` в документации Android. Вам нужен метод, получающий один аргумент — строку (или `CharSequence`).

Для настройки размера и гарнитуры текста используйте атрибуты XML, перечисленные в описании `TextView`.

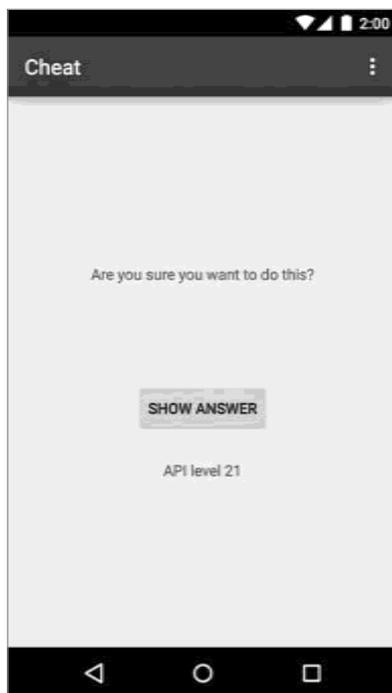


Рис. 6.3. Результат упражнения

7

UI-фрагменты и FragmentManager

В этой главе мы начнем строить приложение CriminalIntent. Оно предназначено для хранения информации об «офисных преступлениях»: грязной посуде, оставленной в раковине, или пустом лотке общего принтера после печати документов.

В приложении CriminalIntent пользователь создает запись о преступлении с заголовком, датой и фотографией. Также можно выбрать подозреваемого в списке контактов и отправить жалобу по электронной почте, опубликовать ее в Twitter, Facebook или другом приложении. Сообщив о преступлении, пользователь освобождается от негатива и может сосредоточиться на текущей задаче.

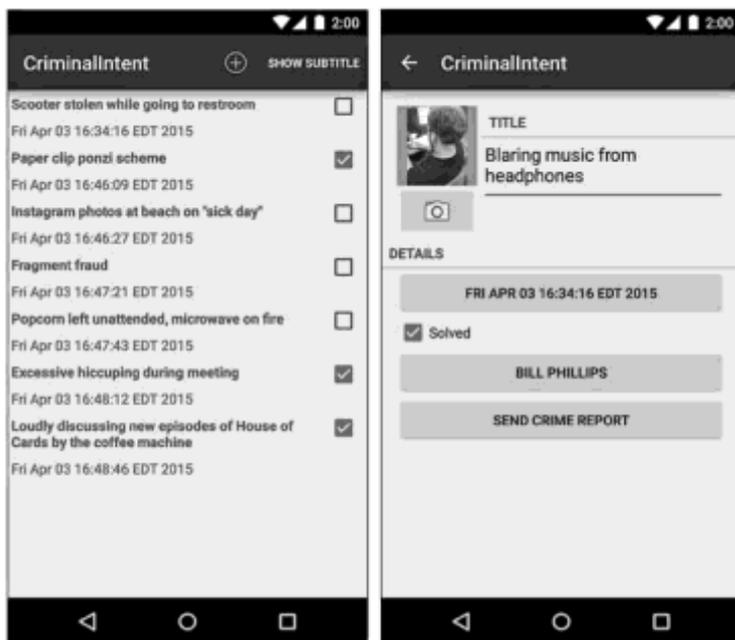


Рис. 7.1. Приложение CriminalIntent

CriminalIntent — сложное приложение, для построения которого нам понадобится целых 13 глав. В нем используется интерфейс типа «список/детализация»: на главном экране выводится список зарегистрированных преступлений. Пользователь может добавить новое или выбрать существующее преступление для просмотра и редактирования информации (рис. 7.1).

Гибкость пользовательского интерфейса

Разумно предположить, что приложение типа «список/детализация» состоит из двух активностей: для управления списком и для управления детализированным представлением. Щелчок на преступлении в списке запускает экземпляр детализированной активности. Нажатие кнопки Back уничтожает активность детализации и возвращает на экран список, в котором можно выбрать другое преступление.

Такая архитектура работает, но что делать, если вам потребуется более сложная схема представления информации и навигации между экранами?

- 1 Допустим, пользователь запустил приложение CriminalIntent на планшете. Экраны планшетов и некоторых больших телефонов достаточно велики для одновременного отображения списка и детализации, по крайней мере в альбомной ориентации (рис. 7.2).



Рис. 7.2. Идеальный интерфейс «список/детализация» для телефонов и планшетов

- 1 Пользователь просматривает описание преступления на телефоне и хочет увидеть следующее преступление в списке. Было бы удобно, если бы пользователь мог провести пальцем по экрану, чтобы перейти к следующему преступлению без возвращения к списку. Каждый жест прокрутки должен обновлять детализированное представление информацией о следующем преступлении.

Эти сценарии объединяет гибкость пользовательского интерфейса: возможность формирования и изменения представления активности во время выполнения в зависимости от того, что нужно пользователю или устройству.

Подобная гибкость в активности не предусмотрена. Представления активности могут изменяться во время выполнения, но код, управляющий этими изменениями, должен находиться в представлении. В результате активность тесно связывается с конкретным экраном, с которым работает пользователь.

Знакомство с фрагментами

Закон Android нельзя нарушить, но можно обойти, передав управление пользовательским интерфейсом приложения от активности одному или нескольким *фрагментам* (fragments).

Фрагмент представляет собой объект контроллера, которому активность может доверить выполнение операций. Чаще всего такой операцией является управление пользовательским интерфейсом — целым экраном или его частью.

Фрагмент, управляющий пользовательским интерфейсом, называется *UI-фрагментом*. UI-фрагмент имеет собственное представление, которое заполняется на основании файла макета. Представление фрагмента содержит элементы пользовательского интерфейса, с которыми будет взаимодействовать пользователь.

Представление активности содержит место, в которое вставляется представление фрагмента, или несколько мест для представлений нескольких фрагментов.

Фрагменты, связанные с активностью, могут использоваться для формирования и изменения экрана в соответствии с потребностями приложения и пользователей. Представление активности формально остается неизменным на протяжении жизненного цикла, и законы Android не нарушаются.

Давайте посмотрим, как эта схема работает в приложении «список/детализация». Представление активности строится из фрагмента списка и фрагмента детализации. Представление детализации содержит подробную информацию о выбранном элементе списка.

При выборе другого элемента на экране появляется новое детализированное представление. С фрагментами эта задача решается легко; активность заменяет фрагмент детализации другим фрагментом детализации (рис. 7.3). Это существенное изменение представления происходит без уничтожения активности.



Рис. 7.3. Переключение фрагмента детализации

Применение UI-фрагментов позволяет разделить пользовательский интерфейс вашего приложения на структурные блоки, а это полезно не только для приложений «список/детализация». Работа с отдельными блоками упрощает построение интерфейсов со вкладками, анимированных боковых панелей и многих других.

Впрочем, за такую гибкость пользовательского интерфейса приходится платить повышением сложности, увеличением количества «подвижных частей», увеличением объема кода. Выгода из использования фрагментов проявится в главах 11 и 17, зато разбираться со сложностью придется прямо сейчас.

Начало работы над CriminalIntent

В этой главе мы возьмемся за представление детализации CriminalIntent. На рис. 7.4 показано, как будет выглядеть приложение CriminalIntent к концу главы.

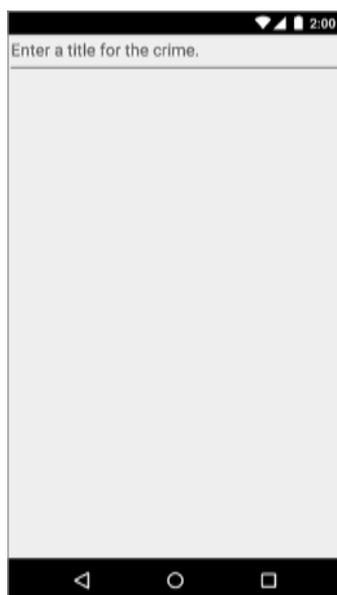


Рис. 7.4. Приложение CriminalIntent к концу главы

На первый взгляд результат не впечатляет. Однако не забывайте, что эта глава всего лишь закладывает основу для более серьезных дел в будущем.

Экраном, показанным на рис. 7.4, будет управлять UI-фрагмент с именем CrimeFragment. Хостом (host) экземпляра CrimeFragment является активность с именем CrimeActivity.

Пока считайте, что хост предоставляет позицию в иерархии представлений, в которой фрагмент может разместить свое представление (рис. 7.5). Фрагмент не может вывести представление на экран сам по себе. Его представление отображается только при размещении в иерархии активности.

Проект CriminalIntent будет большим; диаграмма объектов поможет понять логику его работы. На рис. 7.6 изображена общая структура CriminalIntent. Запоминать все объекты и связи между ними не обязательно, но прежде чем выходить в путь, полезно хотя бы в общих чертах понимать, куда вы направляетесь.

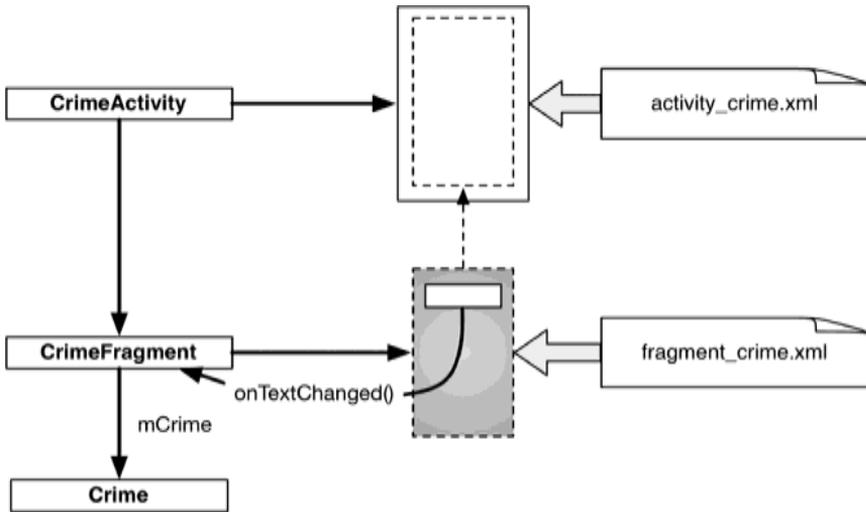


Рис. 7.5. CrimeActivity как хост CrimeFragment

Мы видим, что класс CrimeFragment делает примерно то же, что в GeoQuiz делали активности: он создает пользовательский интерфейс и управляет им, а также обеспечивает взаимодействие с объектами модели.

Мы напишем три класса, изображенных на рис. 7.6: Crime, CrimeFragment и CrimeActivity.

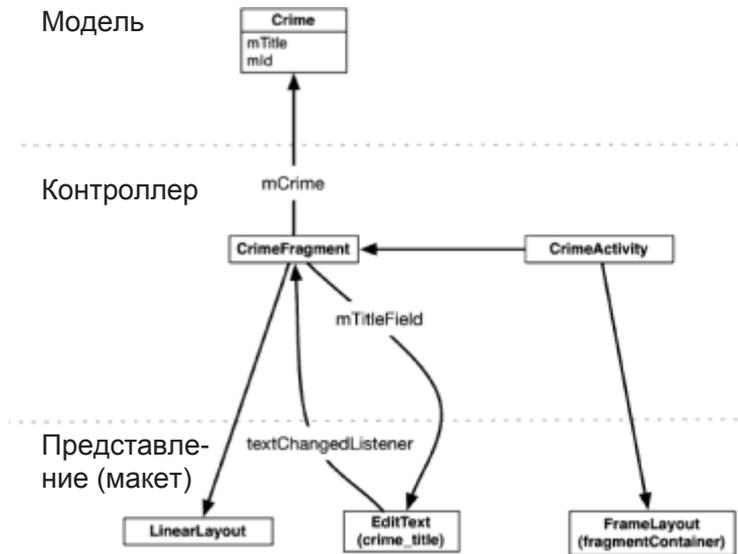


Рис. 7.6. Диаграмма объектов CriminalIntent (для этой главы)

Экземпляр `Crime` представляет одно офисное преступление. В этой главе описание преступления будет состоять только из заголовка и идентификатора. Заголовок содержит содержательный текст (например, «Свалка химических отходов в раковине» или «Кто-то украл мой йогурт!»), а идентификатор однозначно идентифицирует экземпляр `Crime`.

В этой главе мы для простоты будем использовать один экземпляр `Crime`. В класс `CrimeFragment` включается поле (`mCrime`) для хранения этого отдельного инцидента.

Представление `CrimeActivity` состоит из элемента `FrameLayout`, определяющего место, в котором будет отображаться представление `CrimeFragment`.

Представление `CrimeFragment` будет состоять из элементов `LinearLayout` и `EditText`. `CrimeFragment` определяет поле для виджета `EditText` (`mTitleField`) и назначает для него слушателя, обновляющего уровень модели при изменении текста.

Создание нового проекта

Но довольно разговоров; пора построить новое приложение. Создайте новое приложение Android (File ► New Project...). Введите имя приложения `CriminalIntent` и имя пакета `com.bignerdranch.android.criminalintent`, как показано на рис. 7.7.

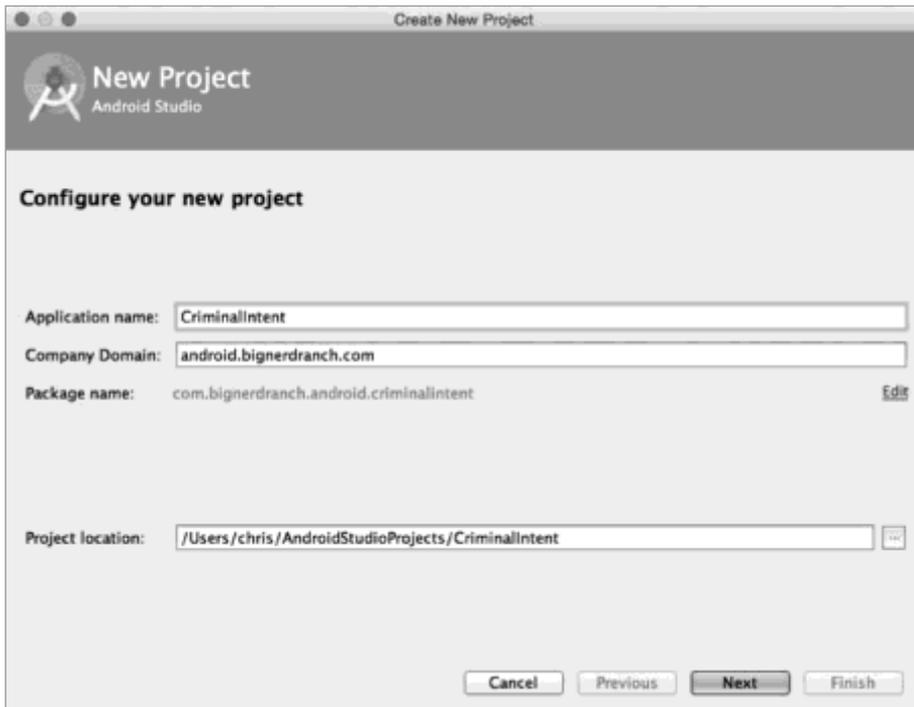


Рис. 7.7. Создание приложения `CriminalIntent`

Щелкните на кнопке Next и задайте минимальный уровень SDK: API 16: Android 4.1. Также проследите затем, чтобы для типов приложений были установлены только флажки Phone и Tablet.

Снова щелкните на кнопке Next, чтобы выбрать разновидность добавляемой активности. Выберите пустую активность (Blank Activity) и продолжайте работу с мастером.

На последнем шаге мастера введите имя активности CrimeActivity и щелкните на кнопке Finish (рис. 7.8).



Рис. 7.8. Настройка CrimeActivity

Фрагменты и библиотека поддержки

Фрагменты появились в API уровня 11 вместе с первыми планшетами на базе Android и неожиданной потребностью в гибкости пользовательского интерфейса. На ранней стадии программирования для Android, описанной в первом издании этой книги, многим разработчикам приходилось поддерживать устройства с API уровня 8 и выше. К счастью, эти разработчики все равно могли пользоваться фрагментами благодаря библиотеке поддержки Android.

Библиотека поддержки (support library) включает полную реализацию фрагментов вплоть до API уровня 4. В этой книге мы будем использовать реализацию фрагментов из библиотеки поддержки вместо реализации, встроенной в ОС Android, — это правильно, потому что библиотека поддержки быстро обновляется при включении новых возможностей в API фрагментов. Чтобы использовать новые возможности, вы можете обновить свой проект последней версией библиотеки поддержки. Подробные обоснования этого решения приведены в разделе «Для любознательных: почему фрагменты из библиотеки поддержки лучше». Учтите, что класс из библиотеки поддержки используется не только в старых версиях при отсутствии «родного» класса; он также используется вместо него в новых версиях.

В библиотеку поддержки входят два ключевых класса: `Fragment` (`android.support.v4.app.Fragment`) и `FragmentActivity` (`android.support.v4.app.FragmentActivity`). Для использования фрагментов нужны активности, которые умеют управлять фрагментами. Класс `FragmentActivity` знает, как управлять реализацией фрагментов из библиотеки поддержки.

На рис. 7.9 приведены имена этих классов и их местонахождение. Поскольку библиотека поддержки (и `android.support.v4.app.Fragment`) находится в вашем приложении, ее можно безопасно использовать независимо от того, где работает приложение.

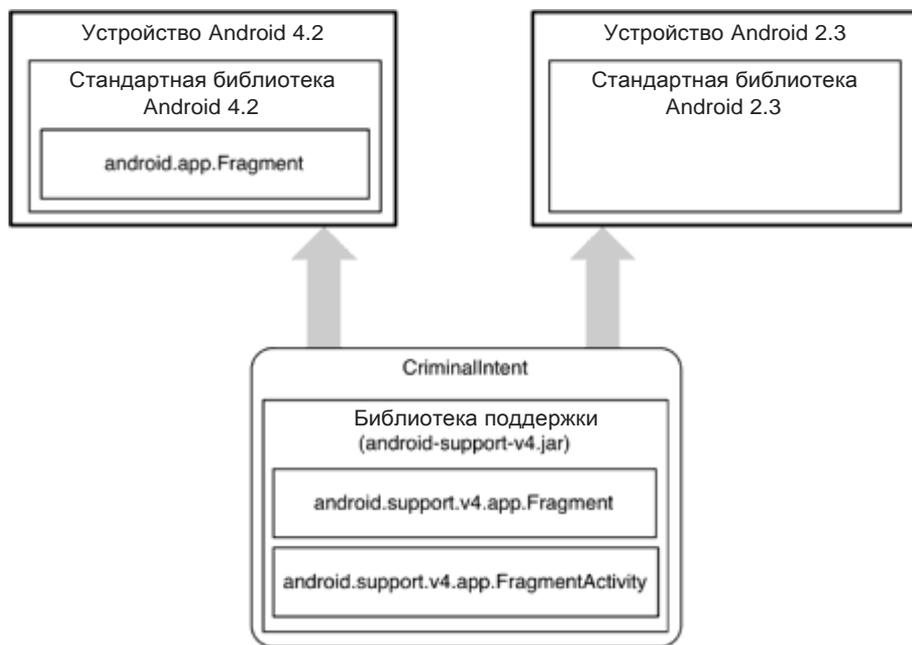


Рис. 7.9. Местонахождение разных классов фрагментов

Добавление зависимостей в Android Studio

Чтобы использовать библиотеку поддержки, необходимо включить ее в состав зависимостей проекта. Откройте файл `build.gradle` из модуля `app`. Ваш проект содержит два файла `build.gradle`: один для проекта в целом, другой для модуля `app`. Мы отредактируем файл, находящийся в `app/build.gradle`.

Листинг 7.1. Зависимости `gradle` (`app/build.gradle`)

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

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

Текущее содержимое секции `dependencies` файла `build.gradle` должно примерно соответствовать листингу 7.1: оно означает, что проект зависит от всех файлов `jar` из каталога `libs` проекта.

Gradle также позволяет задавать зависимости, не скопированные в ваш проект. Во время компиляции проекта Gradle находит, загружает и включает такие зависимости за вас. От вас потребуется лишь правильно задать волшебную строку зависимости, а Gradle сделает все остальное.

Впрочем, запоминать эти строки никому не хочется, поэтому Android Studio поддерживает список часто используемых библиотек. Перейдите к структуре проекта (`File ▶ Project Structure...`). Выберите в левой части модуль `app`, а в нем перейдите на вкладку `Dependencies`. На этой вкладке перечислены зависимости модуля `app`. (Также здесь могут быть приведены другие зависимости, как, например, зависимость `AppCompat` на рис. 7.10. Если в вашем списке присутствуют другие зависимости, не удаляйте их. Библиотека `AppCompat` более подробно рассматривается в главе 13.)

Здесь могут быть указаны дополнительные зависимости, например зависимость от `AppCompat`. Если они присутствуют, не удаляйте их. Библиотека `AppCompat` описана в главе 13.

Нажмите кнопку `+` и выберите зависимость `Library`, чтобы создать новую зависимость (рис. 7.11). Выберите в списке библиотеку `support-v4` и щелкните на кнопке `OK`.

Вернитесь в окно редактора с содержимым `app/build.gradle`; добавленная библиотека появляется в списке (листинг 7.2).

Листинг 7.2. Зависимости `gradle` (`app/build.gradle`)

```
...
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile 'com.android.support:support-v4:22.1.1'
}
```

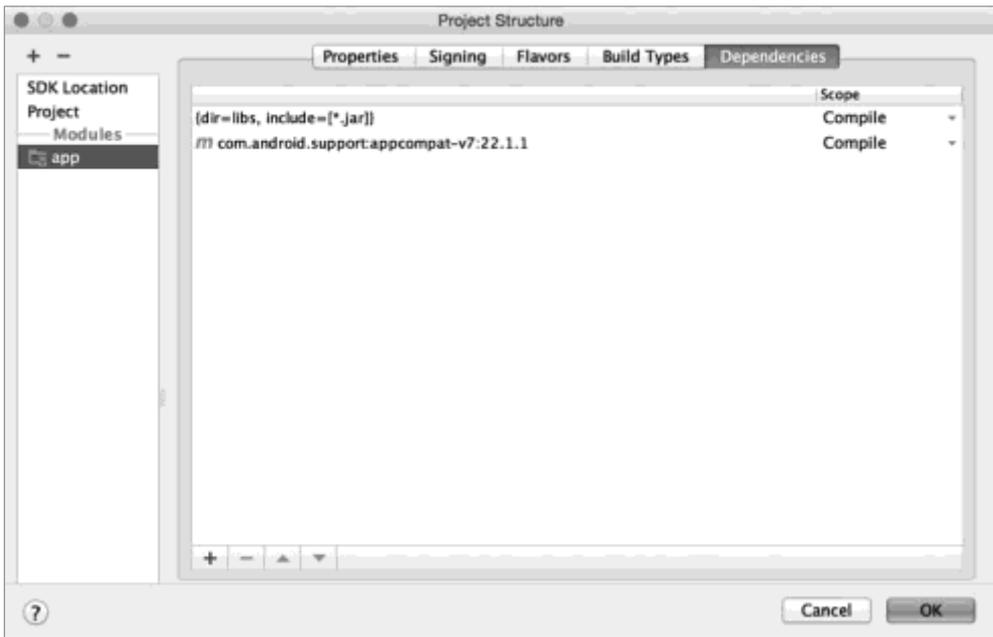


Рис. 7.10. Зависимости приложения

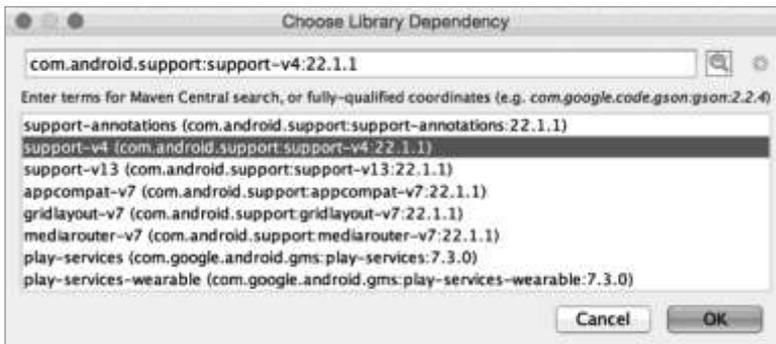


Рис. 7.11. Набор зависимостей

(Если файл редактируется вручную, за пределами окна инструментов Project, проект нужно будет синхронизировать с файлом Gradle. В процессе синхронизации Gradle обновляет построение с учетом изменений, загружая или удаляя необходимые зависимости. При внесении изменений в окне инструментов Project синхронизация запускается автоматически. Чтобы выполнить синхронизацию вручную, выполните команду Tools ▶ Android ▶ Sync Project with Gradle Files.)

В выделенной строке зависимости в листинге 7.2 используется формат координат Maven: *группа:артефакт:версия*. (Maven — программа управления зависимо-

стями; дополнительную информацию о ней можно найти по адресу [https://maven.apache.org/.](https://maven.apache.org/))

Поле *группа* содержит уникальный идентификатор набора библиотек, доступных в репозитории Maven. Часто в качестве идентификатора группы используется базовое имя пакета библиотеки — в данном случае `com.android.support`. Поле *артефакт* содержит имя конкретной библиотеки в пакете. В нашем примере используется имя `support-v4`. Пакет `com.android.support` содержит несколько библиотек: `support-v13`, `appcompat-v7`, `gridlayout-v7` и т. д. Google строит имена библиотек поддержки по схеме *basename-vX*, где *-vX* — минимальный уровень API, поддерживаемый библиотекой. Например, `appcompat-v7` — библиотека совместимости Google для устройств с Android API версии 7 и выше.

Наконец, поле *версия* представляет номер версии библиотеки. Приложение `CriminalIntent` зависит от версии 22.1.1 библиотеки `support-v4`. На момент написания книги последней была версия 22.1.1, но любая более новая версия должна работать в этом проекте. Обычно стоит использовать последнюю версию библиотеки поддержки, чтобы вам были доступны более новые API и получать свежие исправления. Если Android Studio добавит обновленную версию библиотеки, не возвращайтесь к версии, приведенной в листинге.

Итак, библиотека поддержки вошла в число зависимостей проекта; теперь ее нужно использовать. В окне инструментов Project найдите и откройте файл `CrimeActivity.java`. Выберите `FragmentActivity` суперклассом `CrimeActivity`. Заодно удалите шаблонные реализации `onOptionsItemSelected(Menu)` и `onOptionsItemSelected(Menuitem)`. (Меню параметров для приложения `CriminalIntent` будет построено в главе 13.)

Листинг 7.3. Настройка шаблонного кода (`CrimeActivity.java`)

```
public class CrimeActivity extends AppCompatActivity FragmentActivity {

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

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

    @Override
    public boolean onOptionsItemSelected(Menuitem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Прежде чем продолжать с `CrimeActivity`, мы создадим уровень модели `CriminalIntent`. Для этого мы напишем класс `Crime`.

Создание класса `Crime`

В окне инструментов Project щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.criminalintent` и выберите команду `New ▶ Java Class`. Введите имя класса `Crime` и щелкните на кнопке `OK`.

Добавьте в `Crime.java` следующий код.

Листинг 7.4. Добавление кода в класс `Crime` (`Crime.java`)

```
public class Crime {  
  
    private UUID mId;  
    private String mTitle;  
  
    public Crime() {  
        // Генерирование уникального идентификатора  
        mId = UUID.randomUUID();  
    }  
}
```

Затем для свойства `mId`, доступного только для чтения, необходимо сгенерировать только `get`-метод, а для свойства `mTitle` — `get`- и `set`-методы. Щелкните правой кнопкой мыши после конструктора, выберите команду `Generate... ▶ Getter` и затем переменную `mId`. Затем сгенерируйте `get`- и `set`-метод для `mTitle`: повторите этот процесс, но выберите в меню `Generate...` команду `Getter and Setter`.

Листинг 7.5. Сгенерированные `get`- и `set`-методы (`Crime.java`)

```
public class Crime {  
    private UUID mId;  
  
    private String mTitle;  
  
    public Crime() {  
        mId = UUID.randomUUID();  
    }  
  
    public UUID getId() {  
        return mId;  
    }  
  
    public String getTitle() {  
        return mTitle;  
    }  
  
    public void setTitle(String title) {  
        mTitle = title;  
    }  
}
```

Вот и все, что нам понадобится для класса Crime и уровня модели CriminalIntent в этой главе.

Итак, мы создали уровень модели и активность, которая выполняет хостинг фрагмента. А теперь более подробно рассмотрим, как активность выполняет свои функции хоста.

Хостинг UI-фрагментов

Чтобы стать хостом для UI-фрагмента, активность должна:

- ① определить место представления фрагмента в своем макете;
- ① управлять жизненным циклом экземпляра фрагмента.

Жизненный цикл фрагмента

На рис. 7.12 показан жизненный цикл фрагмента. Он имеет много общего с жизненным циклом активности: он тоже может находиться в состоянии остановки, приостановки и выполнения, он тоже содержит методы, переопределяемые для выполнения операций в критических точках, многие из которых соответствуют методам жизненного цикла активности.

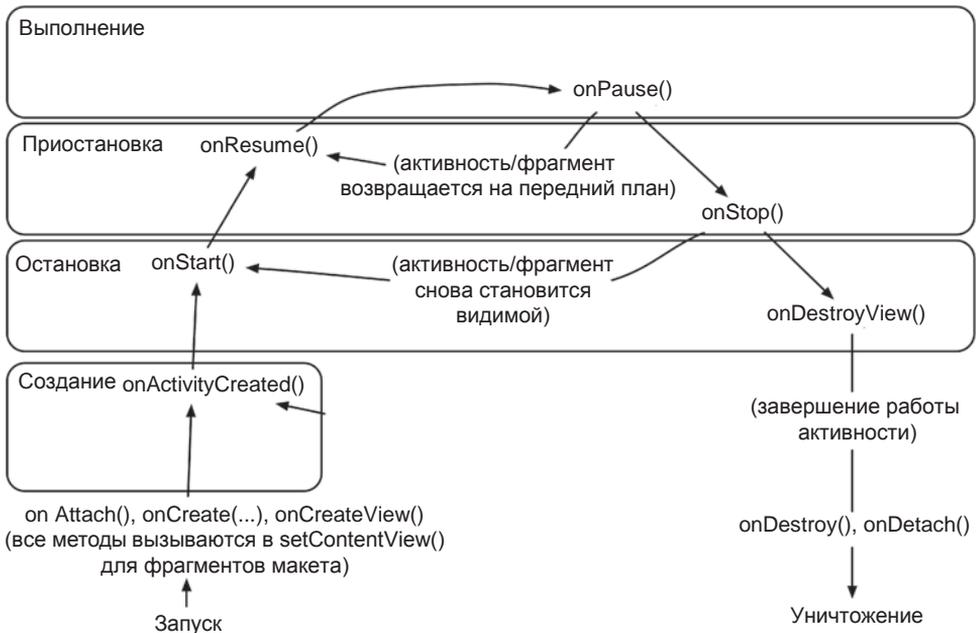


Рис. 7.12. Жизненный цикл фрагмента

Эти соответствия играют важную роль. Так как фрагмент работает по поручению активности, его состояние должно отражать текущее состояние активности. Следовательно, ему необходимы соответствующие методы жизненного цикла для выполнения работы активности.

Принципиальное различие между жизненными циклами фрагмента и активности заключается в том, что методы жизненного цикла фрагмента вызываются активностью-хостом, а не ОС. ОС ничего не знает о фрагментах, используемых активностью; фрагменты — внутреннее дело самой активности.

Методы жизненного цикла фрагментов будут более подробно рассмотрены, когда мы продолжим строить приложение `CriminalIntent`.

Два способа организации хостинга

Существует два основных способа организации хостинга UI-фрагментов в активности:

- ❶ добавление фрагмента в *макет* активности;
- ❶ добавление фрагмента в код активности.

Первый способ — с использованием так называемых *макетных фрагментов* — прост, но недостаточно гибок. Включение фрагмента в макет активности означает жесткую привязку фрагмента и его представления к представлению активности, и вам уже не удастся переключить этот фрагмент на протяжении жизненного цикла активности.

Второй способ сложнее, но только он позволяет управлять фрагментами во время выполнения. Разработчик сам определяет, когда фрагмент добавляется в активность и что с ним происходит после этого. Он может удалить фрагмент, заменить его другим фрагментом, а потом снова вернуть первый.

Итак, для достижения настоящей гибкости пользовательского интерфейса необходимо добавить фрагмент в коде. Именно этот способ мы используем для того, чтобы сделать `CrimeActivity` хостом `CrimeFragment`. Подробности реализации кода будут описаны позднее в этой главе, но сначала мы определим макет `CrimeActivity`.

Определение контейнерного представления

Мы добавим UI-фрагмент в коде активности-хоста, но мы все равно должны найти место для представления фрагмента в иерархии представлений активности. В макете `CrimeActivity` этим местом будет элемент `FrameLayout`, изображенный на рис. 7.13.

Элемент `FrameLayout` станет *контейнерным представлением* для `CrimeFragment`. Обратите внимание: контейнерное представление абсолютно универсально; оно не привязывается к классу `CrimeFragment`. Мы можем (и будем) использовать один макет для хостинга разных фрагментов.

Найдите макет `CrimeActivity` в файле `res/layout/activity_crime.xml`. Откройте файл и замените макет по умолчанию элементом `FrameLayout`, изображенным на рис. 7.13. Разметка XML должна совпадать с приведенной в листинге 7.6.

```

FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/fragment_container"
android:layout_width="match_parent"
android:layout_height="match_parent"

```

Рис. 7.13. Хостинг фрагмента для CrimeActivity

Листинг 7.6. Создание контейнера фрагмента (activity_crime.xml)

```

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>

```

Хотя activity_crime.xml состоит исключительно из контейнерного представления одного фрагмента, макет активности может быть более сложным; он может определять несколько контейнерных представлений, а также собственные виджеты.

Посмотрите файл макета или запустите CriminalIntent, чтобы проверить свой код. Вы увидите только пустой элемент FrameLayout, потому что CrimeActivity еще не выполняет функции хоста фрагмента (рис. 7.14).

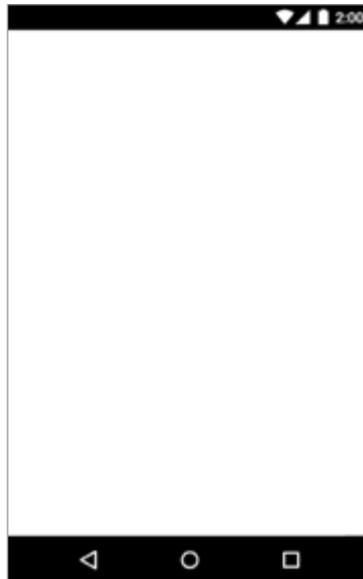


Рис. 7.14. Пустой элемент FrameLayout

Позднее мы напишем код, помещающий представление фрагмента в этот элемент FrameLayout. Но сначала фрагмент нужно создать.

Создание UI-фрагмента

Последовательность действий по созданию UI-фрагмента не отличается от последовательности действий по созданию активности:

- 1 построение интерфейса посредством определения виджетов в файле макета;
- 1 создание класса и назначение макета, который был определен ранее, его представлением;
- 1 подключение виджетов, заполненных на основании макета в коде.

Определение макета CrimeFragment

Представление `CrimeFragment` будет отображать информацию, содержащуюся в экземпляре `Crime`. Со временем в класс `Crime` и представление класса `CrimeFragment` добавится много интересного, а в этой главе мы ограничимся текстовым полем для хранения заголовка.

На рис. 7.15 изображен макет представления `CrimeFragment`. Он состоит из вертикального элемента `LinearLayout`, содержащего `EditText` — виджет с областью для ввода и редактирования текста.

Чтобы создать файл макета, щелкните правой кнопкой мыши на папке `res/layout` в окне инструментов `Project` и выберите команду `New > Layout resource file`. Придайте файлу фрагмента имя `fragment_crime.xml`. Выберите корневым элементом `LinearLayout` и щелкните на кнопке `OK`; `Android Studio` сгенерирует файл за вас.

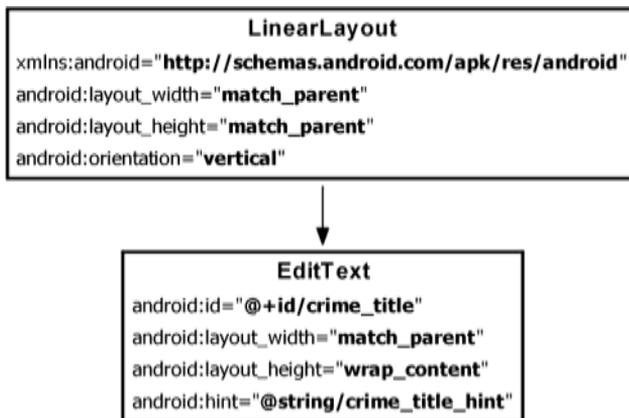


Рис. 7.15. Исходный макет `CrimeFragment`

Когда файл откроется, перейдите к разметке XML. Мастер уже добавил элемент `LinearLayout` за вас. Руководствуясь рис. 7.15, внесите необходимые изменения в `fragment_crime.xml`. Проверьте результаты по листингу 7.7.

Листинг 7.7. Файл макета для представления фрагмента (fragment_crime.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/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/crime_title_hint"
        />
</LinearLayout>
```

Откройте файл res/values/strings.xml, добавьте строковый ресурс crime_title_hint.

Листинг 7.8. Добавление и удаление строк (res/values/strings.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">CriminalIntent</string>
    <string name="hello_world">Hello world!</string>
    <string name="menu_settings">Settings</string>
    <string name="title_activity_crime">CrimeActivity</string>
    <string name="crime_title_hint">Enter a title for the crime.</string>
</resources>
```

Сохраните файлы. Вернитесь к fragment_crime.xml, чтобы увидеть предварительное изображение представления фрагмента.

Создание класса CrimeFragment

Щелкните правой кнопкой мыши на пакете com.bignerdranch.android.criminalintent и выберите команду New ▶ Java Class. Введите имя класса CrimeFragment и щелкните на кнопке ОК, чтобы сгенерировать класс.

Теперь этот класс нужно преобразовать во фрагмент. Измените класс CrimeFragment так, чтобы он субклассировал Fragment.

Листинг 7.9. Субклассирование класса Fragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
}
```

При субклассировании Fragment вы заметите, что Android Studio находит два класса с именем Fragment: Fragment (android.app) и Fragment (android.support.v4.app). Версия Fragment из android.app реализует версию фрагментов, встроенную в ОС Android. Мы используем версию библиотеки поддержки, поэтому при появлении диалогового окна выберите версию класса Fragment из android.support.v4.app (рис. 7.16).

Ваш код должен выглядеть так, как показано в листинге 7.10.


```

        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }
}

```

В этой реализации стоит обратить внимание на пару моментов. Во-первых, метод `Fragment.onCreate(Bundle)` объявлен открытым, тогда как метод `Activity.onCreate(Bundle)` объявлен защищенным. `Fragment.onCreate(...)` и другие методы жизненного цикла `Fragment` должны быть открытыми, потому что они будут вызываться произвольной активностью, которая станет хостом фрагмента.

Во-вторых, как и в случае с активностью, фрагмент использует объект `Bundle` для сохранения и загрузки состояния. Вы можете переопределить `Fragment.onSaveInstanceState(Bundle)` для ваших целей, как и метод `Activity.onSaveInstanceState(Bundle)`.

Обратите внимание на то, что *не происходит* в `Fragment.onCreate(...)`: мы не заполняем представление фрагмента. Экземпляр фрагмента настраивается в `Fragment.onCreate(...)`, но создание и настройка представления фрагмента осуществляются в другом методе жизненного цикла фрагмента:

```

public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState)

```

Именно в этом методе заполняется макет представления фрагмента, а заполненный объект `View` возвращается активности-хосту. Параметры `LayoutInflater` и `ViewGroup` необходимы для заполнения макета. Объект `Bundle` содержит данные, которые используются методом для воссоздания представления по сохраненному состоянию.

В файле `CrimeFragment.java` добавьте реализацию `onCreateView(...)`, которая заполняет разметку `fragment_crime.xml`.

Листинг 7.12. Переопределение `onCreateView(...)` (`CrimeFragment.java`)

```

public class CrimeFragment extends Fragment {
    private Crime mCrime;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, container, false);
        return v;
    }
}

```

В методе `onCreateView(...)` мы явно заполняем представление фрагмента, вызывая `LayoutInflater.inflate(...)` с передачей идентификатора ресурса макета. Вторым параметром определяется родитель представления, что обычно необходимо для правильной настройки виджета. Третьим параметром указывается, нужно ли включать заполненное представление в родителя. Мы передаем `false`, потому что представление будет добавлено в код активности.

Подключение виджетов во фрагменте

В методе `onCreateView(...)` также настраивается реакция виджета `EditText` на ввод пользователя. После того как представление будет заполнено, метод получает ссылку на `EditText` и добавляет слушателя.

Листинг 7.13. Настройка виджета `EditText` (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View v = inflater.inflate(R.layout.fragment_crime, parent, false);

        mTitleField = (EditText)v.findViewById(R.id.crime_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(
                CharSequence c, int start, int count, int after) {
                // Здесь намеренно оставлено пустое место
            }

            @Override
            public void onTextChanged(
                CharSequence c, int start, int before, int count) {
                mCrime.setTitle(c.toString());
            }

            @Override
            public void afterTextChanged(Editable c) {
                // И здесь тоже
            }
        });

        return v;
    }
}
```

Получение ссылок в `Fragment.onCreateView(...)` происходит практически так же, как в `Activity.onCreate(...)`. Единственное различие заключается в том, что для представления фрагмента вызывается метод `View.findViewById(int)`. Метод `Activity.findViewById(int)`, который мы использовали ранее, является вспомогательным методом, вызывающим `View.findViewById(int)` в своей внутренней реализации. У класса `Fragment` аналогичного вспомогательного метода нет, поэтому приходится вызывать основной метод.

Назначение слушателей во фрагменте работает точно так же, как в активности. В листинге 7.13 мы создаем анонимный класс, который реализует интерфейс слушателя `TextWatcher`. Этот интерфейс содержит три метода, но нас интересует только один: `onTextChanged(...)`.

В методе `onTextChanged(...)` мы вызываем `toString()` для объекта `CharSequence`, представляющего ввод пользователя. Этот метод возвращает строку, которая затем используется для задания заголовка `Crime`.

Код `CrimeFragment` готов. Было бы замечательно, если бы вы могли запустить `CriminalIntent` и поэкспериментировать с написанным кодом. К сожалению, это невозможно — фрагменты не могут выводить свои представления на экран. Сначала необходимо добавить `CrimeFragment` в `CrimeActivity`.

Добавление UI-фрагмента в FragmentManager

Когда в Honeycomb появился класс `Fragment`, в класс `Activity` были внесены изменения: в него был добавлен компонент, называемый `FragmentManager`. Он отвечает за управление фрагментами и добавление их представлений в иерархию представлений активности (рис. 7.17).

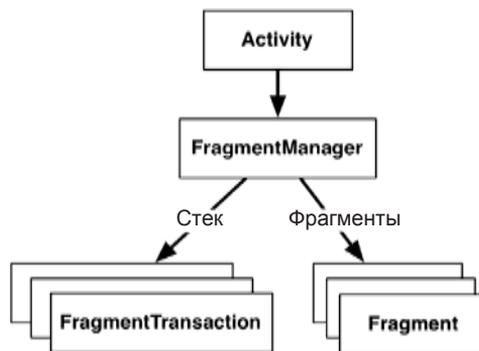


Рис. 7.17. `FragmentManager`

`FragmentManager` управляет двумя структурами: списком фрагментов и стеком транзакций фрагментов (о котором я вскоре расскажу).

В приложении `CriminalIntent` нас интересует только список фрагментов `FragmentManager`.

Чтобы добавить фрагмент в активность в коде, следует обратиться с вызовом к объекту `FragmentManager` активности. Прежде всего необходимо получить сам объект `FragmentManager`. В `CrimeActivity.java` включите следующий код в `onCreate(...)`.

Листинг 7.14. Получение объекта `FragmentManager` (`CrimeActivity.java`)

```
public class CrimeActivity extends FragmentActivity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_crime);  
  
        FragmentManager fm = getSupportFragmentManager();  
    }  
}
```

Если после добавления этой строки возникнет ошибка, проверьте директивы `import` и убедитесь в том, что импортируется версия класса `FragmentManager` из библиотеки поддержки.

Мы вызываем `getSupportFragmentManager()`, потому что в приложении используется библиотека поддержки и класс `FragmentActivity`. Если бы мы не использовали библиотеку поддержки, то вместо этого можно было бы субклассировать `Activity` и вызвать `getFragmentManager()`.

Транзакции фрагментов

После получения объекта `FragmentManager` добавьте следующий код, который передает ему фрагмент для управления. (Позднее мы рассмотрим этот код более подробно, а пока просто включите его в приложение.)

Листинг 7.15. Добавление `CrimeFragment` (`CrimeActivity.java`)

```
public class CrimeActivity extends FragmentActivity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_crime);  
  
        FragmentManager fm = getSupportFragmentManager();  
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);  
  
        if (fragment == null) {  
            fragment = new CrimeFragment();  
            fm.beginTransaction()  
                .add(R.id.fragmentContainer, fragment)  
                .commit();  
        }  
    }  
}
```

Разбираться в коде, добавленном в листинге 7.15, лучше всего не с начала. Найдите операцию `add(...)` и окружающий ее код. Этот код создает и закрепляет *транзакцию фрагмента*.

Листинг 7.16. Транзакция фрагмента (`CrimeActivity.java`)

```
if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
```

Транзакции фрагментов используются для добавления, удаления, присоединения, отсоединения и замены фрагментов в списке фрагментов. Они лежат в основе механизма использования фрагментов для формирования и модификации экранов во время выполнения. `FragmentManager` ведет стек транзакций, по которому вы можете перемещаться.

Метод `FragmentManager.beginTransaction()` создает и возвращает экземпляр `FragmentTransaction`. Класс `FragmentTransaction` использует *динамичный интерфейс*: методы, настраивающие `FragmentTransaction`, возвращают `FragmentTransaction` вместо `void`, что позволяет объединять их вызовы в цепочку. Таким образом, выделенный код в листинге 7.12 означает: «Создать новую транзакцию фрагмента, включить в нее одну операцию `add`, а затем закрепить».

Метод `add(...)` является основным содержанием транзакции. Он получает два параметра: идентификатор контейнерного представления и недавно созданный объект `CrimeFragment`. Идентификатор контейнерного представления вам должен быть знаком: это идентификатор ресурса элемента `FrameLayout`, определенного в файле `activity_crime.xml`.

Идентификатор контейнерного представления выполняет две функции:

- ❶ сообщает `FragmentManager`, где в представлении активности должно находиться представление фрагмента;
- ❷ обеспечивает однозначную идентификацию фрагмента в списке `FragmentManager`.

Когда вам потребуется получить экземпляр `CrimeFragment` от `FragmentManager`, запросите его по идентификатору контейнерного представления.

```
FragmentManager fm = getSupportFragmentManager();
Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);

if (fragment == null) {
    fragment = new CrimeFragment();
    fm.beginTransaction()
        .add(R.id.fragmentContainer, fragment)
        .commit();
}
```

Может показаться странным, что `FragmentManager` идентифицирует `CrimeFragment` по идентификатору ресурса `FrameLayout`. Однако идентификация UI-фрагмента по идентификатору ресурса его контейнерного представления встроена в механизм работы `FragmentManager`. Если вы добавляете в активность несколько фрагментов, то обычно для каждого фрагмента создается отдельный контейнер со своим идентификатором.

Теперь мы можем кратко описать код, добавленный в листинг 7.15, от начала до конца.

Сначала у `FragmentManager` запрашивается фрагмент с идентификатором контейнерного представления `R.id.fragmentContainer`. Если этот фрагмент уже находится в списке, `FragmentManager` возвращает его.

Почему фрагмент может уже находиться в списке? Вызов `CrimeActivity.onCreate(...)` может быть выполнен в ответ на *воссоздание* объекта `CrimeActivity` после его уничтожения из-за поворота устройства или освобождения памяти. При уничтожении активности ее экземпляр `FragmentManager` сохраняет список фрагментов. При воссоздании активности новый экземпляр `FragmentManager` загружает список и воссоздает хранящиеся в нем фрагменты, чтобы все работало как прежде.

С другой стороны, если фрагменты с заданным идентификатором контейнерного представления отсутствуют, значение `fragment` равно `null`. В этом случае мы создаем новый экземпляр `CrimeFragment` и новую транзакцию, которая добавляет фрагмент в список.

Теперь `CrimeActivity` является хостом для `CrimeFragment`. Чтобы убедиться в этом, запустите приложение `CriminalIntent`. На экране отображается представление, определенное в файле `fragment_crime.xml` (рис. 7.18).

Возможно, один виджет на экране выглядит не таким уж большим достижением для всей работы, проделанной в этой главе. Однако мы заложили прочный фундамент для более серьезных задач, которые нам придется решать в приложении `CriminalIntent` в последующих главах.

FragmentManager и жизненный цикл фрагмента

После знакомства с `FragmentManager` стоит еще раз вернуться к жизненному циклу фрагмента (рис. 7.19).

Объект `FragmentManager` активности отвечает за вызов методов жизненного цикла фрагментов в списке. Методы `onAttach(Activity)`,

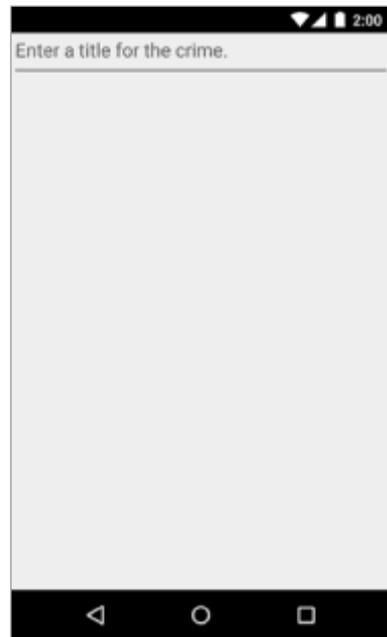


Рис. 7.18. `CrimeActivity` является хостом представления `CrimeFragment`

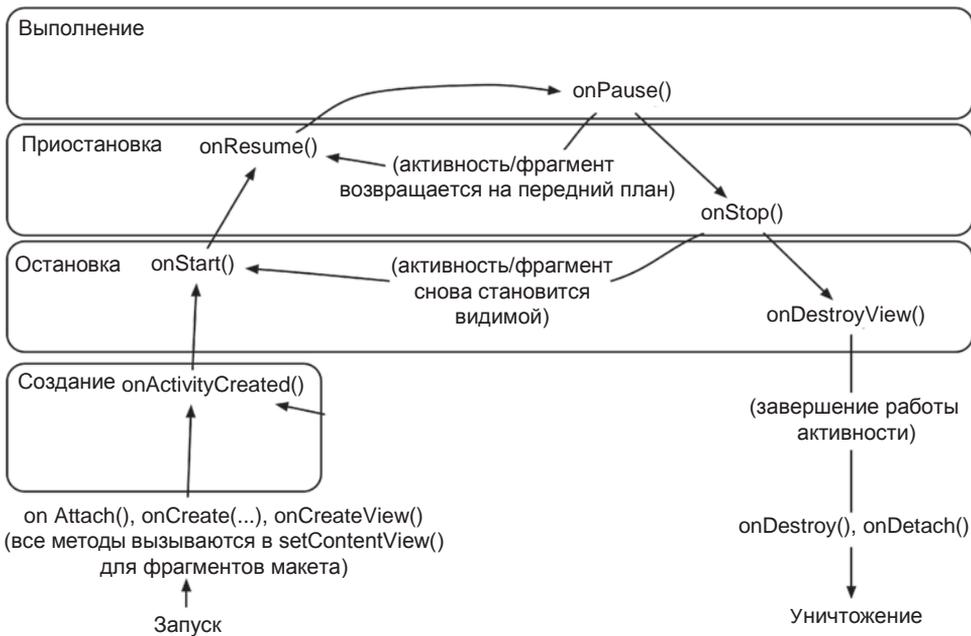


Рис. 7.19. Жизненный цикл фрагмента (повторно)

`onCreate(Bundle)` и `onCreateView(...)` вызываются при добавлении фрагмента в `FragmentManager`.

Метод `onActivityCreated(...)` вызывается после выполнения метода `onCreate(...)` активности-хоста. Мы добавляем `CrimeFragment` в `CrimeActivity.onCreate(...)`, так что этот метод будет вызываться после добавления фрагмента.

Что произойдет, если добавить фрагмент в то время, как активность уже находится в состоянии остановки, приостановки или выполнения? В этом случае `FragmentManager` немедленно проводит фрагмент через все действия, необходимые для его согласования с состоянием активности. Например, при добавлении фрагмента в активность, уже находящуюся в состоянии выполнения, фрагмент получит вызовы `onAttach(Activity)`, `onCreate(Bundle)`, `onCreateView(...)`, `onActivityCreated(Bundle)`, `onStart()` и затем `onResume()`.

После того как состояние фрагмента будет согласовано с состоянием активности, объект `FragmentManager` активности-хоста будет вызывать дальнейшие методы жизненного цикла приблизительно одновременно с получением соответствующих вызовов от ОС для синхронизации состояния фрагмента с состоянием активности.

Архитектура приложений с фрагментами

При разработке приложений с фрагментами очень важно правильно подойти к проектированию. Многие разработчики после первого знакомства с фрагментами пытаются применять их для всех компонентов приложения, подходящих

для повторного использования. Такой способ использования фрагментов ошибочен.

Фрагменты предназначены для инкапсуляции основных компонентов для повторного использования. В данном случае основные компоненты находятся на уровне всего экрана приложения. Если на экране одновременно выводится значительное количество фрагментов, ваш код замусоривается транзакциями фрагментов и неочевидными обязанностями. С точки зрения архитектуры существует другой, более правильный способ организации повторного использования вторичных компонентов — выделение их в специализированное представление (класс, субклассирующий View или один из его субклассов).

Пользуйтесь фрагментами осмотрительно. На практике не рекомендуется отображать более двух или трех фрагментов одновременно (рис. 7.20).



Рис. 7.20. Не гонитесь за количеством

Почему все наши активности используют фрагменты

С этого момента фрагменты будут использоваться во всех приложениях этой книги — даже самых простых. На первый взгляд такое решение кажется чрезмерным: многие примеры, которые вам встретятся в следующих главах, могут быть записаны без фрагментов. Для создания пользовательских интерфейсов и управления ими можно обойтись активностями; возможно, это даже уменьшит объем кода.

Тем не менее мы полагаем, что вам стоит поскорее привыкнуть к паттерну, который наверняка пригодится вам в реальной работе.

Кто-то скажет, что лучше сначала написать простое приложение без фрагментов, а потом добавить их, когда потребуется (и если потребуется). Эта идея заложена в основу методологии экстремального программирования YAGNI. Сокращение YAGNI означает «You Aren't Gonna Need It» («Вам это не понадобится»); этот принцип убеждает вас не писать код, который, по вашему мнению, *может* понадобиться позже. Почему? Потому что YAGNI. Возникает соблазн сказать YAGNI фрагментам.

К сожалению, добавление фрагментов в будущем может превратиться в мину замедленного действия. Превратить активность в активность хоста UI-фрагмента несложно, но при этом возникает множество неприятных ловушек. Если одними интерфейсами будут управлять активности, а другими — фрагменты, это только усугубит ситуацию, потому что вам придется отслеживать эти бессмысленные различия. Гораздо проще с самого начала написать код с использованием фрагментов и не возиться с его последующей переработкой или же запоминать, какой стиль контроллера используется в каждой части вашего приложения.

Итак, в том, что касается фрагментов, мы применяем другой принцип: AUF, или «Always Use Fragments» («Всегда используйте фрагменты»). Выбирая между использованием фрагмента или активности, вы потратите немало нервных клеток, а это просто не стоит того. AUF!

Для любознательных: почему фрагменты из библиотеки поддержки лучше

В этой книге мы отдаем предпочтение реализации фрагментов из библиотеки поддержки перед реализацией, встроенной в ОС Android, — на первый взгляд такое решение кажется неожиданным. В конце концов, реализация фрагментов из библиотеки поддержки изначально создавалась для того, чтобы разработчики могли использовать фрагменты в старых версиях Android, не поддерживающих API. Сегодня большинство разработчиков может работать с версиями Android, включающими поддержку фрагментов.

И все же мы предпочитаем фрагменты из библиотеки поддержки. Почему? Потому что вы можете обновить версию библиотеки поддержки в приложении и в любой момент отправить новую версию своего приложения. Новые версии библиотеки поддержки выходят несколько раз в год. Когда в API фрагментов добавляются новые возможности, они также включаются в API фрагментов в библиотеке поддержки вместе со всеми доступными исправлениями ошибок. Чтобы использовать новые возможности, достаточно обновить версию библиотеки поддержки в новом приложении.

Например, официальная поддержка вложенных фрагментов (хостинг фрагментов во фрагментах) добавилась в Android 4.2. Если вы используете фрагменты из ОС Android и поддерживаете Android 4.0 и выше, вам не удастся использовать этот

API на всех устройствах, поддерживаемых приложением. При использовании библиотеки поддержки вы можете обновить версию библиотеки в своем приложении и создавать вложенные фрагменты, пока не кончится свободная память на устройстве.

У фрагментов из библиотеки поддержки нет сколько-нибудь существенных недостатков. Реализации фрагментов в ОС и библиотеке поддержки почти идентичны. Единственный заметный недостаток — необходимость включения в проект библиотеки поддержки. Однако в текущей версии ее размер меньше мегабайта, причем вполне возможно, что вы все равно будете использовать библиотеку поддержки ради других возможностей.

В этой книге и в разработке своих приложений мы руководствуемся практически теми же соображениями. Библиотека поддержки лучше.

Для любознательных: использование встроенной реализации фрагментов

Если вы не боитесь трудностей и не верите приведенному выше совету, вы можете использовать реализацию фрагментов, встроенную в ОС Android.

Чтобы использовать классы фрагментов стандартной библиотеки, необходимо внести в проект три изменения:

1. Субклассируйте класс `Activity` стандартной библиотеки (`android.app.Activity`) вместо `FragmentActivity`. В API уровня 11 и выше активности содержат готовую поддержку фрагментов.
1. Субклассируйте `android.app.Fragment` вместо `android.support.v4.app.Fragment`.
1. Чтобы получить объект `FragmentManager`, используйте вызов `FragmentManager()` вместо `getSupportFragmentManager()`.

8

Макеты и виджеты

В этой главе мы поближе познакомимся с макетами и виджетами, а также включим хранение даты и статуса в `CriminalIntent`.

Обновление `Crime`

Откройте файл `Crime.java` и добавьте два новых поля. Поле `Date` представляет дату преступления, а поле типа `boolean` — признак того, было ли преступление раскрыто.

Листинг 8.1. Добавление полей в класс `Crime` (`Crime.java`)

```
public class Crime {
    private UUID mId;
    private String mTitle;
    private Date mDate;
    private boolean mSolved;

    public Crime() {
        mId = UUID.randomUUID();
        mDate = new Date();
    }
    ...
}
```

Возможно, `Android Studio` найдет два класса с именем `Date`. Используйте комбинацию клавиш `Option+Return` (или `Alt+Enter`), чтобы импортировать класс вручную. Когда вам будет предложено выбрать импортируемую версию класса `Date`, выберите версию `java.util.Date`.

Инициализация переменной `Date` конструктором `Date` по умолчанию присваивает `mDate` текущую дату.

Затем сгенерируйте `get-` и `set-`методы для своих новых полей (щелкните правой кнопкой мыши на файле и выберите команду `Generate... ▶ Getter and Setter`).

Листинг 8.2. Сгенерированные get- и set-методы (Crime.java)

```
public class Crime {
    ...
    public void setTitle(String title) {
        mTitle = title;
    }

    public Date getDate() {
        return mDate;
    }
    public void setDate(Date date) {
        mDate = date;
    }
    public boolean isSolved() {
        return mSolved;
    }
    public void setSolved(boolean solved) {
        mSolved = solved;
    }
}
```

Наши следующие действия — обновление макета в `fragment_crime.xml` новыми виджетами и их связывание с виджетами в `CrimeFragment.java`.

Обновление макета

Вот как будет выглядеть представление `CrimeFragment` к концу этой главы.

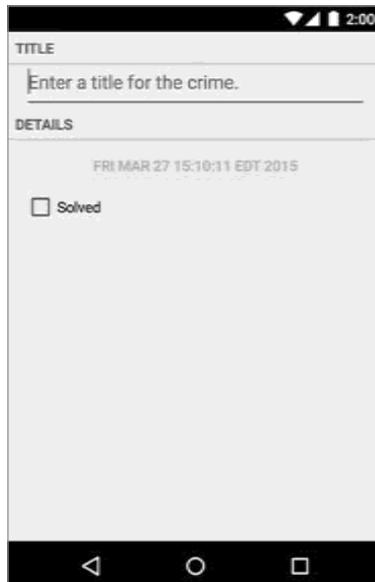


Рис. 8.1. CriminalIntent, эпизод 2

Чтобы добраться до этого экрана, мы добавим в макет CrimeFragment четыре виджета: два виджета TextView, Button и CheckBox.

Откройте файл fragment_crime.xml и внесите изменения, представленные в листинге 8.3. Возможно, вы получите ошибки отсутствия строковых ресурсов — вскоре мы их создадим.

Листинг 8.3. Добавление новых виджетов (fragment_crime.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="wrap_content"
    android:orientation="vertical"
    >
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_title_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <EditText android:id="@+id/crime_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:hint="@string/crime_title_hint"
    />
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/crime_details_label"
        style="?android:listSeparatorTextViewStyle"
    />
    <Button android:id="@+id/crime_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
    />
    <CheckBox android:id="@+id/crime_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/crime_solved_label"
    />
</LinearLayout>
```

Обратите внимание: для виджета Button мы не указали атрибут android:text. Кнопка будет выводить дату, хранящуюся в Crime, а ее текст будет задаваться в коде.

Почему дата выводится на Button? Это заготовка на будущее. Сейчас в качестве даты преступления по умолчанию используется текущая дата, и ее нельзя изменить. В главе 12 мы изменим кнопку так, что при ее нажатии будет вызываться виджет DatePicker, при помощи которого пользователь сможет выбрать другую дату.

У макета имеются другие особенности, заслуживающие упоминания, например атрибут style и атрибуты margin. Но сначала мы заставим CriminallIntent работать с новыми виджетами.

Вернитесь к файлу res/values/strings.xml и добавьте необходимые строковые ресурсы.

Листинг 8.4. Добавление строковых ресурсов (strings.xml)

```
<resources>
  <string name="app_name">CriminallIntent</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
  <string name="title_activity_crime">CrimeActivity</string>
  <string name="crime_title_hint">Enter a title for this crime.</string>
  <string name="crime_title_label">Title</string>
  <string name="crime_details_label">Details</string>
  <string name="crime_solved_label">Solved?</string>
</resources>
```

Сохраните файлы и проверьте возможные опечатки.

Подключение виджетов

Виджет CheckBox должен показывать, было ли преступление раскрыто. Изменение состояния CheckBox также должно приводить к обновлению поля mSolved класса Crime.

От Button пока что требуется только вывод даты из поля mDate объекта Crime.

В файле CrimeFragment.java добавьте две переменные экземпляра.

Листинг 8.5. Добавление переменных экземпляра для виджетов (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private Crime mCrime;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...
    }
}
```

Затем в onCreateView(...) получите ссылку на новую кнопку, задайте в тексте кнопки дату преступления и заблокируйте ее.

Листинг 8.6. Назначение текста Button (CrimeFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, parent, false);
    ...

    mTextField.addTextChangedListener(new TextWatcher() {
        ...
    });

    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setText(mCrime.getDate().toString());
    mDateButton.setEnabled(false);

    return v;
}

```

Блокировка кнопки гарантирует, что она не будет реагировать на нажатия. Также при этом изменяется оформление кнопки, чтобы сообщить пользователю о заблокированном состоянии. В главе 12 блокировка кнопки будет снята при назначении слушателя.

Теперь можно заняться CheckBox: мы получаем ссылку и назначаем слушателя, который будет обновлять поле mSolved объекта Crime.

Листинг 8.7. Назначение слушателя для изменений CheckBox (CrimeFragment.java)

```

...
mDateButton = (Button)v.findViewById(R.id.crime_date);
mDateButton.setText(mCrime.getDate().toString());
mDateButton.setEnabled(false);

mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView, boolean
        isChecked) {
        // Назначение флага раскрытия преступления
        mCrime.setSolved(isChecked);
    }
});

return v;
}

```

При создании OnCheckedChangeListener вам будет предложено выбрать один из двух вариантов импортирования. Выберите версию android.widget.CompoundButton.

Запустите CrimialIntent. Переключите состояние CheckBox и насладитесь видом заблокированной кнопки, на которой отображается текущая дата.

Подробнее об атрибутах макетов XML

Вернемся к некоторым атрибутам, добавленным в файле `fragment_crime.xml`, и ответим на некоторые насущные вопросы по поводу виджетов и атрибутов.

Стили, темы и атрибуты тем

Стиль (style) представляет собой ресурс XML, который содержит атрибуты, описывающие внешний вид и поведение виджета. Например, ниже приведен ресурс стиля, который настраивает виджет на использование увеличенного размера текста.

```
<style name="BigTextStyle">
  <item name="android:textSize">20sp</item>
  <item name="android:layout_margin">3dp</item>
</style>
```

Вы можете создавать собственные стили (мы займемся этим в главе 20). Они добавляются в файл стилей из каталога `res/values/`, а ссылки на них в макетах выглядят так: `@style/my_own_style`.

Еще раз взгляните на виджеты `TextView` из файла `fragment_crime.xml`; каждый виджет имеет атрибут `style`, который ссылается на стиль, созданный Android. С этим конкретным стилем виджеты `TextView` выглядят как разделители списка, а берется он из *темы* приложения. *Тема* (theme) представляет собой набор стилей. Со структурной точки зрения тема сама является ресурсом стиля, атрибуты которого ссылаются на другие ресурсы стилей.

Android предоставляет платформенные темы, которые могут использоваться вашими приложениями. При создании `CriminalIntent` мастер назначает тему приложения, которая определяется тегом `application` в манифесте.

Стиль из темы приложения можно применить к виджету при помощи *ссылки на атрибут темы* (theme attribute reference). Именно это мы делаем в файле `fragment_crime.xml`, используя значение `?android:listSeparatorTextViewStyle`.

Ссылка на атрибут темы приказывает менеджеру ресурсов Android: «Перейди к теме приложения и найди в ней атрибут с именем `listSeparatorTextViewStyle`. Этот атрибут указывает на другой ресурс стиля. Помести значение этого ресурса сюда».

Каждая тема Android включает атрибут с именем `listSeparatorTextViewStyle`, но его определение зависит от оформления конкретной темы. Использование ссылки на атрибут темы гарантирует, что оформление виджетов `TextView` будет соответствовать оформлению вашего приложения.

О том, как работают стили и темы, более подробно рассказано в главе 20.

Плотность пикселей, dp и sp

В файле `fragment_crime.xml` значение атрибута `margin` задается в единицах `dp`. Вы уже видели эти единицы в макетах; пришло время узнать, что они собой представляют.

Иногда значения атрибутов представления задаются в конкретных размерах (как правило, в пикселах, но иногда в пунктах, миллиметрах или дюймах). Чаще всего этот способ используется для атрибутов размера текста, полей и отступов. Размер текста равен высоте текста в пикселах на экране устройства. Поля задают расстояния между представлениями, а отступы задают расстояние между внешней границей представления и его содержимым.

Как было показано в разделе «Добавление значка» в главе 2, Android автоматически масштабирует изображения для разных плотностей пикселей экрана, используя имена каталогов, уточненные спецификаторами плотности (например, `drawable-hdpi`). Но что произойдет, если ваши изображения масштабируются, а поля — нет? Или если пользователь выберет размер текста больше стандартного?

Для решения таких проблем в Android поддерживаются единицы, не зависящие от плотности пикселей; используя их, можно получить одинаковые размеры на экранах с разными плотностями. Android преобразует эти единицы в пиксели во время выполнения, так что вам не придется самостоятельно заниматься сложными вычислениями (рис. 8.2).

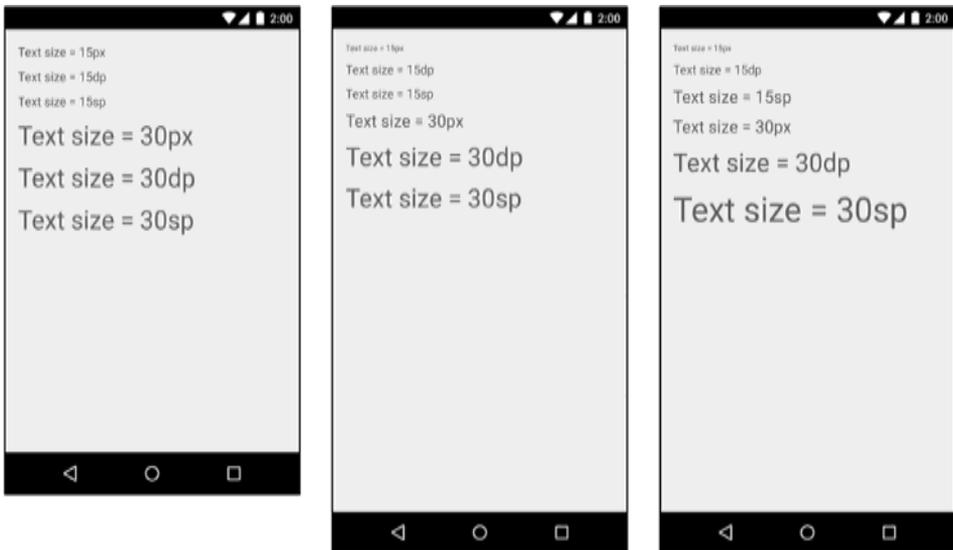


Рис. 8.2. Использование единиц устройства на TextView (слева: MDPI; в середине: HDPI; справа: HDPI с большим текстом)

- ① dp (или dip) — сокращение от «density-independent pixel» (пиксели, не зависящие от плотности); произносится «дип». Обычно эти единицы используются для полей, отступов и всего остального, для чего обычно задаются размеры в пикселах. На экранах с более высокой плотностью единицы dp разворачиваются в большее количество экранных пикселей. Одна единица dp всегда равна 1/160 дюйма на экране устройства. Размер будет одинаковым независимо от плотности пикселей.

- ① `sp` — сокращение от «scale-independent pixel» (пиксели, не зависящие от масштаба). Эти единицы, не зависящие от плотности пикселей устройства, также учитывают выбранный пользователем размер шрифта. Единицы `sp` почти всегда используются для назначения размера текста.
- ① `pt`, `mm`, `in` — масштабируемые единицы (как и `dp`), позволяющие задавать размеры интерфейсных элементов в пунктах (1/72 дюйма), миллиметрах или дюймах. Тем не менее мы не рекомендуем их использовать: не все устройства правильно настроены для правильного масштабирования этих устройств.

На практике и в этой книге почти исключительно используются только единицы `dp` и `sp`. Android преобразует эти значения в пиксели во время выполнения.

Рекомендации по проектированию интерфейсов Android

Для полей в нашем примере в листинге 8.3 используется значение `16dp`. Оно следует рекомендации по проектированию материальных интерфейсов Android. Полный список рекомендаций находится по адресу <http://developer.android.com/design/index.html>.

Современные приложения Android должны соответствовать этим рекомендациям, насколько это возможно. Рекомендации в значительной мере зависят от новой функциональности Android SDK, которая не всегда доступна или легко реализуема на старых устройствах. Некоторые рекомендации могут соблюдаться с использованием библиотеки `AppCompat`, о которой можно прочитать в главе 13.

Параметры макета

Вероятно, вы уже заметили, что некоторые имена атрибутов начинаются с `layout_` (`android:layout_marginLeft`), а у других атрибутов этого префикса нет (`android:text`).

Атрибуты, имена которых *не* начинаются с `layout_`, являются рекомендациями для виджетов. При заполнении виджет вызывает метод для настройки своей конфигурации на основании этих атрибутов и их значений.

Если имя атрибута начинается с `layout_`, то этот атрибут является директивой для родителя этого виджета. Такие атрибуты, называемые *параметрами макета*, сообщают родительскому макету, как следует расположить дочерний элемент внутри родителя.

Даже если объект макета (например, `LinearLayout`) является корневым элементом, он все равно остается виджетом, имеющим родителя, и у него есть параметры макета. Определяя элемент `LinearLayout` в файле `fragment_crime.xml`, мы задали атрибуты `android:layout_width` и `android:layout_height`. Эти атрибуты будут использоваться родительским макетом `LinearLayout` при заполнении. В данном случае параметры макета `LinearLayout` будут использоваться элементом `FrameLayout` в представлении содержимого `CrimeActivity`.

Поля и отступы

В файле `fragment_crime.xml` виджетам назначаются атрибуты `margin` и `padding`. Начинаящие разработчики иногда путают эти атрибуты, определяющие соответственно *поля* и *отступы*. Теперь, когда вы понимаете, что такое параметр макета, будет проще объяснить, чем они различаются. Атрибуты `margin` являются параметрами макета; они определяют расстояние между виджетами. Так как виджет располагает информацией только о самом себе, за соблюдение полей отвечает родитель виджета.

Напротив, отступ не является параметром макета. Атрибут `android:padding` сообщает виджету, с каким превышением размера содержимого он должен прорисовывать себя. Допустим, вы хотите заметно увеличить размеры кнопки даты без изменения размера текста. Добавьте в `Button` следующий атрибут, сохраните макет и запустите приложение заново.

Листинг 8.8. Назначение отступов

```
<Button android:id="@+id/crime_date"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_marginLeft="16dp"
  android:layout_marginRight="16dp"
  android:padding="80dp"
/>
```

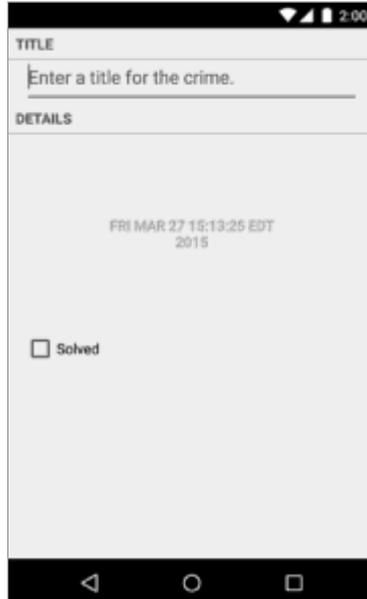


Рис. 8.3. Для любителей больших кнопок

Не забудьте удалить этот атрибут, прежде чем продолжать работу.

Использование графического конструктора

До настоящего момента мы создавали макеты, вводя разметку XML. В этом разделе мы используем графический конструктор для построения альтернативного альбомного макета CrimeFragment.

Большинство встроенных классов макетов, таких как `LinearLayout`, — автоматически изменяют размеры себя и своих потомков при поворотах. Однако в некоторых случаях изменение размеров по умолчанию недостаточно эффективно использует свободное пространство.

Запустите приложение `CriminalIntent` и поверните устройство, чтобы увидеть макет `CrimeFragment` в альбомной ориентации (рис. 8.4).



Рис. 8.4. CrimeFragment в альбомной ориентации

Кнопка даты становится слишком длинной; было бы лучше, если бы в альбомной ориентации кнопка и флажок располагались рядом друг с другом.

Чтобы внести эти изменения, переключитесь в графический конструктор. Откройте файл `fragment_crime.xml` и выберите вкладку `Design` в нижней части панели.

В центре графического конструктора располагается уже знакомая панель предварительного просмотра. Слева располагается *палитра*. На ней размещаются практически все виджеты, которые только можно себе представить, упорядоченные по категориям (рис. 8.5).

Справа находится *дерево компонентов*, которое показывает, как организованы виджеты в макете.

Под деревом компонентов располагается *панель свойств*. На ней можно просматривать и редактировать атрибуты виджета, выделенного в дереве компонентов.

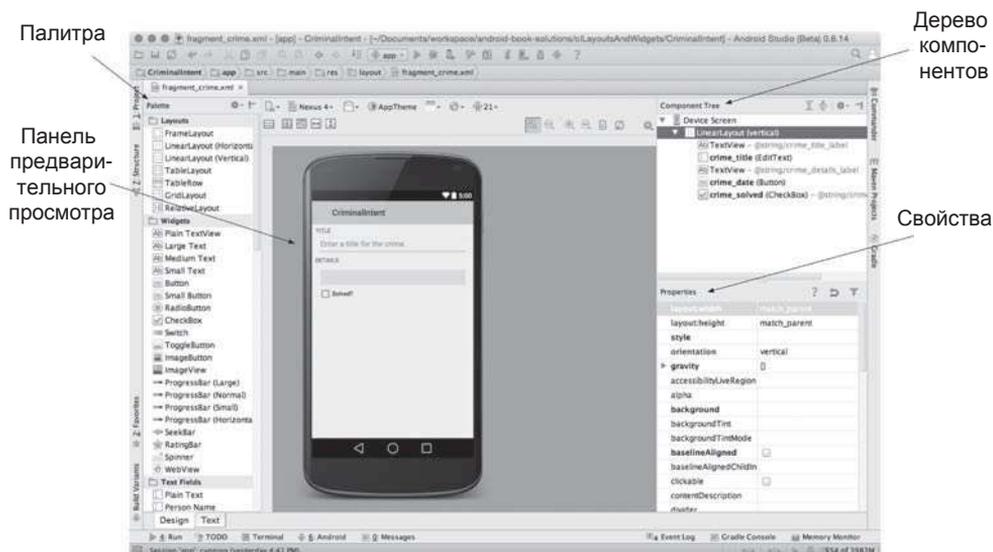


Рис. 8.5. Графический конструктор

Создание альбомного макета

Графический конструктор может сгенерировать альбомную версию файла макета за вас. Найдите кнопку с изображением листа бумаги и эмблемой Android в правом нижнем углу (рис. 8.6). Щелкните на этой кнопке и выберите команду **Create Landscape Variation**.

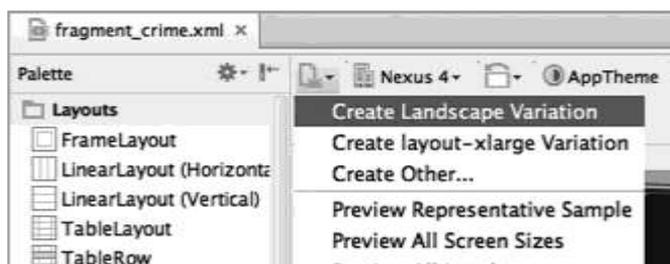


Рис. 8.6. Создание альтернативного макета в графическом конструкторе

На экране появляется новый макет. При этом автоматически создается каталог `res/layout-land`, а существующий файл макета `fragment_crime.xml` копируется в новый каталог.

Какие же изменения следует внести в альбомный макет? Взгляните на рис. 8.7.

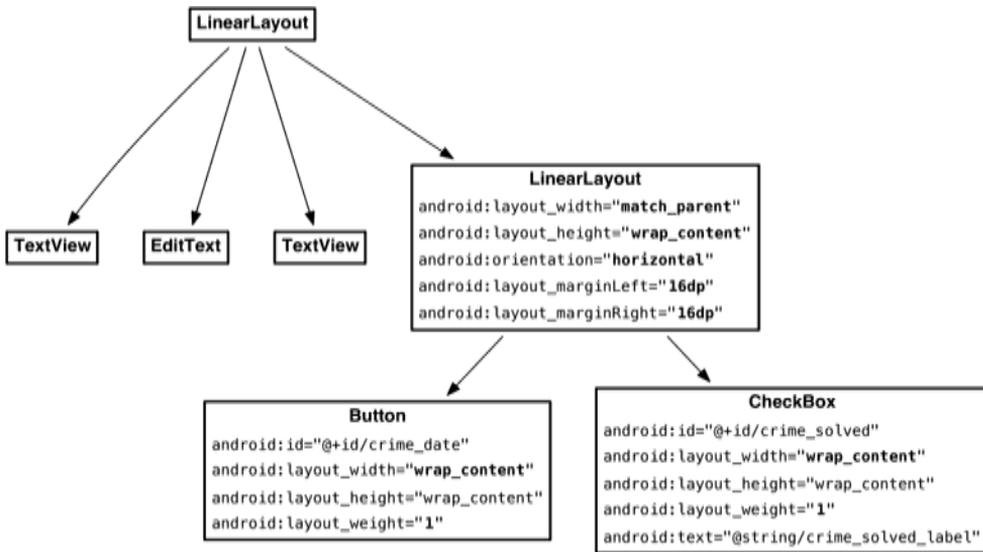


Рис. 8.7. Альбомный макет для CrimeFragment

Изменения можно разделить на четыре группы:

- ① Добавление виджета LinearLayout в макет.
- ① Редактирование атрибутов LinearLayout.
- ① Назначение виджетов Button и CheckBox потомками LinearLayout.
- ① Обновление параметров макета Button и CheckBox.

Добавление нового виджета

Чтобы добавить виджет, можно выделить его в палитре и перетащить на дерево компонентов. Щелкните на категории Layouts в палитре, если она еще не раскрыта. Выберите пункт LinearLayout (Horizontal) и перетащите его на дерево компонентов. Отпустите перетаскиваемый объект над кнопкой даты. Убедитесь в том, что новый виджет LinearLayout является потомком корневого элемента LinearLayout, как показано на рис. 8.8.

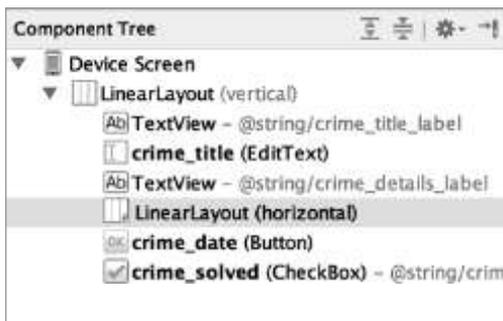


Рис. 8.8. Добавление LinearLayout в fragment_crime.xml

Виджеты также можно добавлять перетаскиванием из палитры в область предварительного просмотра. Однако виджеты Layout часто пусты или закрыты другими представлениями, поэтому может быть трудно понять, где

следует разместить виджет в области предварительного просмотра для получения нужной иерархии. Перетаскивание на дерево компонентов существенно упрощает выполнение этой операции.

Редактирование атрибутов в свойствах

Выберите новый виджет `LinearLayout` в дереве компонентов, чтобы отобразить его атрибуты на панели свойств. Найдите атрибуты `layout:width` и `layout:height`.

Присвойте атрибуту `layout:width` значение `match_parent`, а атрибуту `layout:height` — значение `wrap_content`, как показано на рис. 8.9. Теперь виджет `LinearLayout` заполняет всю доступную ширину и занимает столько места по вертикали, сколько необходимо для отображения `CheckBox` и `Button`.

Мы хотим, чтобы поля нового виджета `LinearLayout` совпадали с полями других виджетов. Откройте атрибут `layout:margin`, выделите поле рядом с `Left` и введите значение `16dp`. Сделайте то же самое для правого (`Right`) поля (рис. 8.10).

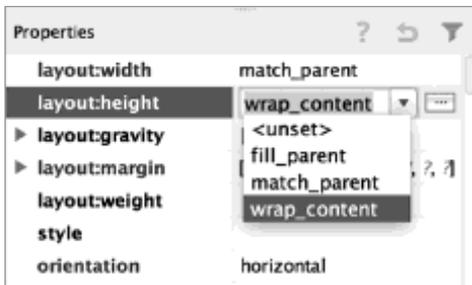


Рис. 8.9. Изменение ширины и высоты `LinearLayout`

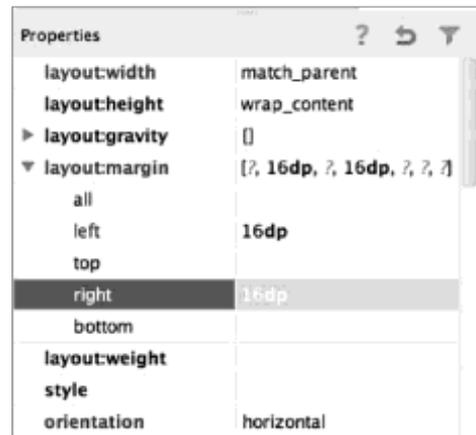


Рис. 8.10. Назначение полей в режиме свойств

Сохраните файл макета и переключитесь на разметку XML при помощи вкладки `text` в нижней части области предварительного просмотра. Вы увидите только что добавленный элемент `LinearLayout` с атрибутами полей.

Реорганизация виджетов в дереве компонентов

Следующий шаг — назначение виджетов `Button` и `CheckBox` потомками нового виджета `LinearLayout`. Вернитесь к графическому конструктору, выберите виджет `Button` и перетащите его на `LinearLayout`.

В дереве компонентов отражается тот факт, что `Button` теперь является потомком нового виджета `LinearLayout` (рис. 8.11). Прделайте то же самое с `CheckBox`.



Рис. 8.11. Button и CheckBox становятся потомками нового виджета LinearLayout

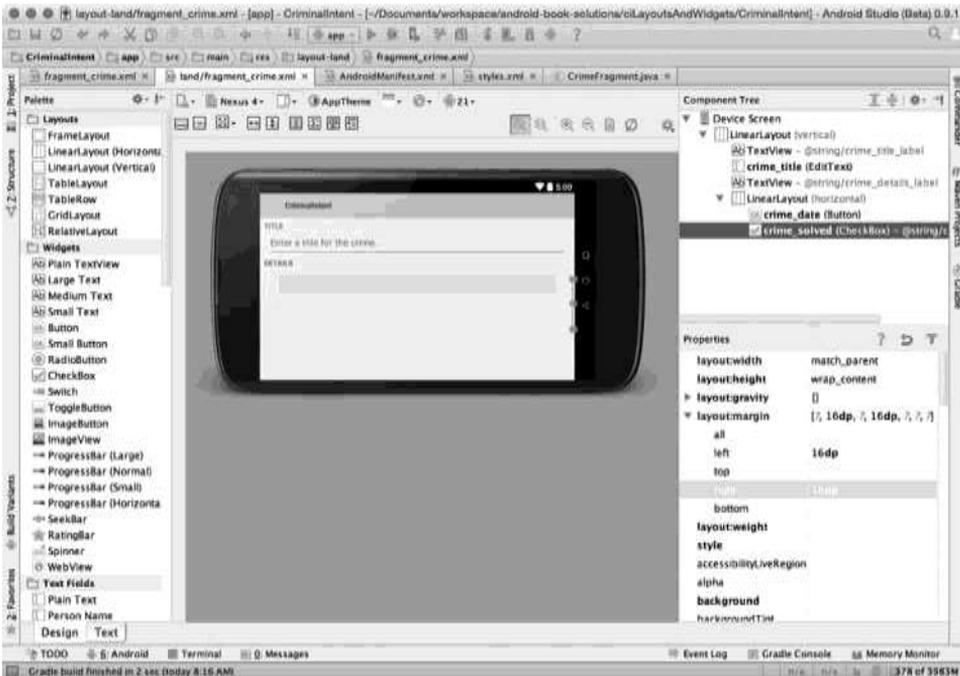


Рис. 8.12. Виджет Button, который был определен первым, скрывает CheckBox

Если потомки размещаются не в том порядке, их можно переупорядочить посредством перетаскивания. Также в дереве компонентов можно удалять виджеты из макета, но будьте осторожны: удаление виджета приводит к удалению его потомков.

В области предварительного просмотра виджет CheckBox не виден — его скрывает Button. Виджет LinearLayout проверил ширину (`match_parent`) своего первого потомка (Button) и выделил ему все пространство, ничего не оставив CheckBox (рис. 8.12).

Чтобы восстановить равноправие потомков `LinearLayout`, мы изменим параметры макета потомков.

Обновление параметров макета потомков

Сначала выделите кнопку даты в дереве компонентов. На панели свойств щелкните на текущем значении `layout:width` и замените его значением `wrap_content`.

Удалите оба значения `16dp` полей кнопки. Теперь, когда кнопка находится внутри `LinearLayout`, поля ей не нужны.

Наконец, найдите поле `layout:weight` в разделе `Layout Parameters` и задайте ему значение `1`. Это поле соответствует атрибуту `android:layout_weight` на рис. 8.7.

Выберите виджет `CheckBox` в дереве компонентов и внесите те же изменения: атрибут `layout_width` должен содержать `wrap_content`, атрибуты полей должны быть пустыми, а атрибут `Weight` должен быть равен `1`.

В области предварительного просмотра убедитесь в том, что оба виджета теперь видны. Сохраните файлы и вернитесь к XML, чтобы подтвердить изменения. В листинге 8.9 приведена соответствующая разметка XML.

Листинг 8.9. Разметка XML макета, созданного в графическом конструкторе (`layout-land/fragment_crime.xml`)

```
...
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/crime_details_label"
    style="?android:listSeparatorTextViewStyle"
/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="16dp"
    android:layout_marginRight="16dp" >
    <Button
        android:id="@+id/crime_date"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
    <CheckBox
        android:id="@+id/crime_solved"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/crime_solved_label" />
    </LinearLayout>
</LinearLayout>
```

Запустите `CriminalIntent`, поверните устройство в альбомную ориентацию и похвалите себя за успешную оптимизацию макета для новой конфигурации устройства.

Как работает android:layout_weight

Атрибут веса `android:layout_weight` сообщает виджету `LinearLayout`, как он должен распределить потомков по размеру контейнера. Обоим виджетам заданы одинаковые значения ширины, но это не гарантирует, что они будут иметь одинаковую ширину на экране. Для определения ширин дочерних представлений `LinearLayout` использует комбинацию параметров `layout_width` и `layout_weight`.

`LinearLayout` вычисляет ширину представления в два прохода. На первом проходе `LinearLayout` проверяет значение `layout_width` (или `layout_height` для вертикальной ориентации). Значение `layout_width` как для `Button`, так и для `CheckBox` теперь равно `wrap_content`, так что каждому представлению выделяется место, достаточное только для его прорисовки (рис. 8.13).

(По области предварительного просмотра трудно понять, как работает система весов, потому что содержимое кнопки в макет не входит. На следующих рисунках показано, как будет выглядеть `LinearLayout`, если кнопка уже имеет содержимое.)



Рис. 8.13. Проход 1: распределение пространства на основании `layout_width`

На следующем проходе `LinearLayout` распределяет дополнительное пространство на основании значений `layout_weight` (рис. 8.14).

В нашем макете `Button` и `CheckBox` имеют одинаковые значения `layout_weight`, поэтому дополнительное пространство распределяется в соотношении 50/50. Если задать весовой коэффициент `Button` равным 2, то ей будет выделено 2/3 дополнительного пространства, а `CheckBox` достанется всего 1/3 (рис. 8.15).



Рис. 8.14. Проход 2: распределение дополнительного пространства на основании `layout_weight`



Рис. 8.15. Неравномерное распределение дополнительного пространства с пропорцией `layout_weight` 2:1

В качестве весового коэффициента может использоваться любое вещественное число. Программисты используют разные системы обозначений весов. В файле `fragment_crime.xml` используется система «рецепт коктейля». Также часто применяются наборы весов, сумма которых составляет 1.0 или 100; в этом случае вес кнопки в приведенном примере составит 0.66 или 66 соответственно.

Что, если вы хотите, чтобы виджет `LinearLayout` выделял для каждого представления ровно 50 % своей ширины? Просто пропустите первый проход, задав атрибуту `layout_width` каждого виджета значение `0dp` вместо `wrap_content`. В этом случае `LinearLayout` принимает решения только на основании значений `layout_weight` (рис. 8.16).



Рис. 8.16. При `layout_width="0dp"` учитываются только значения `layout_weight`

Графический конструктор макетов

Графический конструктор макетов удобен, и Android совершенствует его с каждым выпуском Android Studio. Однако порой он работает медленно и ненадежно, и тогда проще использовать прямой ввод разметки XML. Вы можете переключаться между внесением изменений в графическом конструкторе и в XML.

Не стесняйтесь использовать графический конструктор для создания макетов из этой книги. В дальнейшем, когда потребуются создать макет, мы будем приводить диаграмму наподобие рис. 8.7. Вы сами сможете решить, как создать ее: в виде разметки XML, в графическом конструкторе или сочетанием этих двух способов.

Идентификаторы виджетов и множественные макеты

Два макета, созданные для `CriminalIntent`, отличаются незначительно, но в некоторых ситуациях возможны более серьезные расхождения. В таких случаях перед обращением к виджету в коде необходимо убедиться в том, что он действительно существует.

Если виджет присутствует в одном макете и отсутствует в другом, то для проверки его наличия в текущей ориентации перед вызовом методов следует использовать проверку `null`:

```
Button landscapeOnlyButton = (Button)v.findViewById(R.  
                                id.landscapeOnlyButton);  
if (landscapeOnlyButton != null) {  
    // Операции  
}
```

Наконец, помните, что для того, чтобы ваш код мог найти виджет, последний должен иметь *одинаковые* значения атрибута `android:id` во всех макетах, в которых он присутствует.

Упражнение. Форматирование даты

Объект `Date` больше напоминает временную метку (`timestamp`), чем традиционную дату. При вызове `toString()` для `Date` вы получаете именно временную метку, которая отображается на кнопке. Временные метки хорошо подходят для отчетов, но на кнопке было бы лучше выводить дату в формате, более привычном для людей (например, «Jul 12, 2015»). Для этого можно воспользоваться экземпляром класса `android.text.format.DateFormat`. Хорошей отправной точкой в работе станет описание этого класса в документации Android.

Используйте методы класса `DateFormat` для формирования строки в стандартном формате или же подготовьте собственную форматную строку. Чтобы задача стала более творческой, попробуйте создать форматную строку для вывода дня недели («Tuesday, Jul 22, 2015»).

9

Вывод списков и ListFragment

Уровень модели CriminalIntent в настоящее время состоит из единственного экземпляра Crime. В этой главе мы обновим приложение CriminalIntent, чтобы оно поддерживало списки. В списке для каждого преступления будут отображаться краткое описание и дата, а также признак его раскрытия (рис. 9.1).

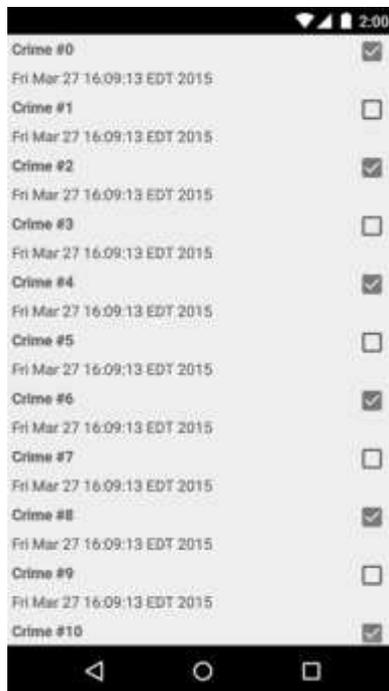


Рис. 9.1. Список преступлений

На рис. 9.2 показана общая структура приложения CriminalIntent для этой главы.

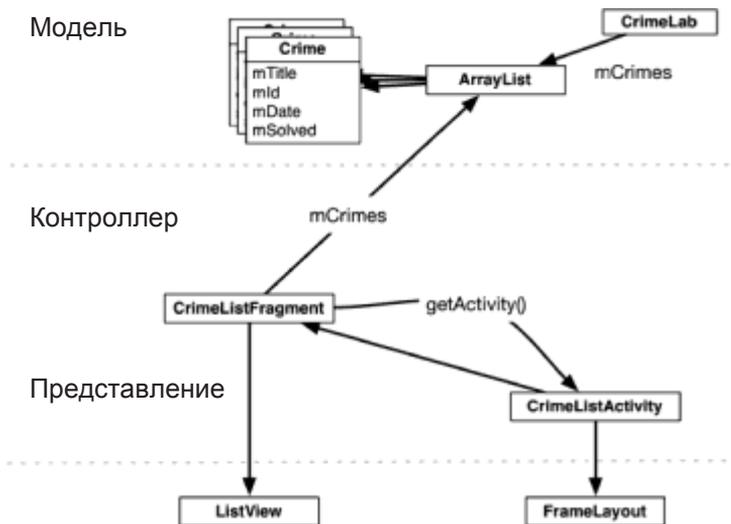


Рис. 9.2. Приложение CriminalIntent со списком

На уровне модели появляется новый объект `CrimeLab`, который представляет собой централизованное хранилище для объектов `Crime`.

Для отображения списка на уровне контроллера `CriminalIntent` появляется новая активность и новый фрагмент: `CrimeListActivity` и `CrimeListFragment`.

(Где находятся классы `CrimeActivity` и `CrimeFragment` на рис. 9.2? Они являются частью представления детализации, поэтому на рисунке их нет. В главе 10 мы свяжем части списка и детализации `CriminalIntent`.)

На рис. 9.2 также видны объекты представлений, связанные с `CrimeListActivity` и `CrimeListFragment`. Представление активности состоит из объекта `FrameLayout`, содержащего фрагмент. Представление фрагмента состоит из `RecyclerView`. Класс `RecyclerView` более подробно рассматривается позднее в этой главе.

Обновление уровня модели `CriminalIntent`

Прежде всего необходимо преобразовать уровень модели `CriminalIntent` из одного объекта `Crime` в список `List` объектов `Crime`.

Синглеты и централизованное хранение данных

Для хранения массива-списка преступлений будет использоваться *синглетный* (singleton) класс. Такие классы допускают создание только одного экземпляра.

Экземпляр синглетного класса существует до тех пор, пока приложение остается в памяти, так что при хранении списка в синглетном объекте данные остаются доступными, что бы ни происходило с активностями, фрагментами и их жизненными циклами. Будьте внимательны, поскольку синглетные классы будут уничтожаться, когда Android удаляет ваше приложение из памяти. Синглет `CrimeLab` не подходит для долгосрочного хранения данных, но позволяет приложению назначить одного владельца данных преступления и предоставляет возможность простой передачи этой информации между классами-контроллерами.

(За дополнительной информацией о синглетных классах обращайтесь к разделу «Для любознательных» в конце этой главы.)

Чтобы создать синглетный класс, следует создать класс с закрытым конструктором и методом `get()`. Если экземпляр уже существует, то `get()` просто возвращает его. Если экземпляр еще не существует, то `get()` вызывает конструктор для его создания.

Щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.criminalintent` и выберите команду `New ▶ Java Class`. Введите имя класса `CrimeLab` и щелкните на кнопке `Finish`.

В файле `CrimeLab.java` реализуйте `CrimeLab` как синглетный класс с закрытым конструктором и методом `get()`.

Листинг 9.1. Синглетный класс (`CrimeLab.java`)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    public static CrimeLab get(Context context) {
        if (sCrimeLab == null) {
            sCrimeLab = new CrimeLab(context);
        }
        return sCrimeLab;
    }
    private CrimeLab(Context context) {
    }
}
```

В реализации `CrimeLab` есть несколько моментов, заслуживающих внимания. Во-первых, обратите внимание на префикс `s` у переменной `sCrimeLab`. Мы используем это условное обозначение Android, чтобы показать, что переменная `sCrimeLab` является статической.

Также обратите внимание на закрытый конструктор `CrimeLab`. Другие классы не смогут создать экземпляр `CrimeLab` в обход метода `get()`.

Наконец, в методе `get()` конструктору `CrimeLab` передается параметр `Context`. Пока этот объект не используется, но мы вернемся к нему в главе 14.

Для начала предоставим `CrimeLab` несколько объектов `Crime` для хранения. В конструкторе `CrimeLab` создайте пустой список `List` объектов `Crime`. Также добавь-

те два метода: `getCrimes()` возвращает `List`, а `getCrime(UUID)` возвращает объект `Crime` с заданным идентификатором (листинг 9.2).

Листинг 9.2. Создание списка `List` объектов `Crime` (`CrimeLab.java`)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;

    public static CrimeLab get(Context context) {
        ...
    }

    private CrimeLab(Context context) {
        mCrimes = new ArrayList<>();
    }

    public List<Crime> getCrimes() {
        return mCrimes;
    }

    public Crime getCrime(UUID id) {
        for (Crime crime : mCrimes) {
            if (crime.getId().equals(id)) {
                return crime;
            }
        }
        return null;
    }
}
```

`List<E>` — интерфейс поддержки упорядоченного списка объектов заданного типа. Оно определяет методы получения, добавления и удаления элементов. Одна из распространенных реализаций `List` — `ArrayList` — использует для хранения элементов списка обычный массив Java.

Так как `mCrimes` содержит `ArrayList`, а `ArrayList` также является частным случаем `List`, и `ArrayList` и `List` являются действительными типами для `mCrimes`. В подобных ситуациях мы рекомендуем использовать в объявлении переменной тип интерфейса: `List`. В этом случае, если вам когда-либо понадобится перейти на другую реализацию `List`, например `LinkedList`, вы легко сможете это сделать.

В строке создания экземпляра `mCrimes` используется «ромбовидный» синтаксис `<>`, появившийся в Java 7. Эта сокращенная запись приказывает компилятору определить тип элементов, которые будут храниться в `List`, на основании обобщенного аргумента, переданного при объявлении переменной. В данном случае компилятор заключает, что `ArrayList` содержит объекты `Crime`, потому что в объявлении переменной `private List<Crime> mCrimes;` указан обобщенный аргумент `Crime`. (До Java 7 разработчикам приходилось использовать более длинную эквивалентную конструкцию `mCrimes = new ArrayList<Crime>();`.)

Со временем List будет содержать объекты Crime, созданные пользователем, которые будут сохраняться и загружаться повторно. А пока заполним массив 100 однообразных объектов Crime (листинг 9.3).

Листинг 9.3. Генерирование тестовых объектов (CrimeLab.java)

```
private CrimeLab(Context Context) {
    mContext = appContext;
    mCrimes = new ArrayList<>();
    for (int i = 0; i < 100; i++) {
        Crime crime = new Crime();
        crime.setTitle("Crime #" + i);
        crime.setSolved(i % 2 == 0); // Для каждого второго объекта
        mCrimes.add(crime);
    }
}
```

Теперь у нас имеется полностью загруженный уровень модели и 100 преступлений для вывода на экран.

Абстрактная активность для хостинга фрагмента

Вскоре мы создадим класс CrimeListActivity, предназначенный для выполнения функций хоста для CrimeListFragment. Но сначала будет создано представление для CrimeListActivity.

Обобщенный макет для хостинга фрагмента

Для CrimeListActivity можно просто воспользоваться макетом, определенным в файле activity_crime.xml (листинг 9.4). Этот макет определяет виджет FrameLayout как контейнерное представление для фрагмента, который затем указывается в коде активности.

Листинг 9.4. Файл activity_crime.xml уже содержит универсальную разметку

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
/>
```

Поскольку в файле activity_crime.xml не указан конкретный фрагмент, он может использоваться для любой активности, выполняющей функции хоста для одного фрагмента. Переименуем его в activity_fragment.xml, чтобы отразить этот факт.

В окне инструментов Project щелкните правой кнопкой мыши на файле `res/layout/activity_crime.xml`. (Будьте внимательны — щелкнуть нужно на `activity_crime.xml`, а не на `fragment_crime.xml`.)

Выберите в контекстном меню команду Refactor ▶ Rename.... Введите имя `activity_fragment.xml`. При переименовании ресурса ссылки на него обновляются автоматически.

Среда Android Studio должна автоматически обновить ссылки на новый файл `activity_fragment.xml`. Если вы получите сообщение об ошибке в `CrimeActivity.java`, вам придется вручную обновить ссылку `CrimeActivity`, как показано в листинге 9.5.

Листинг 9.5. Обновление файла макета для `CrimeActivity` (`CrimeActivity.java`)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_crime);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}
```

Абстрактный класс Activity

Для создания класса `CrimeListActivity` можно повторно использовать код `CrimeActivity`. Взгляните на код, написанный для `CrimeActivity` (листинг 9.5): он прост и практически универсален. Собственно, в нем есть всего одно не универсальное место: создание экземпляра `CrimeFragment` перед его добавлением в `FragmentManager`.

Листинг 9.6. Класс `CrimeActivity` почти универсален (`CrimeActivity.java`)

```
public class CrimeActivity extends FragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);

        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragmentContainer);
    }
}
```

```

        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragmentContainer, fragment)
                .commit();
        }
    }
}

```

Почти в каждой активности, которая будет создаваться в этой книге, будет присутствовать такой же код. Чтобы нам не приходилось вводить его снова и снова, мы выделим его в абстрактный класс.

Создайте новый класс с именем `SingleFragmentActivity` в пакете `CriminalIntent`. Сделайте его субклассом `FragmentActivity` и объявите абстрактным классом.

Листинг 9.7. Создание абстрактной активности (`SingleFragmentActivity.java`)

```

public abstract class SingleFragmentActivity extends FragmentActivity {
}

```

Теперь включите следующий фрагмент в `SingleFragmentActivity.java`. Не считая выделенных частей, он идентичен старому коду `CrimeActivity`.

Листинг 9.8. Добавление обобщенного суперкласса (`SingleFragmentActivity.java`)

```

public abstract class SingleFragmentActivity extends FragmentActivity {
    protected abstract Fragment createFragment();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);

        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
}

```

В этом коде представление активности заполняется по данным `activity_fragment.xml`. Затем мы ищем фрагмент в `FragmentManager` этого контейнера, создавая и добавляя его, если он не существует.

Код в листинге 9.8 отличается от кода `CrimeActivity` только абстрактным методом `createFragment()`, который используется для создания экземпляра фрагмента.

Субклассы `SingleFragmentActivity` реализуют этот метод так, чтобы он возвращал экземпляр фрагмента, хостом которого является активность.

Использование абстрактного класса

Попробуем использовать класс с `CrimeListActivity`. Создайте новый класс с именем `CrimeListActivity`, назначьте его суперклассом `SingleFragmentActivity`, удалите реализацию `onCreate(Bundle)` и реализуйте метод `createFragment()` так, как показано в листинге 9.9.

Листинг 9.9. Переработка `CrimeListActivity` (`CrimeListActivity.java`)

```
public class CrimeActivity extends FragmentActivity SingleFragmentActivity {
    /** Вызывается при исходном создании активности. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);
        if (fragment == null) {
            fragment = new CrimeFragment();
            fm.beginTransaction()
                .add(R.id.fragment_container, fragment)
                .commit();
        }
    }
    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
    }
}
```

Создание новых контроллеров

А сейчас мы создадим два новых класса-контроллера: `CrimeListActivity` и `CrimeListFragment`.

Щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.criminalintent`, выберите команду `New ▸ Java Class` и присвойте классу имя `CrimeListActivity`.

Измените новый класс `CrimeListActivity` так, чтобы он тоже субклассировал `SingleFragmentActivity` и реализовал метод `createFragment()`.

Листинг 9.10. Реализация `CrimeListActivity` (`CrimeListActivity.java`)

```
public class CrimeListActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return new CrimeListFragment();
    }
}
```

Если ваш класс `CrimeListActivity` содержит другие методы, такие как `onCreate`, — удалите их. Пусть класс `SingleFragmentActivity` выполняет свою работу, а реализация `CrimeListActivity` будет по возможности простой.

Класс `CrimeListFragment` еще не был создан. Исправим этот недочет.

Снова щелкните правой кнопкой мыши на пакете `com.bignerdranch.android.criminalintent`, выберите команду `New ▶ Java Class` и присвойте классу имя `CrimeListFragment`.

Листинг 9.11. Реализация `CrimeListFragment` (`CrimeListActivity.java`)

```
public class CrimeListFragment extends Fragment {
    // Пока пусто
}
```

Пока `CrimeListFragment` остается пустой оболочкой фрагмента. Мы будем работать с этим классом позднее в этой главе.

Класс `SingleFragmentActivity` существенно сократит объем кода и сэкономит немало времени в процессе чтения. А пока ваш код активности стал аккуратным и компактным.

Объявление `CrimeListActivity`

Теперь, когда класс `CrimeListActivity` создан, его следует объявить в манифесте. Кроме того, список преступлений должен выводиться на первом экране, который виден пользователю после запуска `CriminalIntent`; следовательно, активность `CrimeListActivity` должна быть активностью лаунчера.

Включите в манифест объявление `CrimeListActivity` и переместите фильтр интенгов из объявления `CrimeActivity` в объявление `CrimeListActivity`, как показано в листинге 9.12.

Листинг 9.12. Объявление `CrimeListActivity` активностью лаунчера (`AndroidManifest.xml`)

```
...
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity android:name=".CrimeListActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".CrimeActivity"
        android:label="@string/app_name">
        <intent-filter>
```

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

CrimeListActivity теперь является активностью лаунчера. Запустите Criminal-Intent; на экране появляется виджет FrameLayout из CrimeListActivity, содержащий пустой фрагмент CrimeListFragment (рис. 9.3).

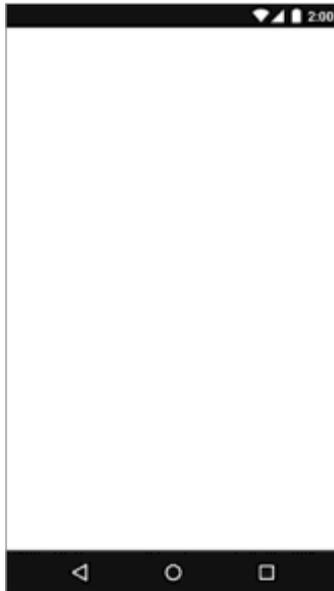


Рис. 9.3. Пустой экран CrimeListActivity

RecyclerView, Adapter и ViewHolder

Итак, в CrimeListFragment должен отображаться список. Для этого мы воспользуемся классом RecyclerView.

Класс RecyclerView является субклассом ViewGroup. Он выводит список дочерних объектов View, по одному для каждого элемента. В зависимости от сложности отображаемых данных дочерние объекты View могут быть сложными или очень простыми.

Наша первая реализация передачи данных для отображения будет очень простой: в элементе списка будет отображаться только краткое описание объекта Crime, а объект View представляет собой простой виджет TextView (рис. 9.4).

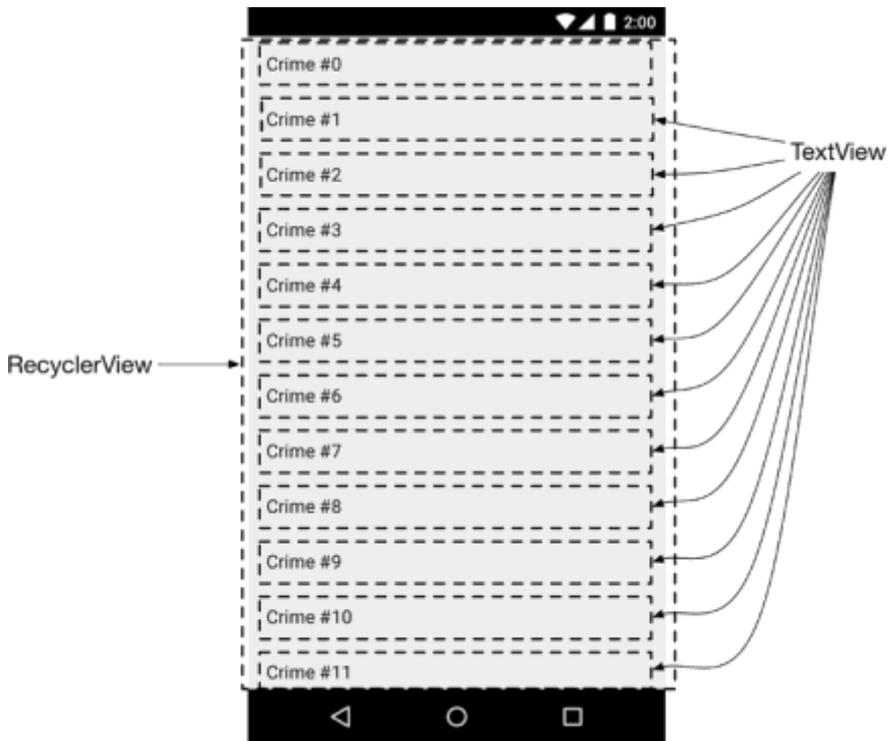


Рис. 9.4. RecyclerView с дочерними виджетами TextView

На рис. 9.4 изображены 12 виджетов TextView. Позднее вы сможете запустить `CriminalIntent` и провести по экрану, чтобы прокрутить 100 виджетов TextView для просмотра всех объектов `Crime`. Значит ли это, что вам придется управлять 100 объектами TextView? Благодаря RecyclerView — нет, не придется.

Решение с созданием TextView для каждого элемента списка быстро становится нереальным. Как нетрудно представить, список может содержать намного более 100 элементов, а виджеты TextView могут быть намного сложнее использованной нами простой реализации. Кроме того, представление View необходимо объекту `Crime` только во время его нахождения на экране, поэтому держать наготове 100 представлений нет необходимости. Будет намного эффективнее создавать объекты представлений только тогда, когда они действительно необходимы.

Виджет RecyclerView именно так и поступает. Вместо того чтобы создавать 100 представлений View, он создает 12 — достаточно для заполнения экрана. А когда виджет View выходит за пределы экрана, RecyclerView использует его заново вместо того, чтобы просто выбрасывать. Короче говоря, RecyclerView снова и снова перерабатывает использованные виджеты представлений.

ViewHolder и Adapter

Единственная обязанность RecyclerView — повторное использование виджетов TextView и их позиционирование на экране. Чтобы обеспечить их исходное размещение, он работает с двумя классами, которые мы вскоре построим: subclass Adapter и subclass ViewHolder.

Обязанности класса ViewHolder невелики, поэтому начнем с него. ViewHolder делает только одно: он удерживает объект View (рис. 9.5).

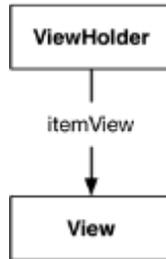


Рис. 9.5. Прimitивная работа ViewHolder

Не много, но для этого ViewHolder и существует. Типичный subclass ViewHolder выглядит так:

Листинг 9.13. Типичный subclass ViewHolder

```

public class ListRow extends RecyclerView.ViewHolder {
    public ImageView mThumbnail;

    public ListRow(View view) {
        super(view);

        mThumbnail = (ImageView) view.findViewById(R.id.thumbnail);
    }
}
  
```

После этого вы можете создать объект ListRow и работать как с переменной mThumbnail, которую вы создали сами, так и с переменной itemView — полем, значение которого суперкласс RecyclerView.ViewHolder назначает за вас. Поле itemView — главная причина для существования ViewHolder: в нем хранится ссылка на все представление View, переданное super(view).

Листинг 9.14. Типичное использование ViewHolder

```

ListRow row = new ListRow(inflater.inflate(R.layout.list_row, parent,
    false));
View view = row.itemView;
ImageView thumbnailView = row.mThumbnail;
  
```

Виджет RecyclerView никогда не создает объекты View. Он всегда создает объекты ViewHolder, которые приносят itemView с собой (рис. 9.6).

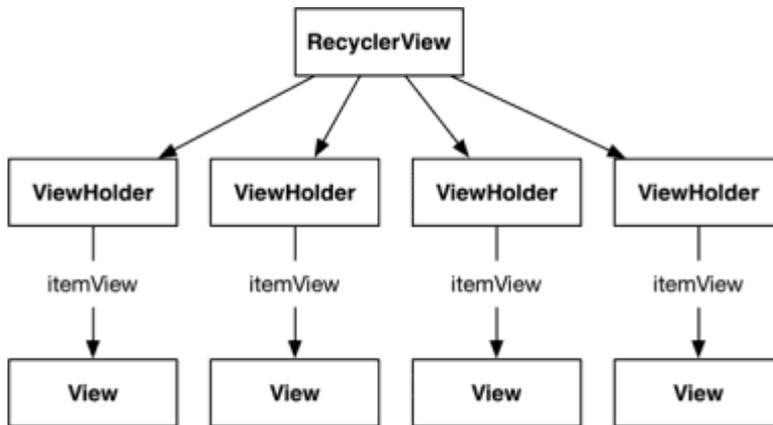


Рис. 9.6. RecyclerView с объектами ViewHolder

Для простых представлений View класс ViewHolder имеет минимум обязанностей. Для более сложных View упрощает связывание различных частей itemView с Crime и повышает его эффективность.

Вы увидите, как работает этот механизм, позднее в этой главе, когда мы займемся построением сложного представления View.

Адаптеры

Схема на рис. 9.6 несколько упрощена. Виджет RecyclerView не создает ViewHolder самостоятельно. Вместо этого он обращается с запросом к *адаптеру* (adapter) — объекту контроллера, который находится между RecyclerView и набором данных с информацией, которую должен вывести RecyclerView.

Адаптер отвечает за:

- ① создание необходимых объектов ViewHolder;
- ① связывание ViewHolder с данными из уровня модели.

Построение адаптера начинается с определения subclasses RecyclerView.Adapter. Ваш subclass адаптера инкапсулирует список преступлений, полученных от CrimeLab.

Когда виджету RecyclerView требуется объект представления для отображения, он вступает в диалог со своим адаптером. На рис. 9.7 приведен пример возможного диалога, который может быть инициирован RecyclerView.

Сначала RecyclerView запрашивает общее количество объектов в списке, для чего он вызывает метод getItemCount() адаптера.

Затем RecyclerView вызывает метод onCreateViewHolder(ViewGroup, int) адаптера для создания нового объекта ViewHolder вместе с его «полезной нагрузкой»: отображаемым представлением.

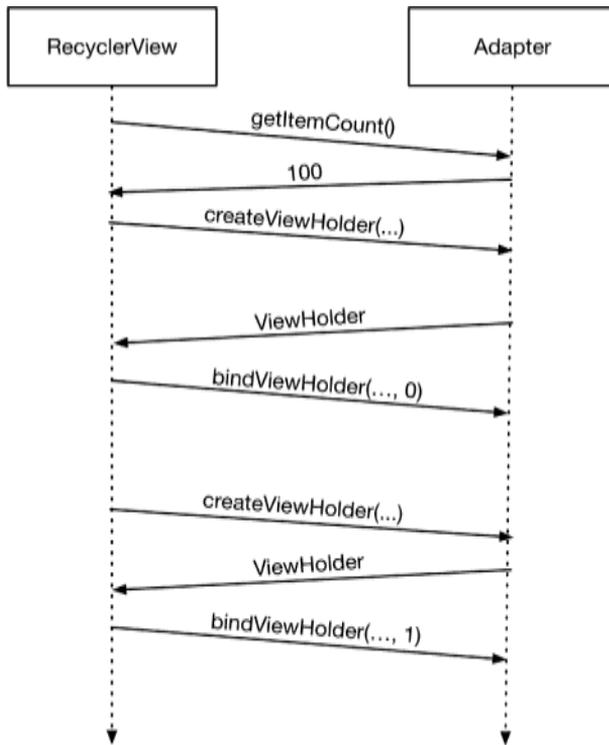


Рис. 9.7. Диалог RecyclerView-Adapter

Наконец, RecyclerView вызывает `onBindViewHolder(ViewHolder, int)`. RecyclerView передает этому методу объект `ViewHolder` и позицию. Адаптер получает данные модели для указанной позиции и *связывает* их с представлением `View` объекта `ViewHolder`. Чтобы выполнить связывание, адаптер заполняет `View` в соответствии с данными из объекта модели.

После завершения этого процесса RecyclerView помещает элемент списка на экран. Обратите внимание: метод `createViewHolder(ViewGroup, int)` вызывается намного реже, чем `onBindViewHolder(ViewHolder, int)`. После того как будет создано достаточное количество объектов `ViewHolder`, RecyclerView перестает вызывать `createViewHolder(...)`. Вместо этого он экономит время и память за счет переработки старых объектов `ViewHolder`.

Использование RecyclerView

Но довольно разговоров, пора взяться за реализацию. Класс `RecyclerView` находится в одной из многочисленных библиотек поддержки Google. Первым шагом в использовании `RecyclerView` станет добавление библиотеки `RecyclerView` к зависимостям приложения.

Откройте окно структуры проекта командой File ▶ Project Structure.... Выберите модуль app слева, перейдите на вкладку Dependencies. Щелкните на кнопке + и выберите Library dependency, чтобы добавить зависимость.

Найдите и выделите библиотеку recyclerview-v7; щелкните на кнопке ОК, чтобы добавить библиотеку как зависимость (рис. 9.8).

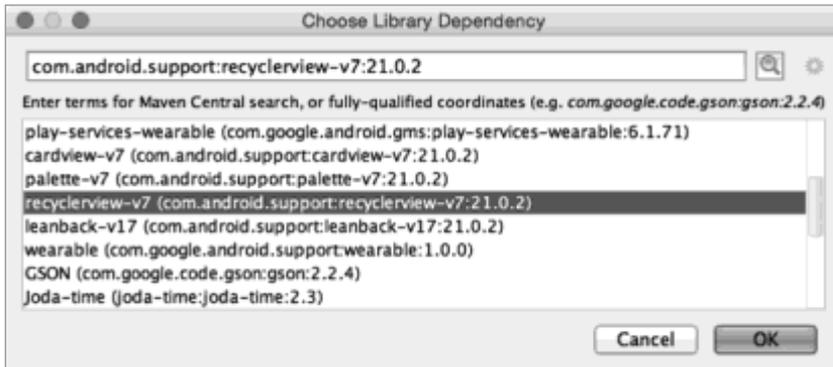


Рис. 9.8. Добавление зависимости для RecyclerView

Виджет RecyclerView будет находиться в файле макета CrimeListFragment. Сначала необходимо создать файл макета: щелкните правой кнопкой мыши на каталоге res/layout и выберите команду New ▶ Layout resource file. Введите имя fragment_crime_list и щелкните на кнопке ОК, чтобы создать файл.

Откройте только что созданный файл fragment_crime_list, замените его корневое представление на RecyclerView и присвойте ему идентификатор.

Листинг 9.15. Включение RecyclerView в файл макета (fragment_crime_list.xml)

```
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/crime_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Представление CrimeListFragment готово; теперь его нужно связать с фрагментом. Измените класс CrimeListFragment так, чтобы он использовал этот файл макета и находил RecyclerView в файле макета, как показано в листинге 9.16.

Листинг 9.16. Подготовка представления для CrimeListFragment (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;

    @Override
```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_crime_list, container,
        false);

    mCrimeRecyclerView = (RecyclerView) view
        .findViewById(R.id.crime_recycler_view);
    mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager
        (getActivity()));

    return view;
}
}
```

Обратите внимание: сразу же после создания виджета RecyclerView ему назначается другой объект `LayoutManager`. Это необходимо для работы виджета RecyclerView. Если вы забудете предоставить ему объект `LayoutManager`, возникнет ошибка.

Ранее мы говорили, что обязанности RecyclerView сводятся к переработке виджетов `TextView` и размещению их на экране. Но виджет RecyclerView не занимается размещением элементов на экране самостоятельно — он поручает эту задачу `LayoutManager`. Объект `LayoutManager` управляет позиционированием элементов, а также определяет поведение прокрутки. Таким образом, при отсутствии `LayoutManager` виджет RecyclerView просто погибнет в тщетной попытке что-нибудь сделать. Возможно, в будущем ситуация изменится, но сейчас дело обстоит именно так.

Вам на выбор предоставляются несколько встроенных вариантов `LayoutManagers`; другие варианты можно найти в сторонних библиотеках. Мы будем использовать класс `LinearLayoutManager`, который размещает элементы в вертикальном списке. Позднее вы научитесь использовать `GridLayoutManager` для размещения элементов в виде таблицы.

Запустите приложение. Вы снова видите пустой экран, но сейчас перед вами пустой виджет RecyclerView. Объекты `Crime` остаются невидимыми до тех пор, пока не будут определены реализации `Adapter` и `ViewHolder`.

Реализация адаптера и ViewHolder

Начнем с определения `ViewHolder` как внутреннего класса `CrimeListFragment`.

Листинг 9.17. Простая реализация `ViewHolder` (`CrimeListFragment.java`)

```
public class CrimeListFragment extends Fragment {
    ...

    private class CrimeHolder extends RecyclerView.ViewHolder {

        public TextView mTitleTextView;

        public CrimeHolder(View itemView) {
            super(itemView);
        }
    }
}
```

```

        mTitleTextView = (TextView) itemView;
    }
}

```

В своем текущем виде ViewHolder хранит ссылку на одно представление: виджет TextView для заголовка. Ожидается, что itemView относится к типу TextView;; в противном случае при выполнении кода произойдет ошибка. Позднее в этой главе обязанности ViewHolder будут расширены.

После определения ViewHolder создайте адаптер.

Листинг 9.18. Начало работы над адаптером (CrimeListFragment.java)

```

public class CrimeListFragment extends Fragment {
    ...

    private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
        private List<Crime> mCrimes;

        public CrimeAdapter(List<Crime> crimes) {
            mCrimes = crimes;
        }
    }
}

```

(Код в листинге 9.18 не компилируется. Вскоре мы исправим этот недостаток.)

Класс RecyclerView взаимодействует с адаптером, когда потребуется создать объект ViewHolder или связать его с объектом Crime. Сам виджет RecyclerView ничего не знает об объекте Crime, но адаптер располагает полной информацией о Crime.

Затем реализуйте три метода CrimeAdapter.

Листинг 9.19. Реализация методов CrimeAdapter (CrimeListFragment.java)

```

private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...

    @Override
    public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(getActivity());
        View view = inflater
            .inflate(android.R.layout.simple_list_item_1, parent, false);
        return new CrimeHolder(view);
    }

    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {
        Crime crime = mCrimes.get(position);
        holder.mTitleTextView.setText(crime.getTitle());
    }

    @Override

```

```
    public int getItemCount() {  
        return mCrimes.size();  
    }  
}
```

В этом коде далеко не все очевидно. Начнем с реализации `onCreateViewHolder`.

Метод `onCreateViewHolder` вызывается виджетом `RecyclerView`, когда ему требуется новое представление для отображения элемента. В этом методе мы создаем объект `View` и упаковываем его в `ViewHolder`. `RecyclerView` пока не ожидает, что представление будет связано с какими-либо данными.

Для получения представления мы заполняем макет из стандартной библиотеки `Android` с именем `simple_list_item_1`. Этот макет содержит один виджет `TextView`, оформленный так, чтобы он хорошо смотрелся в списке. Позднее в этой главе мы создадим более сложное представление для элементов списка.

Далее идет `onBindViewHolder`: этот метод связывает представление `View` объекта `ViewHolder` с объектом модели. При вызове он получает `ViewHolder` и позицию в наборе данных. Позиция используется для нахождения правильных данных модели, после чего `View` обновляется в соответствии с этими данными.

В нашей реализации эта позиция определяет индекс объекта `Crime` в массиве. После получения нужного объекта мы связываем его с `View`, присваивая его заголовок виджету `TextView` из `ViewHolder`.

Итак, адаптер готов; остается связать его с `RecyclerView`. Реализуйте метод `updateUI`, который настраивает пользовательский интерфейс `CrimeListFragment`. Пока он создает объект `CrimeAdapter` и назначает его `RecyclerView`.

Листинг 9.20. Подготовка адаптера (`CrimeListFragment.java`)

```
public class CrimeListFragment extends Fragment {  
  
    private RecyclerView mCrimeRecyclerView;  
    private CrimeAdapter mAdapter;  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.fragment_crime_list,  
                                    container, false);  
        mCrimeRecyclerView = (RecyclerView) view  
            .findViewById(R.id.crime_recycler_view);  
        mCrimeRecyclerView.setLayoutManager(new LinearLayoutManager  
            (getActivity()));  
  
        updateUI();  
  
        return view;  
    }  
  
    private void updateUI() {  
        CrimeLab crimeLab = CrimeLab.get(getActivity());  
        List<Crime> crimes = crimeLab.getCrimes();  
    }  
}
```

```
mAdapter = new CrimeAdapter(crimes);  
mCrimeRecyclerView.setAdapter(mAdapter);  
}  
...  
}
```

В следующих главах метод `updateUI()` будет расширяться по мере усложнения пользовательского интерфейса.

Запустите приложение `CriminalIntent` и прокрутите новый список `RecyclerView`, который должен выглядеть примерно так, как показано на рис. 9.9.



Рис. 9.9. Список объектов Crime

Настройка элементов списка

До настоящего момента в каждом элементе списка выводится только краткое описание `Crime` в простом текстовом поле `TextView`.

Что делать, если элементы списка не должны ограничиваться выводом простого текста? Если вы хотите изменить дизайн элементов? Перемещение представления элемента списка в отдельный файл макета позволит решить обе задачи и одновременно сделает код более аккуратным.

Создание макета элемента списка

В приложении CriminalIntent макет элемента списка должен включать краткое описание преступления, дату и признак раскрытия (рис. 9.10). Такой макет состоит из двух виджетов — `TextView` и `CheckBox`.



Рис. 9.10. Список с пользовательским макетом элементов

Новый макет элемента списка создается точно так же, как для представления активности или фрагмента. В окне инструментов Project щелкните правой кнопкой мыши на каталоге `res/layout` и выберите команду `New` ▶ `Layout resource file`. В открывшемся диалоговом окне введите имя файла `list_item_crime`, выберите корневой элемент `RelativeLayout` и щелкните на кнопке `OK`.

`RelativeLayout` позволяет использовать параметры макета для размещения дочерних представлений относительно корневого макета и друг друга. Мы хотим, чтобы виджет `CheckBox` автоматически выравнивался по правой стороне `RelativeLayout`. Два виджета `TextView` будут выравниваться относительно `CheckBox`.

На рис. 9.11 изображены виджеты макета пользовательского элемента списка. Виджет `CheckBox` должен определяться первым, несмотря на то что он находится у правого края макета. Это связано с тем, что `TextView` будет использовать идентификатор `CheckBox` в значении атрибута. По той же причине определение виджета `TextView` с кратким описанием должно предшествовать определению виджета `TextView` с датой. В файле макета виджет должен определяться до того, как другие виджеты смогут использовать его идентификатор в своих определениях.

Обратите внимание: при использовании идентификатора виджета в определении другого виджета не нужно включать знак `+`. Он используется для создания идентификатора при его первом вхождении в файл макета, обычно в атрибуте `android:id`.

Макет пользовательского элемента списка завершен, и мы можем переходить к следующему шагу — обновлению адаптера.

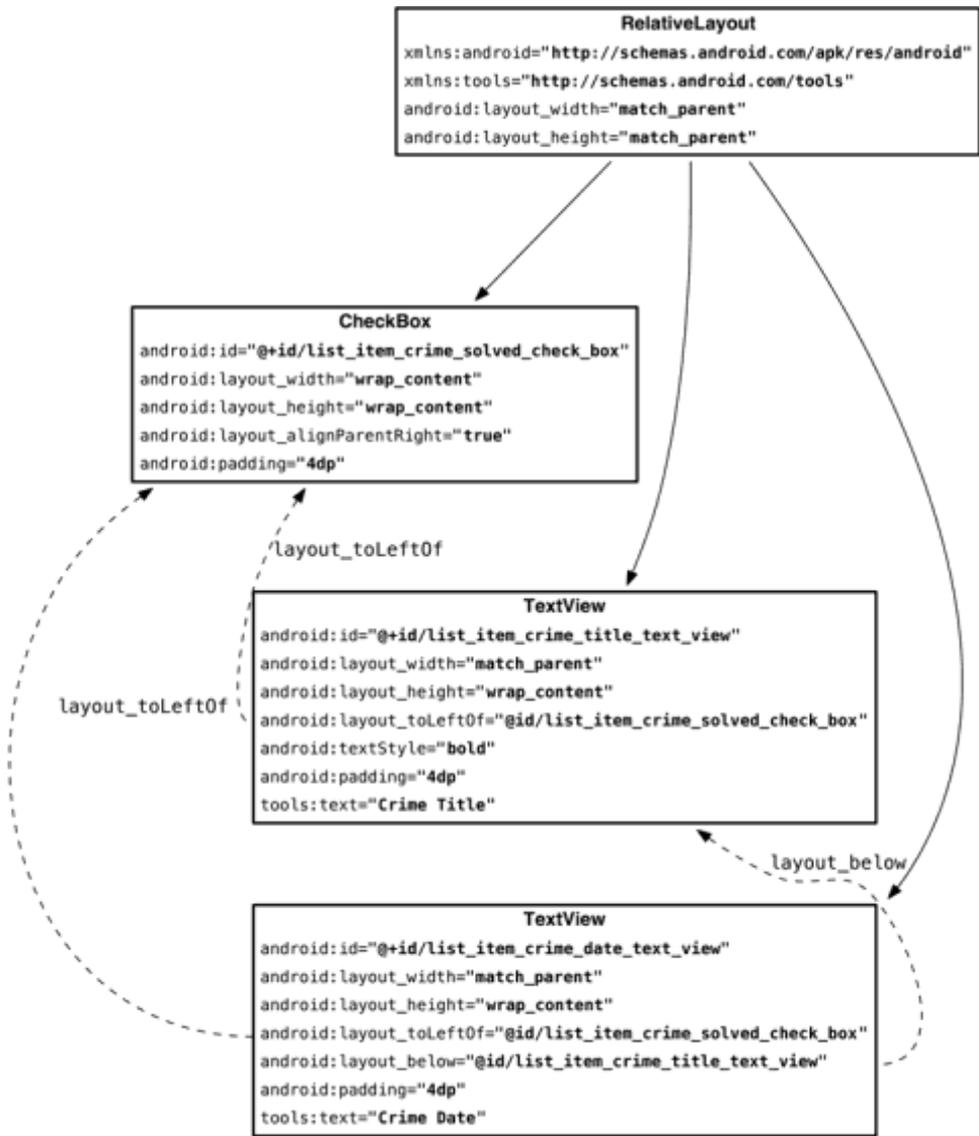


Рис. 9.11. Макет пользовательского элемента списка (list_item_crime.xml)

Использование нового представления элемента списка

Теперь внесите изменения в класс CrimeAdapter, чтобы использовать новый файл макета list_item_crime.

Листинг 9.21. Заполнение пользовательского макета (CrimeListFragment.java)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {
    ...
    @Override
    public CrimeHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        LayoutInflater inflater = LayoutInflater.from(getActivity());
        View view = inflater
            .inflate(android.R.layout.simple_list_item_1
                R.layout.list_item_crime, parent, false);
        return new CrimeHolder(view);
    }
    ...
}
```

Наконец, пора наделить CrimeHolder новыми обязанностями. Внесите изменения в класс CrimeHolder, чтобы он получал виджет TextView с кратким описанием, TextView с датой и CheckBox с признаком раскрытия преступления.

Листинг 9.22. Поиск представлений в CrimeHolder (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder {
    public TextView mTitleTextView;
    private TextView mTitleTextView;
    private TextView mDateTextView;
    private CheckBox mSolvedCheckBox;

    public CrimeHolder(View itemView) {
        super(itemView);

        mTitleTextView = (TextView) itemView;
        mTitleTextView = (TextView)
            itemView.findViewById(R.id.list_item_crime_title_text_view);
        mDateTextView = (TextView)
            itemView.findViewById(R.id.list_item_crime_date_text_view);
        mSolvedCheckBox = (CheckBox)
            itemView.findViewById(R.id.list_item_crime_solved_check_box);
    }
}
```

Здесь объекты ViewHolder проявляют себя. Вызовы findViewById(int) часто обходятся достаточно дорого. Они обходят itemView шаг за шагом в поисках нужного представления: «Здравствуйте, вы list_item_crime_title_text_view? Нет? Извините за беспокойство». Это требует времени, а для выполнения поиска приходится обходить все окружение в памяти.

Объекты ViewHolder способны существенно упростить задачу. Сохраняя результаты вызовов findViewById(int), вы будете тратить время только в onCreateViewHolder(...). При вызове onBindViewHolder(...) эта работа уже проделана, и это хорошо, потому что onBindViewHolder(...) вызывается намного чаще onCreateViewHolder(...).

С другой стороны, процесс связывания несколько усложняется. Добавьте в `CrimeHolder` метод `bindCrime(Crime)`, чтобы немного упростить код.

Листинг 9.23. Связывание представлений в `CrimeHolder` (`CrimeListFragment.java`)

```
private class CrimeHolder extends RecyclerView.ViewHolder {

    private Crime mCrime;

    ---
    public void bindCrime(Crime crime) {
        mCrime = crime;
        mTitleTextView.setText(mCrime.getTitle());
        mDateTextView.setText(mCrime.getDate().toString());
        mSolvedCheckBox.setChecked(mCrime.isSolved());
    }
}
```

Получив объект `Crime`, объект `CrimeHolder` обновляет `TextView` с кратким описанием, `TextView` с датой и `CheckBox` с признаком раскрытия преступления в соответствии с содержимым `Crime`.

Теперь у `CrimeHolder` есть все необходимое для выполнения его работы. `CrimeAdapter` достаточно использовать новый метод `bindCrime`.

Листинг 9.24. Связывание адаптера с `CrimeHolder` (`CrimeListFragment.java`)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {

    ---
    @Override
    public void onBindViewHolder(CrimeHolder holder, int position) {
        Crime crime = mCrimes.get(position);
        holder.mTitleTextView.setText(crime.getTitle());
        holder.bindCrime(crime);
    }
    ---
}
```

Запустите `CriminalIntent` и понаблюдайте за новым макетом `list_item_crime` в действии (рис. 9.12).

Щелчки на элементах списка

Вместе с `RecyclerView` вы получаете приятное бесплатное приложение: `CriminalIntent` теперь может реагировать на касания элементов списка.

В главе 10 при касании объекта `Crime` в списке будет запускаться представление детализации. Пока ограничимся простым отображением уведомления `Toast`.



Рис. 9.12. А теперь с новыми макетами элементов!

Как вы, возможно, заметили, виджет RecyclerView при всей своей мощи и широте возможностей берет на себя минимум реальных обязанностей (нам остается только завидовать). То же происходит и в этом случае: обработкой событий касания в основном придется заниматься вам. RecyclerView может передавать вам низкоуровневые события касания, если они вам нужны. Впрочем, в большинстве случаев это излишне.

Вместо этого можно обрабатывать касания так, как это обычно делается: назначением обработчика OnClickListener. Так как каждое представление View связывается с некоторым ViewHolder, вы можете сделать объект ViewHolder реализацией OnClickListener для своего View.

Внесите изменения в класс CrimeHolder для обработки касаний в строках.

Листинг 9.25. Обработка касаний в CrimeHolder (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ...
    public CrimeHolder(View itemView) {
        super(itemView);
        itemView.setOnClickListener(this);
    }
    ...
}
```

```
@Override
public void onClick(View v) {
    Toast.makeText(getActivity(),
        mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
        .show();
}
}
```

В листинге 9.25 сам объект `CrimeHolder` реализует интерфейс `OnClickListener`. Для `itemView` — представления `View` всей строки — `CrimeHolder` назначается получателем событий щелчка.

Запустите приложение `CriminalIntent` и коснитесь строки в списке. На экране появляется уведомление `Toast` с сообщением о касании.

Для любознательных: `ListView` и `GridView`

Базовый вариант ОС Android включает классы `ListView`, `GridView` и `Adapter`. До выхода Android 5.0 эти классы считались наиболее правильным способом создания разнообразных списков или таблиц.

API этих компонентов очень близки к API `RecyclerView`. Класс `ListView` или `GridView` отвечает за прокрутку наборов вариантов, но почти ничего не знает об отдельных элементах списка. `Adapter` отвечает за создание всех представлений в списке. С другой стороны, классы `ListView` и `GridView` не заставляют применять паттерн `ViewHolder` (хотя вы можете — и даже должны — применять его).

Сейчас старые реализации практически вытеснены `RecyclerView` из-за сложностей, связанных с изменением поведения `ListView` или `GridView`. Например, возможность создания `ListView` с горизонтальной прокруткой не включена в API `ListView`, а ее реализация требует значительной работы. Создание нестандартного макета и реализация прокрутки в `RecyclerView` все равно требует значительной работы, но класс `RecyclerView` строился в расчете на расширение, поэтому все не так плохо.

Другая важная особенность `RecyclerView` — анимация элементов списка. Анимация добавления и удаления элементов в `ListView` или `GridView` — сложная, ненадежная задача. `RecyclerView` существенно упрощает ее, включает несколько встроженных анимаций и позволяет легко настроить эти анимации.

Например, если обнаруживается, что элемент в позиции 0 переместился в позицию 5, перемещение можно анимировать следующим образом:

```
mRecyclerView.getAdapter().notifyItemMoved(0, 5);
```

Для любознательных: синглеты

Паттерн «Синглет», используемый в `CrimeLab`, очень часто применяется в Android. Синглеты пользуются дурной славой, потому что они открывают возможности для многочисленных злоупотреблений, усложняющих сопровождение кода.

Синглеты часто применяются в Android, так как их они живут дольше отдельных фрагментов или активностей. Синглет переживает повороты устройства и существует при перемещении между активностями и фрагментами в приложении.

Синглеты могут стать удобными владельцами объектов модели. Представьте более сложное приложение `CriminalIntent` с множеством активностей и фрагментов, изменяющих информацию о преступлениях. Когда один контроллер изменяет преступление, как организовать передачу информации об обновленном преступлении другим контроллерам? Если `CrimeLab` является владельцем объектов преступлений и все изменения в них проходят через `CrimeLab`, распространение информации об изменениях значительно упрощается. При передаче управления между контроллерами можно передавать идентификатор преступления, а каждый контроллер извлекает полный объект преступления из `CrimeLab` по идентификатору.

Тем не менее у синглетов имеются свои недостатки. Например, хотя они хорошо подходят для хранения данных с большим сроком жизни, чем у контроллера, у синглетов все же имеется срок жизни. Android в какой-то момент после переключения от приложения может освободить память, и тогда синглеты будут уничтожены вместе со всеми переменными экземпляров. Синглеты не подходят для долгосрочного хранения данных (в отличие, скажем, от записи файлов на диск или их отправки на веб-сервер).

Синглеты также усложняют модульное тестирование кода. Экземпляры `CrimeLab` трудно заменить их фиктивными версиями, потому что код напрямую вызывает статический метод объекта `CrimeLab`. На практике Android-разработчики обычно решают эту проблему при помощи инструмента, называемого *внедрителем зависимостей*. Он позволяет организовать использование синглетных объектов, сохраняя при этом возможность замены их при необходимости.

Как говорилось ранее, синглеты часто используются некорректно. У программиста возникает искушение применять синглеты везде и повсюду, потому что они удобны: к ним можно обращаться откуда угодно и в них можно хранить любую информацию, которая понадобится вам позднее. Но при этом вы не пытаетесь ответить на важные вопросы: где используются эти данные? Чем важен этот метод?

Синглет обходит эти вопросы. Любой разработчик, который придет после вас, откроет синглет и обнаружит некое подобие сундука, в который сваливают всякий хлам: батарейки, стяжки для кабелей, старые фотографии. Зачем здесь все это? Убедитесь в том, что все содержимое синглета действительно глобально и для хранения данных в синглете существуют веские причины.

Впрочем, с учетом всего сказанного синглеты являются ключевым компонентом хорошо спланированного приложения Android — при условии правильного использования.

10

Аргументы фрагментов

В этой главе мы наладим совместную работу списка и детализации в приложении `CriminalIntent`. Когда пользователь щелкает на элементе списка преступлений, на экране возникает новый экземпляр `CrimeActivity`, который является хостом для экземпляра `CrimeFragment` с подробной информацией о конкретном экземпляре `Crime` (рис. 10.1).

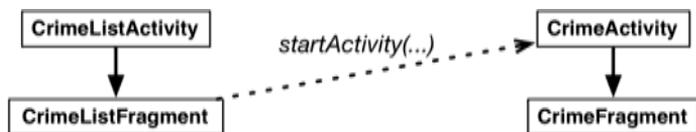


Рис. 10.1. Запуск `CrimeActivity` из `CrimeListActivity`

В приложении `GeoQuiz` одна активность (`QuizActivity`) запускала другую (`CheatActivity`). В приложении `CriminalIntent` активность `CrimeActivity` будет запускаться из фрагмента `CrimeListFragment`.

Запуск активности из фрагмента

Запуск активности из фрагмента осуществляется практически так же, как запуск активности из другой активности. Вы вызываете метод `Fragment.startActivity(Intent)`, который вызывает соответствующий метод `Activity` во внутренней реализации.

В реализации `onListItemClick(...)` из `CrimeListFragment` замените уведомление кодом, запускающим экземпляр `CrimeActivity`.

Листинг 10.1. Запуск `CrimeActivity` (`CrimeListFragment.java`)

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
```

...

```
@Override
public void onClick(View v) {
    Toast.makeText(getActivity(),
        mCrime.getTitle() + " clicked!", Toast.LENGTH_SHORT)
        .show();

    Intent intent = new Intent(getActivity(), CrimeActivity.class);
    startActivity(intent);
}
}
```

Здесь класс `CrimeListFragment` создает явный интент с указанием класса `CrimeActivity`. `CrimeListFragment` использует метод `getActivity()` для передачи активности-хоста как объекта `Context`, необходимого конструктору `Intent`.

Запустите приложение `CriminalIntent`. Щелкните на любой строке списка; открывается новый экземпляр `CrimeActivity`, управляющий фрагментом `CrimeFragment` (рис. 10.2).



Рис. 10.2. Пустой экземпляр `CrimeFragment`

Экземпляр `CrimeFragment` еще не содержит данных конкретного объекта `Crime`, потому что мы не сообщили ему, какой именно объект `Crime` следует отображать.

Включение дополнения

Чтобы сообщить CrimeFragment, какой объект Crime следует отображать, можно передать идентификатор в дополнении (extra) объекта Intent при запуске CrimeActivity.

Начните с создания нового метода newIntent в CrimeActivity.

Листинг 10.2. Создание нового метода newIntent (CrimeActivity.java)

```
public class CrimeActivity extends SingleFragmentActivity {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";
    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimeActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }
    ---
}
```

После создания явного интента мы вызываем putExtra(...), передавая строковый ключ и связанное с ним значение (crimeId). В данном случае вызывается версия putExtra(String, Serializable), потому что UUID является объектом Serializable.

Затем обновите класс CrimeHolder, чтобы он использовал метод newIntent с передачей идентификатора преступления.

Листинг 10.3. Сохранение и передача Crime (CrimeListFragment.java)

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ---
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(getActivity(), CrimeActivity.class);
        Intent intent = CrimeActivity.newIntent(getActivity(),
            mCrime.getId());
        startActivity(intent);
    }
}
```

Чтение дополнения

Идентификатор преступления сохранен в интенте, принадлежащем CrimeActivity, однако прочитать и использовать эти данные должен класс CrimeFragment.

Существует два способа, которыми фрагмент может обратиться к данным из интента активности: простое и прямолинейное обходное решение и сложная, гибкая полноценная реализация. Сначала мы опробуем первый способ, а потом реализуем сложное гибкое решение с *аргументами фрагментов*.

В простом решении `CrimeFragment` просто использует метод `getActivity()` для прямого обращения к интенту `CrimeActivity`. Вернитесь к классу `CrimeFragment`, прочитайте дополнение из интента `CrimeActivity` и используйте его для получения данных `Crime`.

Листинг 10.4. Сохранение и передача `Crime` (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {
    ...

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        mCrime = new Crime();
        UUID crimeId = (UUID) getActivity().getIntent()
        .getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
        mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    }
    ...
}
```

Если не считать вызова `getActivity()`, листинг 10.4 практически не отличается от кода выборки дополнения из кода активности. Метод `getIntent()` возвращает объект `Intent`, используемый для запуска `CrimeActivity`. Мы вызываем `getSerializableExtra(String)` для `Intent`, чтобы извлечь `UUID` в переменную.

После получения идентификатора мы используем его для получения объекта `Crime` от `CrimeLab`.

Обновление представления `CrimeFragment` данными `Crime`

Теперь, когда фрагмент `CrimeFragment` получает объект `Crime`, его представление может отобразить данные `Crime`. Обновите метод `onCreateView(...)`, чтобы он выводил краткое описание преступления и признак раскрытия (код вывода даты уже имеется).

Листинг 10.5. Обновление объектов представления (`CrimeFragment.java`)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup parent,
    Bundle savedInstanceState) {
    ...

    mTitleField = (EditText)v.findViewById(R.id.crime_title);
    mTitleField.setText(mCrime.getTitle());
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
    });
    ...
    mSolvedCheckBox = (CheckBox)v.findViewById(R.id.crime_solved);
    mSolvedCheckBox.setChecked(mCrime.isSolved());
}
```

```
mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener() {  
    ...  
});  
...  
return v;  
}
```

Запустите приложение CriminalIntent. Выберите строку Crime #4 и убедитесь в том, что на экране появляется экземпляр CrimeFragment с правильными данными преступления (рис. 10.3).

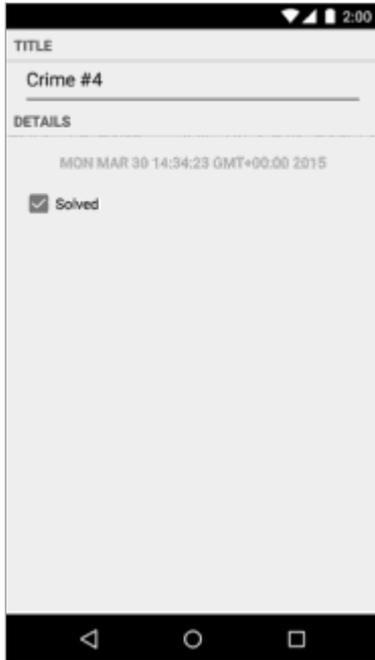


Рис. 10.3. Преступление, выбранное в списке

Недостаток прямой выборки

Обращение из фрагмента к интенту, принадлежащему активности-хосту, упрощает код. С другой стороны, оно нарушает инкапсуляцию фрагмента. Класс CrimeFragment уже не является структурным элементом, пригодным для повторного использования, потому что он предполагает, что его хостом всегда будет активность с объектом Intent, определяющим дополнение с именем com.bignerdranch.android.criminalintent.crime_id.

Возможно, для CrimeFragment такое предположение разумно, но оно означает, что класс CrimeFragment в своей текущей реализации не может использоваться с произвольной активностью.

Другое, более правильное решение — сохранение идентификатора в месте, принадлежащем `CrimeFragment` (вместо хранения его в личном пространстве `CrimeActivity`). В этом случае объект `CrimeFragment` может прочитать данные, не полагаясь на присутствие конкретного дополнения в интенте активности. Такое «место», принадлежащее фрагменту, называется *пакетом аргументов* (`arguments bundle`).

Аргументы фрагментов

К каждому экземпляру фрагмента может быть прикреплен объект `Bundle`. Этот объект содержит пары «ключ-значение», которые работают также, как дополнения интентов `Activity`. Каждая такая пара называется *аргументом* (`argument`).

Чтобы создать аргументы фрагментов, вы сначала создаете объект `Bundle`, а затем используете `put`-методы `Bundle` соответствующего типа (по аналогии с методами `Intent`) для добавления аргументов в пакет:

```
Bundle args = new Bundle();
args.putSerializable(EXTRA_MY_OBJECT, myObject);
args.putInt(EXTRA_MY_INT, myInt);
args.putCharSequence(EXTRA_MY_STRING, myString);
```

Присоединение аргументов к фрагменту

Чтобы присоединить пакет аргументов к фрагменту, вызовите метод `Fragment.setArguments(Bundle)`. Присоединение должно быть выполнено после создания фрагмента, но до его добавления в активность.

Для этого программисты `Android` используют схему с добавлением в класс `Fragment` статического метода с именем `newInstance()`. Этот метод создает экземпляр фрагмента, упаковывает и задает его аргументы.

Когда активности-хосту потребуется экземпляр этого фрагмента, она вместо прямого вызова конструктора вызывает метод `newInstance()`. Активность может передать `newInstance(...)` любые параметры, необходимые фрагменту для создания аргументов.

Включите в `CrimeFragment` метод `newInstance(UUID)`, который получает `UUID`, создает пакет аргументов, создает экземпляр фрагмента, а затем присоединяет аргументы к фрагменту.

Листинг 10.6. Метод `newInstance(UUID)` (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";

    private Crime mCrime;
    private EditText mTitleField;
```

```

private Button mDateButton;
private CheckBox mSolvedCheckbox;

public static CrimeFragment newInstance(UUID crimeId) {
    Bundle args = new Bundle();
    args.putSerializable(ARG_CRIME_ID, crimeId);

    CrimeFragment fragment = new CrimeFragment();
    fragment.setArguments(args);
    return fragment;
}
...
}

```

Теперь класс `CrimeActivity` должен вызывать `CrimeFragment.newInstance(UUID)` каждый раз, когда ему потребуется создать `CrimeFragment`. При вызове передается значение `UUID`, полученное из дополнения. Вернитесь к классу `CrimeActivity`, в методе `createFragment()` получите дополнение из интента `CrimeActivity` и передайте его `CrimeFragment.newInstance(UUID)`.

Константу `EXTRA_CRIME_ID` также можно сделать закрытой, потому что ни одному другому классу не потребуется работать с этим дополнением. (Учтите, что в коде вся строка зачеркнута и заменена для ясности, на практике достаточно заменить `public` на `private` в первом изменении.)

Листинг 10.7. Использование `newInstance(UUID)` (`CrimeActivity.java`)

```

public class CrimeActivity extends SingleFragmentActivity {
    public static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";
    ...

    @Override
    protected Fragment createFragment() {
        return new CrimeFragment();
        UUID crimeId = (UUID) getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);
        return CrimeFragment.newInstance(crimeId);
    }
}

```

Учтите, что потребность в независимости не является двусторонней. Класс `CrimeActivity` должен многое знать о классе `CrimeFragment` — например, то, что он содержит метод `newInstance(UUID)`. Это нормально; активность-хост должна располагать конкретной информацией о том, как управлять фрагментами, но фрагментам такая информация об их активности не нужна (по крайней мере, если вы хотите сохранить гибкость независимых фрагментов).

Получение аргументов

Когда фрагменту требуется получить доступ к его аргументам, он вызывает метод `getArguments()` класса `Fragment`, а затем один из `get`-методов `Bundle` для конкретного типа.

В методе `CrimeFragment.onCreate(...)` замените код упрощенного решения выборкой `UUID` из аргументов фрагмента.

Листинг 10.8. Получение идентификатора преступления из аргументов (`CrimeFragment.java`)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    UUID crimeId = (UUID) getActivity().getIntent().
    getSerializableExtra(CrimeActivity.EXTRA_CRIME_ID);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);

    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}
```

Запустите приложение `CriminalIntent`. Оно работает точно также, но архитектура с независимостью `CrimeFragment` должна вызвать у вас приятные чувства. Кроме того, она хорошо подготовит нас к следующей главе, в которой мы реализуем более сложную систему навигации в `CriminalIntent`.

Перезагрузка списка

Осталась еще одна подробность, которой нужно уделить внимание. Запустите приложение `CriminalIntent`, щелкните на элементе списка и внесите изменения в подробную информацию о преступлении. Эти изменения сохраняются в модели, но при возвращении к списку содержимое `RecyclerView` остается неизменным.

Мы должны сообщить адаптеру `RecyclerView`, что набор данных изменился (или мог измениться), чтобы тот мог заново получить данные и повторно загрузить список. Работая со стеком возврата `FragmentManager`, можно перезагрузить список в нужный момент.

Когда `CrimeListFragment` запускает экземпляр `CrimeActivity`, последний помещается на вершину стека. При этом экземпляр `CrimeActivity`, который до этого находился на вершине, приостанавливается и останавливается.

Когда пользователь нажимает кнопку `Back` для возвращения к списку, экземпляр `CrimeActivity` извлекается из стека и уничтожается. В этот момент `CrimeListActivity` запускается и продолжает выполнение.

Когда экземпляр `CrimeListActivity` продолжает выполнение, он получает вызов `onResume()` от ОС. При получении этого вызова `CrimeListActivity` его экземпляр `FragmentManager` вызывает `onResume()` для фрагментов, хостом которых в настоя-

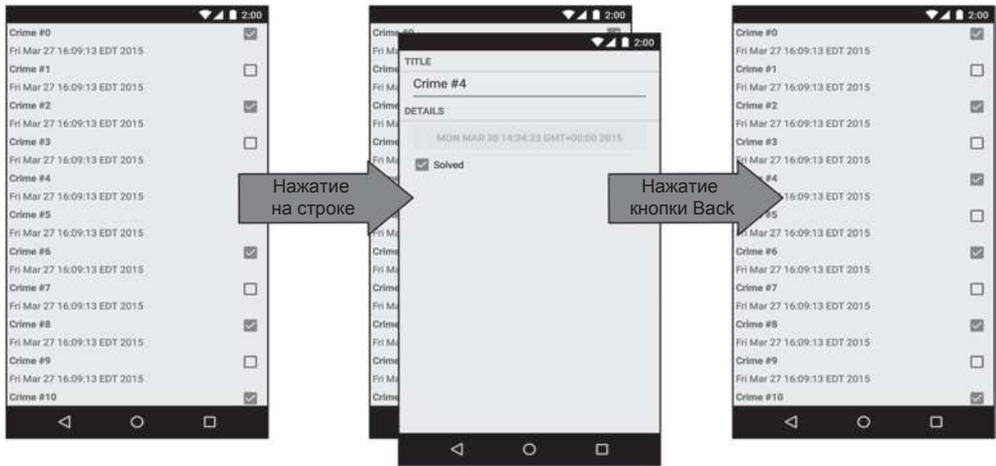


Рис. 10.4. Стек возврата CriminalIntent

щее время является активность. В нашем случае это единственный фрагмент CrimeListFragment.

В классе CrimeListFragment переопределите onResume() и иницируйте вызов updateUI() для перезагрузки списка. Измените метод updateUI() для вызова notifyDataSetChanged(), если объект CrimeAdapter уже создан.

Листинг 10.9. Перезагрузка списка в onResume() (CrimeListFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
}

@Override
public void onResume() {
    super.onResume();
    updateUI();
}

private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }
}

```

Почему для обновления RecyclerView переопределяется метод onResume(), а не onStart()? Мы не можем предполагать, что активность останавливается при нахождении перед ней другой активности. Если другая активность прозрачна, ваша активность может быть только приостановлена. Если же ваша активность приостановлена, а код обновления находится в onStart(), список не будет перезагружаться. В общем случае самым безопасным местом для выполнения действий, связанных с обновлением представления фрагмента, является метод onResume().

Запустите приложение CriminalIntent. Выберите преступление в списке и измените его подробную информацию. Вернувшись к списку, вы немедленно увидите свои изменения.

За последние две главы приложение CriminalIntent серьезно продвинулось вперед. Давайте взглянем на обновленную диаграмму объектов (рис. 10.5).

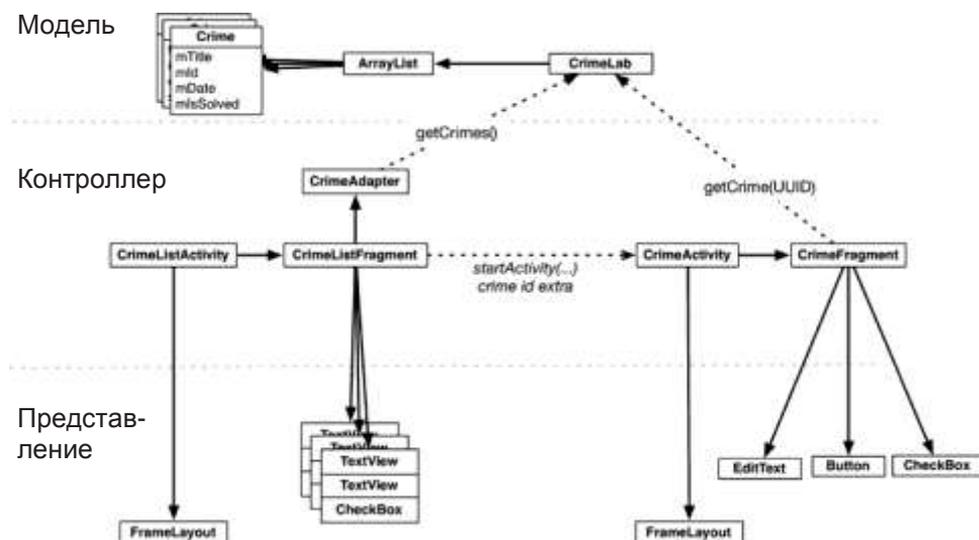


Рис. 10.5. Обновленная диаграмма объектов CriminalIntent

Получение результатов с использованием фрагментов

В этой главе нам не требовалось, чтобы запущенная активность возвращала результат. А если бы это было нужно? Код выглядел бы почти так же, как в приложении GeoQuiz. Вместо метода startActivityForResult(...) класса Activity использовался бы метод Fragment.startActivityForResult(...). Вместо Activity.onActivityResult(...) мы переопределим Fragment.onActivityResult(...).

```
public class CrimeListFragment extends Fragment {
    private static final int REQUEST_CRIME = 1;
```

```

...
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
        Intent intent = CrimeActivity.newIntent(getActivity(),
            mCrime.getId());
        startActivityForResult(intent, REQUEST_CRIME);
    }
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == REQUEST_CRIME) {
        // Обработка результата
    }
}
...
}

```

Метод `Fragment.startActivityForResult(Intent,int)` похож на одноименный метод класса `Activity`. Он включает дополнительный код передачи результата вашему фрагменту от активности-хоста.

Возвращение результата от фрагмента выглядит немного иначе. Фрагмент может получить результат от активности, но не может вернуть собственный результат — на это способны только активности. Таким образом, хотя `Fragment` содержит методы `startActivityForResult(...)` и `onActivityResult(...)`, у него нет методов `setResult(...)`.

Вместо этого вы приказываете *активности-хосту* вернуть значение; вот как это делается:

```

public class CrimeFragment extends Fragment {
    ...
    public void returnResult() {
        getActivity().setResult(Activity.RESULT_OK, null);
    }
}

```

Упражнение. Эффективная перезагрузка RecyclerView

Метод `notifyDataSetChanged` адаптера хорошо подходит для того, чтобы приказывать `RecyclerView` перезагрузить все элементы, видимые в настоящее время.

В `CriminalIntent` этот метод ужасающе неэффективен, потому что при возвращении к `CrimeListFragment` заведомо изменилось не более одного объекта `Crime`.

Используйте метод `notifyItemChanged(int)` объекта `RecyclerView.Adapter`, чтобы перезагрузить один элемент в списке. Изменить код для вызова этого метода несложно; труднее обнаружить, в какой позиции произошло изменение, и перезагрузить правильный элемент.

Для любознательных: зачем использовать аргументы фрагментов?

Все это выглядит как-то сложно. Почему бы просто не задать переменную экземпляра в `CrimeFragment` при создании?

Потому что такое решение будет работать не всегда. Когда ОС создает заново ваш фрагмент (либо вследствие изменения конфигурации, либо при переходе пользователя к другому приложению с последующем освобождением памяти ОС), все переменные экземпляров теряются. Также помните о возможной нехватке памяти.

Если вы хотите, чтобы решение работало во всех случаях, придется сохранять аргументы.

Один из возможных вариантов — использование механизма сохранения состояния экземпляров. Идентификатор преступления заносится в обычную переменную экземпляра, сохраняется вызовом `onSaveInstanceState(Bundle)`, а затем извлекается из `Bundle` в `onCreate(Bundle)`. Такое решение будет работать всегда.

Сдругой стороны, оно усложняет сопровождение. Если вы вернетесь к фрагменту через несколько лет и добавите еще один аргумент, нельзя исключать, что вы забудете сохранить его в `onSaveInstanceState(Bundle)`.

Разработчики Android предпочитают решение с аргументами фрагментов, потому что в этом случае они предельно явно и четко обозначают свои намерения. Через несколько лет вы вернетесь к коду и будете знать, что идентификатор преступления — это аргумент, который надежно передается новым экземплярам этого фрагмента. При добавлении нового аргумента вы знаете, что его нужно сохранить в пакете аргументов.

11

ViewPager

В этой главе мы создадим новую активность, которая станет хостом для CrimeFragment. Макет активности будет состоять из экземпляра ViewPager. Включение виджета ViewPager в пользовательский интерфейс позволяет «листать» элементы списка, проводя пальцем по экрану (рис. 11.1).



Рис. 11.1. Листание страниц

На рис. 11.2 представлена обновленная диаграмма CriminalIntent. Новая активность с именем CrimePagerActivity займет место CrimeActivity. Ее макет состоит из экземпляра ViewPager.

Все новые объекты, которые необходимо создать, находятся в пунктирном прямоугольнике на приведенной диаграмме. Для реализации листания страничных представлений в CriminalIntent ничего другого менять не придется. В частности, класс CrimeFragment останется неизменным благодаря той работе по обеспечению независимости CrimeFragment, которую мы проделали в главе 10.

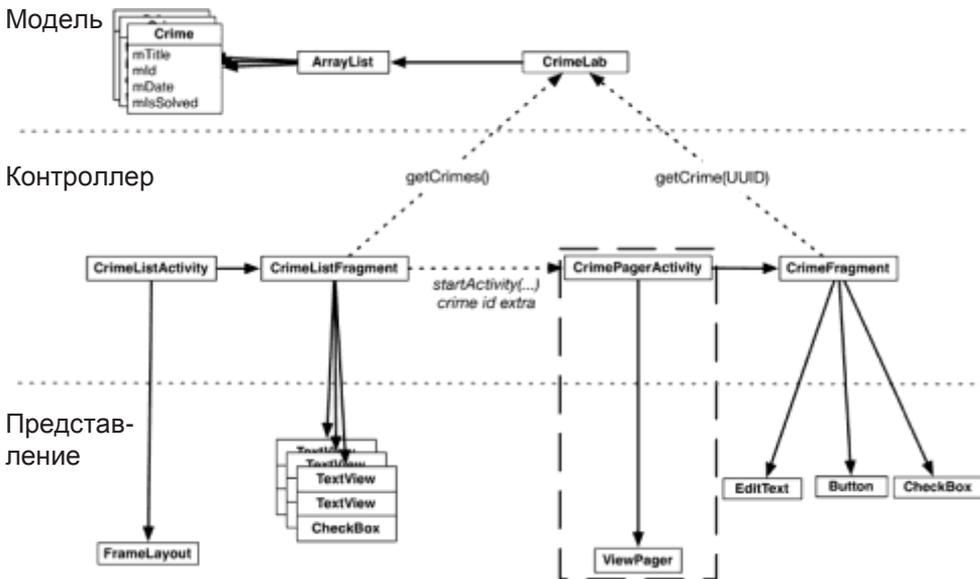


Рис. 11.2. Диаграмма объектов CrimePagerActivity

В этой главе нам предстоит:

- ❶ создать класс CrimePagerActivity;
- ❶ определить иерархию представлений, состоящую из ViewPager;
- ❶ связать экземпляр ViewPager с его адаптером CrimePagerActivity;
- ❶ изменить метод CrimeHolder.onClick(...) так, чтобы он запускал CrimePager-Activity вместо CrimeActivity.

Создание CrimePagerActivity

Класс CrimePagerActivity будет субклассом FragmentActivity. Он создает экземпляр и управляет ViewPager.

Создайте новый класс с именем CrimePagerActivity. Назначьте его суперклассом FragmentActivity и создайте представление для активности.

Листинг 11.1. Создание ViewPager (CrimePagerActivity.java)

```

public class CrimePagerActivity extends FragmentActivity {

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

Файл макета еще не существует. Создайте новый файл макета в `res/layout/` и присвойте ему имя `activity_crime_pager`. Назначьте его корневым представлением `ViewPager` и присвойте ему атрибуты, показанные на рис. 11.3. Обратите внимание на необходимость использования полного имени пакета `ViewPager` (`android.support.v4.view.ViewPager`).

```

android.support.v4.view.ViewPager
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/activity_crime_pager_view_pager"
android:layout_width="match_parent"
android:layout_height="match_parent"

```

Рис. 11.3. Определение `ViewPager` в `CrimePagerActivity` (`activity_crime_pager.xml`)

Полное имя пакета используется при добавлении в файл макета, потому что класс `ViewPager` определен в библиотеке поддержки. В отличие от `Fragment`, класс `ViewPager` доступен *только* в библиотеке поддержки; в более поздних версиях SDK так и не появилось «стандартного» класса `ViewPager`.

ViewPager и PagerAdapter

Класс `ViewPager` в чем-то похож на `RecyclerView`. Чтобы класс `RecyclerView` мог выдавать представления, ему необходим экземпляр `Adapter`. Классу `ViewPager` необходим адаптер `PagerAdapter`.

Однако взаимодействие между `ViewPager` и `PagerAdapter` намного сложнее взаимодействия между `RecyclerView` и `Adapter`. К счастью, мы можем использовать `FragmentStatePagerAdapter` — субкласс `PagerAdapter`, который берет на себя многие технические подробности.

`FragmentStatePagerAdapter` сводит взаимодействие к двум простым методам: `getCount()` и `getItem(int)`. При вызове метода `getItem(int)` для позиции в массиве преступлений следует вернуть объект `CrimeFragment`, настроенный для вывода информации объекта в заданной позиции.

В классе `CrimePagerActivity` добавьте следующий код для назначения `PagerAdapter` класса `ViewPager` и реализации его методов `getCount()` и `getItem(int)`.

Листинг 11.2. Назначение `PagerAdapter` (`CrimePagerActivity.java`)

```

public class CrimePagerActivity extends FragmentActivity {
    private ViewPager mViewPager;
    private List<Crime> mCrimes;

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

```

```
mViewPager = (ViewPager) findViewById(R.id.activity_crime_pager_
    view_pager);

mCrimes = CrimeLab.get(this).getCrimes();
FragmentManager fragmentManager = getSupportFragmentManager();
mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
    @Override
    public Fragment getItem(int position) {
        Crime crime = mCrimes.get(position);
        return CrimeFragment.newInstance(crime.getId());
    }

    @Override
    public int getCount() {
        return mCrimes.size();
    }
});
}
```

Пройдемся по этому коду. После поиска ViewPager в представлении активности мы получаем от CrimeLab набор данных — контейнер List объектов Crime. Затем мы получаем экземпляр FragmentManager для активности.

На следующем шаге адаптером назначается безымянный экземпляр FragmentStatePagerAdapter. Для создания FragmentStatePagerAdapter необходим объект FragmentManager. Не забывайте, что FragmentStatePagerAdapter — ваш агент, управляющий взаимодействием с ViewPager. Чтобы ваш агент мог выполнить свою работу с фрагментами, возвращаемыми в getItem(int), он должен быть способен добавить их в активность. Вот почему ему необходим экземпляр FragmentManager.

(Что именно делает агент? Вкратце, он добавляет возвращаемые фрагменты в активность и помогает ViewPager идентифицировать представления фрагментов для их правильного размещения. Более подробная информация приведена в разделе «Для любознательных» в конце главы.)

Два метода PagerAdapter весьма просты. Метод getCount() возвращает текущее количество элементов в списке. Все существенное происходит в методе getItem(int). Он получает экземпляр Crime для заданной позиции в наборе данных, после чего использует его идентификатор для создания и возвращения правильно настроенного экземпляра CrimeFragment.

Интеграция CrimePagerActivity

Теперь можно переходить к удалению класса CrimeActivity и замене его классом CrimePagerActivity.

Начнем с добавления метода newInstance в CrimePagerActivity вместе с дополнением для идентификатора преступления.

Листинг 11.3. Создание `newIntent` (`CrimePagerActivity.java`)

```
public class CrimePagerActivity extends FragmentActivity {
    private static final String EXTRA_CRIME_ID =
        "com.bignerdranch.android.criminalintent.crime_id";

    private ViewPager mViewPager;
    private List<Crime> mCrimes;

    public static Intent newIntent(Context packageContext, UUID crimeId) {
        Intent intent = new Intent(packageContext, CrimePagerActivity.class);
        intent.putExtra(EXTRA_CRIME_ID, crimeId);
        return intent;
    }

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

        UUID crimeId = (UUID) getIntent()
            .getSerializableExtra(EXTRA_CRIME_ID);
        ---
    }
}
```

Теперь нужно сделать так, чтобы при выборе элемента списка в `CrimeListFragment` запускался экземпляр `CrimePagerActivity` вместо `CrimeActivity`.

Вернитесь к файлу `CrimeListFragment.java` и измените метод `CrimeHolder.onClick(...)`, чтобы он запускал `CrimePagerActivity`.

Листинг 11.4. Запуск активности (`CrimeListFragment.java`)

```
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ---
    @Override
    public void onClick(View v) {
        Intent intent = CrimeActivity.newIntent(getActivity(),
            mCrime.getId());
        Intent intent = CrimePagerActivity.newIntent(getActivity(),
            mCrime.getId());
        startActivity(intent);
    }
}
```

Также необходимо добавить `CrimePagerActivity` в манифест, чтобы ОС могла запустить эту активность. Пока манифест будет открыт, заодно удалите объявление `CrimeActivity`. Для этого достаточно заменить в манифесте `CrimeActivity` на `CrimePagerActivity`.

Листинг 11.5. Добавление CrimePagerActivity в манифест (AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>
  ...
  <application ...>
    ...

    <activity
      android:name=".CrimeActivity"
      android:name=".CrimePagerActivity"
      android:label="@string/app_name" >
    </activity>

    ...
  </application>
</manifest>
```

Наконец, чтобы не загромождать проект, удалите CrimeActivity.java в окне инструментов Project.

Запустите приложение CriminalIntent. Нажмите на строке Crime #0, чтобы просмотреть подробную информацию. Проведите по экрану влево или вправо, чтобы просмотреть другие элементы списка. Обратите внимание: переключение страниц происходит плавно и без задержек. По умолчанию ViewPager загружает элемент, находящийся на экране, а также по одному соседнему элементу в каждом направлении, чтобы отклик на жест прокрутки был немедленным. Количество загружаемых соседних страниц можно настроить вызовом `setOffscreenPageLimit(int)`.

Однако с ViewPager еще не все идеально. Вернитесь к списку при помощи кнопки Вски и щелкните на другом элементе. Вы снова увидите информацию первого элемента — вместо того, который был запрошен.

По умолчанию ViewPager отображает в своем экземпляре PagerAdapter первый элемент. Чтобы вместо него отображался элемент, выбранный пользователем, назначьте текущим элементом ViewPager элемент с указанным индексом.

В конце CrimePagerActivity.onCreate(...) найдите индекс отображаемого преступления; для этого переберите и проверьте идентификаторы всех преступлений. Когда вы найдете экземпляр Crime, у которого поле mId совпадает с crimeId в дополнении интента, измените текущий элемент по индексу найденного объекта Crime.

Листинг 11.6. Назначение исходного элемента (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        ...

        FragmentManager fragmentManager = getSupportFragmentManager();
        mViewPager.setAdapter(new FragmentStatePagerAdapter(fragmentManager) {
            ...
        });
    }
}
```

```

for (int i = 0; i < mCrimes.size(); i++) {
    if (mCrimes.get(i).getId().equals(crimeld)) {
        mViewPager.setCurrentItem(i);
        break;
    }
}
}
}
}

```

Запустите приложение CriminalIntent. При выборе любого элемента списка должна отображаться подробная информация правильного объекта Crime. Вот и все! Теперь наш экземпляр ViewPager полностью готов к работе.

FragmentManager и FragmentPagerAdapter

Существует еще один тип PagerAdapter, который можно использовать в приложениях; он называется FragmentPagerAdapter.

FragmentPagerAdapter используется точно так же, как FragmentStatePagerAdapter, и отличается от него только способом выгрузки неиспользуемых фрагментов (рис. 11.4).

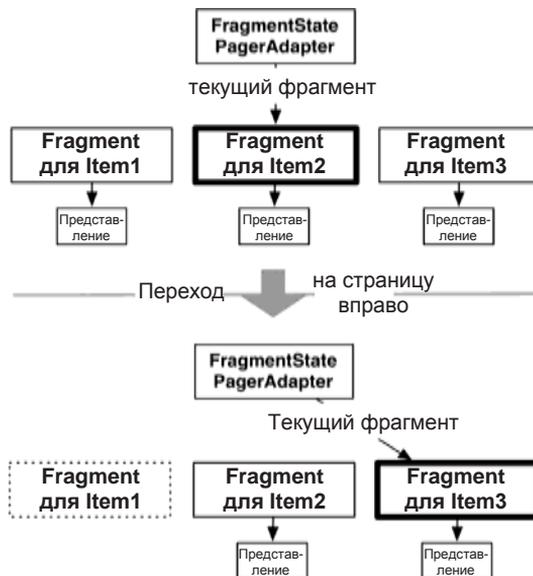


Рис. 11.4. Управление фрагментами `FragmentStatePagerAdapter`

При использовании класса `FragmentStatePagerAdapter` неиспользуемый фрагмент уничтожается. Происходит закрепление транзакции для полного удаления фрагмента из объекта `FragmentManager` активности. Наличие «состояния»

у `FragmentManager` определяется тем фактом, что экземпляр при уничтожении сохраняет объект `Bundle` вашего фрагмента в методе `onSaveInstanceState(Bundle)`. Когда пользователь возвращается обратно, новый фрагмент восстанавливается по состоянию этого экземпляра.

С другой стороны, `FragmentPagerAdapter` ничего подобного не делает. Когда фрагмент становится ненужным, `FragmentPagerAdapter` вызывает для транзакции `detach(Fragment)` вместо `remove(Fragment)`. Представление фрагмента при этом уничтожается, но экземпляр фрагмента продолжает существовать в `FragmentManager`. Таким образом, фрагменты, созданные `FragmentPagerAdapter`, никогда не уничтожаются (рис. 11.5).

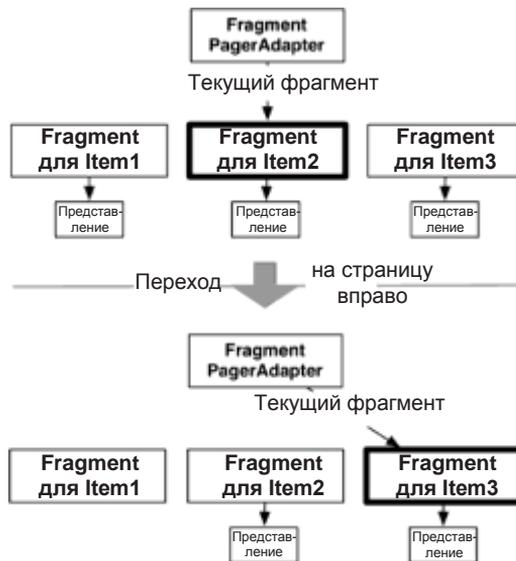


Рис. 11.5. Управление фрагментами `FragmentPagerAdapter`

Выбор используемого адаптера зависит от приложения. Как правило, `FragmentManager` более экономно расходует память. Приложение `CriminalIntent` выводит список, который со временем может стать достаточно длинным, причем к каждому элементу списка может прилагаться фотография. Хранить всю эту информацию в памяти нежелательно, поэтому мы используем `FragmentManager`.

С другой стороны, если интерфейс содержит небольшое фиксированное количество фрагментов, использование `FragmentManager` безопасно и уместно. Самый характерный пример такого рода — интерфейс со вкладками. Некоторые детализированные представления не помещаются на одном экране, поэтому отображаемая информация распределяется между несколькими вкладками. Добавление `ViewPager` с перебором вкладок делает этот интерфейс интуитивным. Хранение фрагментов в памяти упрощает управление кодом контроллера, а поскольку

этот стиль интерфейса обычно использует всего два или три фрагмента на активность, проблемы с нехваткой памяти крайне маловероятны.

Для любознательных: как работает ViewPager

Классы ViewPager и PagerAdapter незаметно выполняют большую часть рутинной работы. В этом разделе приведена более подробная информация о том, что при этом происходит.

Прежде чем мы перейдем к обсуждению, предупреждаем: в большинстве случаев понимать все технические подробности не обязательно.

Но если вы захотите реализовать интерфейс PagerAdapter самостоятельно, вы должны знать, чем отношения ViewPager-PagerAdapter отличаются от обычных отношений RecyclerView-Adapter.

Когда может возникнуть необходимость в самостоятельной реализации интерфейса PagerAdapter? Когда в ViewPager должны размещаться не фрагменты, а нечто иное. Например, если вы захотите разместить в ViewPager обычные представления View, скажем графические поля, вы реализуете интерфейс PagerAdapter.

Почему ViewPager, а не RecyclerView?

Использование RecyclerView в данном случае потребует большого объема работы, потому что вы не сможете использовать существующие экземпляры Fragment. Adapter ожидает, что вы сможете предоставить View мгновенно. Но когда будет создано представление вашего фрагмента, решает FragmentManager, а не вы. Таким образом, когда RecyclerView обратится к Adapter за представлением вашего фрагмента, вы не сможете создать фрагмент и немедленно выдать его представление.

Именно по этой причине и существует класс ViewPager. Вместо Adapter он использует класс с именем PagerAdapter. Этот класс сложнее Adapter, потому что он выполняет больший объем работы по управлению представлениями. Ниже кратко перечислены основные различия.

Вместо метода `onBindViewHolder(...)`, возвращающего ViewHolder с соответствующим представлением, PagerAdapter содержит следующие методы:

```
public Object instantiateItem(ViewGroup container, int position)
public void destroyItem(ViewGroup container, int position, Object object)
public abstract boolean isViewFromObject(View view, Object object)
```

Метод `pagerAdapter.instantiateItem(ViewGroup, int)` приказывает адаптеру создать представление элемента списка для заданной позиции и добавить его в контейнер ViewGroup; метод `destroyItem(ViewGroup, int, Object)` приказывает уничтожить этот элемент. Обратите внимание: метод `instantiateItem(ViewGroup, int)` не приказывает создать представление *немедленно*. PagerAdapter может создать представление в любой момент в будущем.

После того как представление было создано, ViewPager в какой-то момент замечает его. Чтобы понять, к какому элементу списка оно относится, ViewPager вы-

зывает метод `isViewFromObject(View, Object)`. Параметр `Object` содержит объект, полученный при вызове `instantiateltem(ViewGroup, int)`. Таким образом, если `ViewPager` вызывает `instantiateltem(ViewGroup, 5)` и получает объект `A`, вызов `isViewFromObject(View, A)` должен вернуть `true`, если переданный экземпляр `View` относится к элементу `5`, и `false` в противном случае.

Этот процесс достаточно сложен для `ViewPager`, но не для класса `PagerAdapter`, который должен уметь только создавать представления, уничтожать представления и определять, к какому объекту относится представление. Менее жесткие требования позволяют реализации `PagerAdapter` создавать и добавлять новый фрагмент в `instantiateltem(ViewGroup, int)` и возвращать фрагмент как отслеживаемый экземпляр `Object`. При этом `isViewFromObject(View, Object)` выглядит примерно так:

```
@Override
public boolean isViewFromObject(View view, Object object) {
    return ((Fragment)object).getView() == view;
}
```

Реализовывать переопределения `PagerAdapter` каждый раз, когда потребуется использовать `ViewPager`, было бы слишком утомительно. Хорошо, что у нас есть `FragmentPagerAdapter` и `FragmentStatePagerAdapter`.

Для любознательных: формирование макетов представлений в коде

В этой книге макеты представлений определяются исключительно в XML-файлах макетов. Также возможно создавать макеты представлений в коде.

Собственно, `ViewPager` можно было бы определить полностью в коде вообще без файла макета:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ViewPager viewPager = new ViewPager(this);
    setContentView(viewPager);

    ...
}
```

Создание представления не требует никакого волшебства; просто вызовите его конструктор и передайте параметр `Context`. Всю иерархию представлений можно создать на программном уровне, вместо того чтобы использовать файлы макетов. Тем не менее строить представления в коде не рекомендуется, потому что файлы макетов предоставляют ряд преимуществ.

Первое преимущество файлов макетов заключается в том, что они обеспечивают четкое разделение между контроллером и объектами представлений в приложении. Представление существует в XML, контроллер существует в коде Java. Это разделение упрощает сопровождение кода за счет ограничения объема изменений в контроллере при изменении представления, и наоборот.

Другое преимущество определения представлений в XML заключается в том, что система уточнения ресурсов Android позволяет автоматически выбрать версию файла XML в зависимости от свойств устройства.

Как было показано в главе 3, эта система позволяет легко сменить файл макета в зависимости от ориентации устройства (а также других параметров конфигурации).

Есть ли у файлов макетов какие-либо недостатки? Пожалуй, хлопоты с созданием файла XML и его заполнением. Если вы создаете всего одно представление, иногда эти хлопоты могут показаться излишними.

В остальном сколько-нибудь серьезных недостатков нет — группа разработки Android никогда не рекомендовала строить иерархии представлений на программном уровне, даже в прежние времена, когда разработчикам приходилось беспокоиться о быстродействии больше, чем сейчас. Даже если вам требуется нечто настолько мелкое, как представление с идентификатором (что часто требуется для представлений, созданных на программном уровне), проще создать файл макета.

12 Диалоговые окна

Диалоговые окна требуют от пользователя внимания и ввода данных. Обычно они используются для принятия решений или отображения важной информации. В этой главе мы добавим диалоговое окно, в котором пользователь может изменить дату преступления.

При нажатии кнопки даты в CrimeFragment открывается диалоговое окно, показанное на рис. 12.1.



Рис. 12.1. Диалоговое окно для выбора даты

Диалоговое окно на рис. 12.1 является экземпляром `AlertDialog` — subclasses `Dialog`. Именно этот многоцелевой subclass `Dialog` вы будете чаще всего использовать в своих программах.

В версии Lollipop диалоговые окна прошли визуальную переработку. `AlertDialog` в Lollipop автоматически используют новый стиль. В более ранних версиях Android окна `AlertDialog` возвращаются к старому стилю, изображенному слева на рис. 12.2.

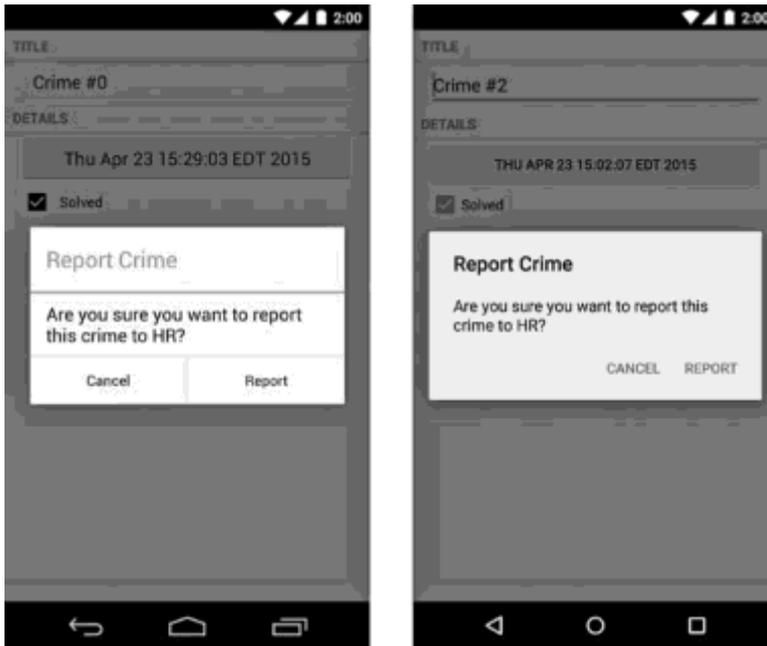


Рис. 12.2. Старый и новый визуальный стиль

Конечно, вместо доисторических диалоговых окон было бы лучше всегда отображать окна в новом стиле независимо от версии Android на устройстве пользователя. Этого можно добиться при помощи библиотеки `AppCompat`.

`AppCompat` — библиотека совместимости, разработанная компанией Google, которая реализует некоторые возможности новых версий Android на старых устройствах. В этой главе мы используем библиотеку `AppCompat` для создания стабильного оформления диалоговых окон во всех поддерживаемых версиях Android. В главах 13 и 20 также будут использоваться другие возможности библиотеки `AppCompat`.

Библиотека `AppCompat`

Чтобы использовать библиотеку `AppCompat`, сначала необходимо добавить ее в число зависимостей. Возможно, она там уже есть; это зависит от того, как создавался проект.

Откройте окно Project Structure (File ► Project Structure...), выберите модуль app и щелкните на вкладке Dependencies. Если библиотека AppCompat не входит в список, добавьте ее — щелкните на кнопке + и выберите зависимость appcompat-v7 в списке (рис. 12.3).



Рис. 12.3. Выбор зависимости AppCompat

Библиотека AppCompat содержит собственный класс AlertDialog, который вы будете использовать в своих программах. Эта версия AlertDialog очень похожа на ту, которая включена в ОС Android. Чтобы использовать именно ту версию, которая вам нужна, необходимо импортировать правильную версию AlertDialog. Мы будем использовать `android.support.v7.app.AlertDialog`.

Создание DialogFragment

При использовании объекта AlertDialog обычно удобно упаковать его в экземпляр DialogFragment — subclasses Fragment. Вообще говоря, экземпляр AlertDialog может отображаться и без DialogFragment, но Android так поступать не рекомендует. Управление диалоговым окном из FragmentManager открывает больше возможностей для его отображения.

Кроме того, «минимальный» экземпляр AlertDialog исчезнет при повороте устройства. С другой стороны, если экземпляр AlertDialog упакован во фрагмент, после поворота диалоговое окно будет создано заново и появится на экране.

Для приложения CriminalIntent мы создадим subclass DialogFragment с именем DatePickerFragment. В коде DatePickerFragment создается и настраивается экземпляр AlertDialog, отображающий виджет DatePicker. В качестве хоста DatePickerFragment используется экземпляр CrimePagerActivity.

На рис. 12.4 изображена схема этих отношений.

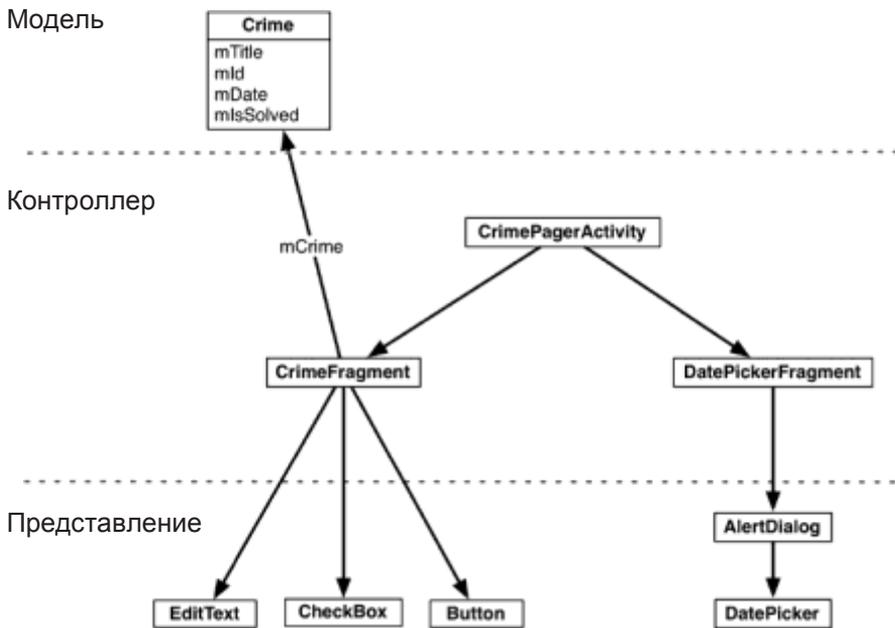


Рис. 12.4. Диаграмма объектов для двух фрагментов с хостом CrimePagerActivity

Наши первоочередные задачи:

- ① создание класса DatePickerFragment;
- ① построение AlertDialog;
- ① вывод диалогового окна на экран с использованием fragmentManager.

Позднее в этой главе мы подключим виджет DatePicker и организуем передачу необходимых данных между CrimeFragment и DatePickerFragment.

Прежде чем браться за работу, добавьте строковый ресурс (листинг 12.1).

Листинг 12.1. Добавление строки заголовка диалогового окна (values/strings.xml)

```
<resources>
...
<string name="crime_solved_label">Solved</string>
<string name="date_picker_title">Date of crime:</string>
</resources>
```

Создайте новый класс с именем DatePickerFragment и назначьте его суперклассом DialogFragment. Обязательно выберите версию DialogFragment из библиотеки поддержки: android.support.v4.app.DialogFragment.

Класс DialogFragment содержит следующий метод:

```
public Dialog onCreateDialog(Bundle savedInstanceState)
```

Экземпляр `FragmentManager` активности-хоста вызывает этот метод в процессе вывода `DialogFragment` на экран.

Добавьте в файл `DatePickerFragment.java` реализацию `onCreateDialog(...)`, которая создает `AlertDialog` с заголовком и одной кнопкой ОК. (Виджет `DatePicker` мы добавим позднее.)

Проследите за тем, чтобы импортировалась версия `AlertDialog` из `AppCompat: android.support.v7.app.AlertDialog`.

Листинг 12.2. Создание `DialogFragment` (`DatePickerFragment.java`)

```
public class DatePickerFragment extends DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        return new AlertDialog.Builder(getActivity())
            .setTitle(R.string.date_picker_title)
            .setPositiveButton(android.R.string.ok, null)
            .create();
    }
}
```

В этой реализации используется класс `AlertDialog.Builder`, предоставляющий динамичный интерфейс для конструирования экземпляров `AlertDialog`.

Сначала мы передаем объект `Context` конструктору `AlertDialog.Builder`, который возвращает экземпляр `AlertDialog.Builder`.

Затем вызываются два метода `AlertDialog.Builder` для настройки диалогового окна:

```
public AlertDialog.Builder setTitle(int titleId)
public AlertDialog.Builder setPositiveButton(int textId,
    DialogInterface.OnClickListener listener)
```

Метод `setPositiveButton(...)` получает строковый ресурс и объект, реализующий `DialogInterface.OnClickListener`. В листинге 12.2 передается константа `Android` для кнопки ОК и `null` вместо слушателя. Слушатель будет реализован позднее в этой главе.

Положительная кнопка (`Positive`) нажимается пользователем для подтверждения информации в диалоговом окне. В `AlertDialog` также можно добавить еще две кнопки: отрицательную (`Negative`) и нейтральную (`Neutral`). Эти обозначения определяют позицию кнопок в диалоговом окне (если их несколько).

Построение диалогового окна завершается вызовом `AlertDialog.Builder.create()`, который возвращает настроенный экземпляр `AlertDialog`.

Этим возможности `AlertDialog` и `AlertDialog.Builder` не исчерпываются; подробности достаточно хорошо изложены в документации разработчика. А пока давайте перейдем к механике вывода диалогового окна на экран.

Отображение DialogFragment

Как и все фрагменты, экземпляры DialogFragment находятся под управлением экземпляра FragmentManager активности-хоста.

Для добавления экземпляра DialogFragment в FragmentManager и вывода его на экран используются следующие методы экземпляра фрагмента:

```
public void show(FragmentManager manager, String tag)
public void show(FragmentTransaction transaction, String tag)
```

Строковый параметр однозначно идентифицирует DialogFragment в списке FragmentManager. Выбор версии (с FragmentManager или FragmentTransaction) зависит только от вас: если передать FragmentTransaction, за создание и закрепление транзакции отвечаете вы. При передаче FragmentManager транзакция автоматически создается и закрепляется для вас.

В нашем примере передается FragmentManager.

Добавьте в CrimeFragment константу для метки DatePickerFragment. Затем в методе onCreateView(...) удалите код, блокирующий кнопку даты, и назначьте слушателя View.OnClickListener, который отображает DatePickerFragment при нажатии кнопки даты.

Листинг 12.3. Отображение DialogFragment (CrimeFragment.java)

```
public class CrimeFragment extends Fragment {
    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...

        mDateButton = (Button) v.findViewById(R.id.crime_date);
        mDateButton.setText(mCrime.getDate().toString());
        mDateButton.setEnabled(false);
        mDateButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = new DatePickerFragment();
                dialog.show(manager, DIALOG_DATE);
            }
        });

        mSolvedCheckBox = (CheckBox) v.findViewById(R.id.crime_solved);
        ...
    }
}
```

```
        return v;  
    }  
    ---  
}
```

Запустите приложение CriminalIntent и нажмите кнопку даты, чтобы диалоговое окно появилось на экране (рис. 12.5).



Рис. 12.5. AlertDialog с заголовком и кнопкой

Назначение содержимого диалогового окна

Далее мы включим в AlertDialog виджет DatePicker при помощи метода AlertDialog.Builder:

```
public AlertDialog.Builder setView(View view)
```

Метод настраивает диалоговое окно для отображения переданного объекта View между заголовком и кнопкой(-ами).

В окне инструментов Project создайте новый файл макета с именем dialog_date.xml и назначьте его корневым элементом DatePicker. Макет будет состоять из одного объекта View (DatePicker), который мы заполним и передадим setView(...).

Настройте макет DatePicker так, как показано на рис. 12.6.

```

DatePicker
xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/dialog_date_date_picker"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:calendarViewShown="false"

```

Рис. 12.6. Макет DatePicker (layout/dialog_date.xml)

В методе `DatePickerFragment.onCreateDialog(...)` заполните представление и назначьте его диалоговому окну.

Листинг 12.4. Включение DatePicker в AlertDialog (`DatePickerFragment.java`)

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    View v = LayoutInflater.from(getActivity())
    .inflate(R.layout.dialog_date, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}

```

Запустите приложение `CriminalIntent`. Нажмите кнопку даты и убедитесь в том, что в диалоговом окне теперь отображается `DatePicker`. На устройствах с версией Lollipop отображается календарный виджет (рис. 12.7).

Календарная версия на рис. 12.7 появилась вместе с концепцией материального оформления (`Material design`). Эта версия виджета `DatePicker` игнорирует атрибут `calendarViewShown`, заданный в макете. Но на устройствах с более старыми версиями Android вы увидите старую «дисковую» версию `DatePicker`, которая учитывает значение этого атрибута (рис. 12.8).

Обе версии работают. Впрочем, новая выглядит лучше.

Почему мы возмемся с определением и заполнением макета, когда объект `DatePicker` можно было бы создать в коде так, как показано ниже?

```

@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    DatePicker dp = new DatePicker(getActivity());

    return new AlertDialog.Builder(getActivity())
        .setView(dp)
        ...
        .create();
}

```



Рис. 12.7. DatePicker в Lollipop



Рис. 12.8. AlertDialog с DatePicker

Использование макета упрощает изменения в случае изменения его содержимого. Предположим, вы захотели, чтобы рядом с DatePicker в диалоговом окне отображался виджет TimePicker. При использовании заполнения можно просто обновить файл макета, и новое представление появится на экране.

Также обратите внимание на то, что дата, выбранная в DatePicker, автоматически сохраняется при поворотах. (Чтобы убедиться в этом, откройте диалоговое окно, выберите новую дату и нажмите Fn+Control+F12/Ctrl+F1.) Как это происходит? Помните, что представления могут сохранять состояние между изменениями конфигурации, но только в том случае, если у них есть атрибут id. При создании DatePicker в dialog_date.xml вы также приказали инструментарию построения программы сгенерировать уникальный идентификатор для этого виджета DatePicker. Если DatePicker создается в коде, то для сохранения состояния вам придется назначить идентификатор DatePicker на программном уровне.

Итак, наше диалоговое окно успешно отображается. В следующем разделе мы свяжем его с полем даты Crime и позаботимся о том, чтобы пользователь мог вводить данные.

Передача данных между фрагментами

Мы передавали данные между двумя активностями; мы передавали данные между двумя фрагментными активностями. Теперь нужно передать данные между дву-

фрагментами, хостом которых является одна активность, — CrimeFragment и DatePickerFragment (рис. 12.9).

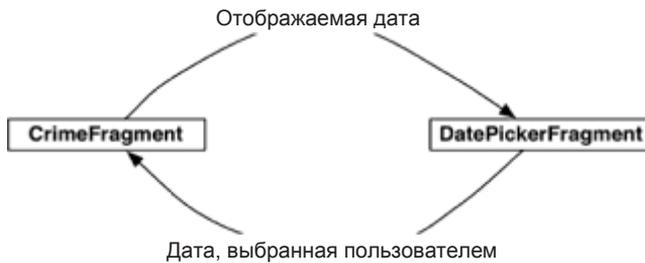


Рис. 12.9. Взаимодействие между CrimeFragment и DatePickerFragment

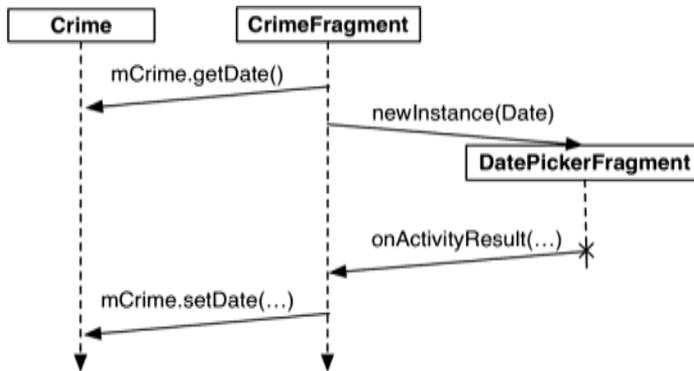


Рис. 12.10. Последовательность событий взаимодействия между CrimeFragment и DatePickerFragment

Чтобы передать дату преступления DatePickerFragment, мы напишем метод `newInstance(Date)` и сделаем объект `Date` аргументом фрагмента.

Чтобы вернуть новую дату фрагменту CrimeFragment для обновления уровня модели и его собственного представления, мы упакуем ее как дополнительный объект `Intent` и передадим этот объект `Intent` в вызове `CrimeFragment.onActivityResult(...)` (рис. 12.10).

Вызов `Fragment.onActivityResult(...)` может показаться странным — с учетом того, что активность-хост не получает вызова `Activity.onActivityResult(...)` в этом взаимодействии. Но как будет показано позднее в этой главе, использование `onActivityResult(...)` для передачи данных от одного фрагмента к другому не только работает, но и улучшает гибкость отображения фрагмента диалогового окна.

Передача данных DatePickerFragment

Чтобы получить данные в DatePickerFragment, мы сохраним дату в пакете аргументов DatePickerFragment, где DatePickerFragment сможет обратиться к ней.

Создание аргументов фрагмента и присваивание им значений обычно выполняется в методе `newInstance()`, заменяющем конструктор фрагмента. Добавьте в файл `DatePickerFragment.java` метод `newInstance(Date)`.

Листинг 12.5. Добавление метода `newInstance(Date)` (`DatePickerFragment.java`)

```
public class DatePickerFragment extends DialogFragment {

    private static final String ARG_DATE = "date";

    private DatePicker mDatePicker;

    public static DatePickerFragment newInstance(Date date) {
        Bundle args = new Bundle();
        args.putSerializable(ARG_DATE, date);

        DatePickerFragment fragment = new DatePickerFragment();
        fragment.setArguments(args);
        return fragment;
    }

    ...
}
```

В классе `CrimeFragment` удалите вызов конструктора `DatePickerFragment` и замените его вызовом `DatePickerFragment.newInstance(Date)`.

Листинг 12.6. Добавление вызова `newInstance()` (`CrimeFragment.java`)

```
@Override
public View onCreateView(LayoutInflater inflater,
    ViewGroup parent, Bundle savedInstanceState) {
    ...

    mDateButton = (Button)v.findViewById(R.id.crime_date);
    mDateButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            FragmentManager manager = getActivity()
                .getSupportFragmentManager();
            DatePickerFragment dialog = new DatePickerFragment();
            DatePickerFragment dialog = DatePickerFragment
                .newInstance(mCrime.getDate());
            dialog.show(manager, DIALOG_DATE);
        }
    });

    return v;
}
```

Экземпляр `DatePickerFragment` должен инициализировать `DatePicker` по информации, хранящейся в `Date`. Однако для инициализации `DatePicker` необхо-

димо иметь целочисленные значения месяца, дня и года. Объект `Date` больше напоминает временную метку и не может предоставить нужные целые значения напрямую.

Чтобы получить нужные значения, следует создать объект `Calendar` и использовать `Date` для определения его конфигурации. После этого вы сможете получить нужную информацию из `Calendar`.

В методе `onCreateDialog(...)` получите объект `Date` из аргументов и используйте его с `Calendar` для инициализации `DatePicker`.

Листинг 12.7. Извлечение даты и инициализация `DatePicker`
(`DatePickerFragment.java`)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    Date date = (Date) getArguments().getSerializable(ARG_DATE);

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    View v = LayoutInflater.from(getActivity())
        .inflate(R.layout.dialog_date, null);

    mDatePicker = (DatePicker) v.findViewById(R.id.dialog_date_date_picker);
    mDatePicker.init(year, month, day, null);

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null)
        .create();
}
```

Теперь `CrimeFragment` успешно сообщает `DatePickerFragment`, какую дату следует отобразить. Вы можете запустить приложение `CriminalIntent` и убедиться в том, что все работает так же, как прежде.

Возвращение данных `CrimeFragment`

Чтобы экземпляр `CrimeFragment` получал данные от `DatePickerFragment`, нам необходимо каким-то образом отслеживать отношения между двумя фрагментами.

С активностями вы вызываете `startActivityForResult(...)`, а `ActivityManager` отслеживает отношения между родительской и дочерней активностью. Когда дочерняя активность прекращает существование, `ActivityManager` знает, какая активность должна получить результат.

Назначение целевого фрагмента

Для создания аналогичной связи можно назначить `CrimeFragment` *целевым фрагментом* (target fragment) для `DatePickerFragment`. Эта связь будет автоматически восстановлена после того, как и `CrimeFragment`, и `DatePickerFragment` уничтожаются и заново создаются ОС. Для этого вызывается следующий метод `Fragment`:

```
public void setTargetFragment(Fragment fragment, int requestCode)
```

Метод получает фрагмент, который станет целевым, и код запроса, аналогичный передаваемому `startActivityResult(...)`. По коду запроса целевой фрагмент позднее может определить, какой фрагмент возвращает информацию.

`FragmentManager` сохраняет целевой фрагмент и код запроса. Чтобы получить их, вызовите `getTargetFragment()` и `getTargetrequestCode()` для фрагмента, назначившего целевой фрагмент.

В файле `CrimeFragment.java` создайте константу для кода запроса, а затем назначьте `CrimeFragment` целевым фрагментом экземпляра `DatePickerFragment`.

Листинг 12.8. Назначение целевого фрагмента (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {

    private static final String ARG_CRIME_ID = "crime_id";
    private static final String DIALOG_DATE = "DialogDate";

    private static final int REQUEST_DATE = 0;
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup parent,
        Bundle savedInstanceState) {
        ...

        mDateButton.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                FragmentManager manager = getFragmentManager();
                DatePickerFragment dialog = DatePickerFragment
                    .newInstance(mCrime.getDate());
                dialog.setTargetFragment(CrimeFragment.this, REQUEST_DATE);
                dialog.show(manager, DIALOG_DATE);
            }
        });

        return v;
    }
    ...
}
```

Передача данных целевому фрагменту

Итак, связь между `CrimeFragment` и `DatePickerFragment` создана, и теперь нужно вернуть дату `CrimeFragment`. Дата будет включена в объект `Intent` как дополнение.

Какой метод будет использоваться для передачи интента целевому фрагменту? Как ни странно, DatePickerFragment передаст его при вызове CrimeFragment.onActivityResult(int, int, Intent).

Метод Activity.onActivityResult(...) вызывается ActivityManager для родительской активности при уничтожении дочерней активности. При работе с активностями вы не вызываете Activity.onActivityResult(...) самостоятельно; это делает ActivityManager. После того как активность получит вызов, экземпляр FragmentManager активности вызывает Fragment.onActivityResult(...) для соответствующего фрагмента.

Если хостом двух фрагментов является одна активность, то для возвращения данных можно воспользоваться методом Fragment.onActivityResult(...) и вызывать его непосредственно для целевого фрагмента. Он содержит все необходимое:

- ① код запроса, соответствующий коду, переданному setTargetFragment(...), по которому целевой фрагмент узнает, кто возвращает результат;
- ① код результата для определения выполняемого действия;
- ① экземпляра Intent, который может содержать дополнительные данные.

В классе DatePickerFragment создайте закрытый метод, который создает интент, помещает в него дату как дополнение, а затем вызывает CrimeFragment.onActivityResult(...).

Листинг 12.9. Обратный вызов целевого фрагмента (DatePickerFragment.java)

```
public class DatePickerFragment extends DialogFragment {

    public static final String EXTRA_DATE =
        "com.bignerdranch.android.criminalintent.date";

    private static final String ARG_DATE = "date";
    ...

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        ...
    }

    private void sendResult(int resultCode, Date date) {
        if (getTargetFragment() == null) {
            return;
        }

        Intent intent = new Intent();
        intent.putExtra(EXTRA_DATE, date);

        getTargetFragment()
            .onActivityResult(getTargetRequestCode(), resultCode, intent);
    }
}
```

Пришло время воспользоваться новым методом `sendResult`. Когда пользователь нажимает кнопку положительного ответа в диалоговом окне, приложение должно получить дату из `DatePicker` и отправить результат `CrimeFragment`. В коде `onCreateDialog(...)` замените параметр `null` вызова `setPositiveButton(...)` реализацией `DialogInterface.OnClickListener`, которая возвращает выбранную дату и вызывает `sendResult`.

Листинг 12.10. Передача информации (`DatePickerFragment.java`)

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    ...

    return new AlertDialog.Builder(getActivity())
        .setView(v)
        .setTitle(R.string.date_picker_title)
        .setPositiveButton(android.R.string.ok, null);
        .setPositiveButton(android.R.string.ok,
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog, int which) {
                    int year = mDatePicker.getYear();
                    int month = mDatePicker.getMonth();
                    int day = mDatePicker.getDayOfMonth();
                    Date date = new GregorianCalendar(year, month, day).
                        getTime();
                    sendResult(Activity.RESULT_OK, date);
                }
            })
        .create();
}
```

В классе `CrimeFragment` переопределите метод `onActivityResult(...)`, чтобы он возвращал дополнение, задавал дату в `Crime` и обновлял текст кнопки даты.

Листинг 12.11. Реакция на получение данных от диалогового окна (`CrimeFragment.java`)

```
public class CrimeFragment extends Fragment {
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (resultCode != Activity.RESULT_OK) {
            return;
        }
    }
}
```

```

    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        mDateButton.setText(mCrime.getDate().toString());
    }
}
}

```

Код, задающий текст кнопки, идентичен коду из `onCreateView(...)`. Чтобы избежать задания текста в двух местах, мы инкапсулируем этот код в закрытом методе `updateDate()`, а затем вызовем его в `onCreateView(...)` и `onActivityResult(...)`.

Вы можете сделать это вручную или поручить работу Android Studio. Выделите всю строку кода, которая задает текст `mDateButton`, щелкните на ней правой кнопкой мыши и выберите команду `Refactor` ▶ `Extract` ▶ `Method...` (рис. 12.11).

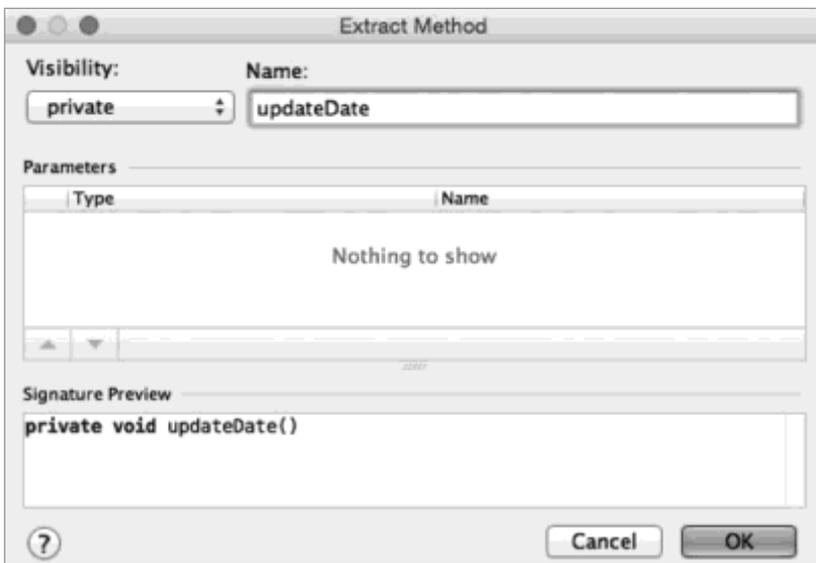


Рис. 12.11. Извлечение метода в Android Studio

Выберите закрытый уровень видимости метода и введите имя `updateDate`. Щелкните на кнопке `OK`; среда Android Studio сообщает, что ей удалось найти еще одно место, в котором использовалась эта строка кода. Щелкните на кнопке `Yes`, чтобы разрешить Android Studio обновить второе вхождение. Убедитесь в том, что код был выделен в метод `updateDate` (листинг 12.12).

Листинг 12.12. Выделение кода в метод `updateDate()` (`CrimeFragment.java`)

```

public class CrimeFragment extends Fragment {
    ...
    @Override

```

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    View v = inflater.inflate(R.layout.fragment_crime, container, false);
    ...

    mDateButton = (Button) v.findViewById(R.id.crime_date);
    updateDate();
    ...
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (resultCode != Activity.RESULT_OK) {
        return;
    }

    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
        updateDate();
    }
}

private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}
}
```

Круг замкнулся — данные передаются туда и обратно.

Запустите приложение `CriminalIntent` и убедитесь в том, что вы действительно можете управлять датой. Измените дату `Crime`; новая дата должна появиться в представлении `CrimeFragment`. Вернитесь к списку преступлений, проверьте дату `Crime` и убедитесь в том, что уровень модели действительно обновлен.

Больше гибкости в представлении DialogFragment

Использование `onActivityResult(...)` для возвращения данных целевому фрагменту особенно удобно, когда ваше приложение получает много данных от пользователя и нуждается в большем пространстве для их ввода. При этом приложение должно хорошо работать на телефонах и планшетах.

На экране телефона свободного места не так много, поэтому вы, скорее всего, используете для ввода данных активность с полноэкранным фрагментом. Дочерняя активность будет запускаться вызовом `startActivityForResult()` из фрагмента родительской активности. При уничтожении дочерней активности родительская активность будет получать вызов `onActivityResult(...)`, который будет перенаправляться фрагменту, запустившему дочернюю активность (рис. 12.12).

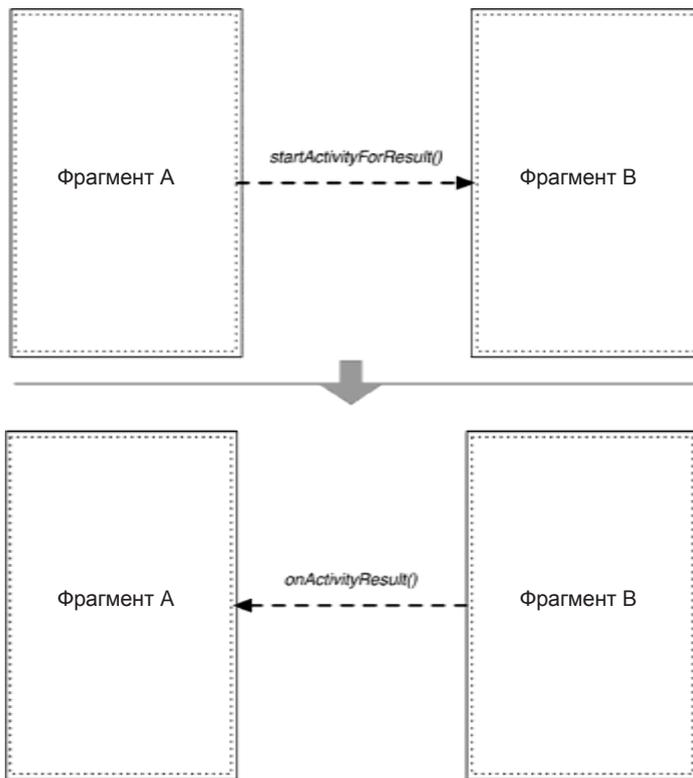


Рис. 12.12. Взаимодействие между активностями на телефонах

На планшетах, где экранного пространства больше, часто бывает лучше отобразить `DialogFragment` для ввода тех же данных. В таком случае вы задаете целевой фрагмент и вызываете `show(...)` для фрагмента диалогового окна. При закрытии фрагмент диалогового окна вызывает для своей цели `onActivityResult(...)`.

Метод `onActivityResult(...)` фрагмента будет вызываться всегда, независимо от того, запустил ли фрагмент активность или отобразил диалоговое окно. Следовательно, мы можем использовать один код для разных вариантов представления информации.

Когда один код используется и для полноэкранного, и для диалогового фрагмента, для подготовки вывода в обоих случаях вместо `onCreateDialog(...)` можно переопределить `DialogFragment.onCreateView(...)`.

Упражнение. Новые диалоговые окна

Напишите еще один диалоговый фрагмент `TimePickerFragment` для выбора времени преступления. Используйте виджет `TimePicker`, добавьте в `CrimeFragment` еще одну кнопку для отображения `TimePickerFragment`.

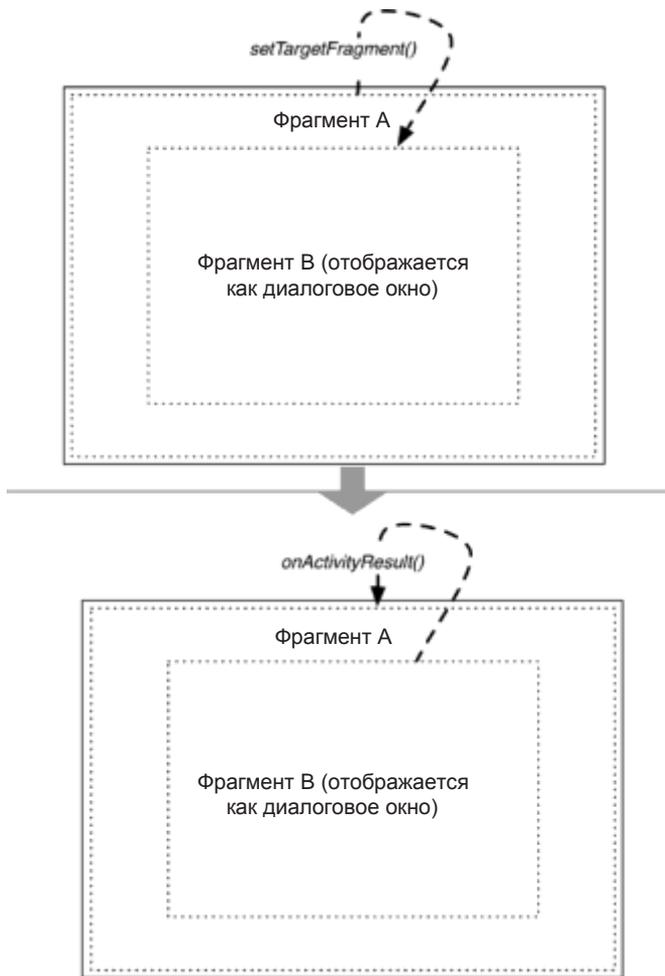


Рис. 12.13. Взаимодействие между фрагментами на планшетах

Упражнение. DialogFragment

Попробуйте справиться с более сложной задачей — изменением представления `DatePickerFragment`.

Первая часть упражнения — предоставить представление `DatePickerFragment` с переопределением `onCreateView` вместо `onCreateDialog`. При таком способе создания `DialogFragment` не будет отображаться со встроенными областями заголовка и кнопок в верхней и нижней части диалогового окна. Вам придется самостоятельно создать кнопку ОК в `dialog_date.xml`.

После того как представление `DatePickerFragment` будет создано в `onCreateView`, вы можете отобразить фрагмент `DatePickerFragment` как диалоговое окно или встроить его в активность. Во второй части упражнения создайте новый субкласс `SingleFragmentActivity` и сделайте эту активность хостом для `DatePickerFragment`.

При таком представлении `DatePickerFragment` вы будете использовать механизм `startActivityForResult` для возвращения даты `CrimeFragment`. В `DatePickerFragment`, если целевой фрагмент не существует, используйте метод `setResult(int, intent)` активности-хоста для возвращения даты фрагменту.

В последней части этого упражнения измените приложение `CriminalIntent` так, чтобы фрагмент `DatePickerFragment` отображался как полноэкранный фрагмент при запуске на телефоне. Если приложение запущено на планшете, `DatePickerFragment` отображается как диалоговое окно. Возможно, вам стоит забежать вперед и прочитать в главе 17 о том, как оптимизировать приложение для разных размеров экрана.

13

Панель инструментов

Панель инструментов (toolbar) является ключевым компонентом любого хорошо спроектированного приложения Android. Панель инструментов содержит действия, которые могут выполняться пользователем, и новые средства навигации, а также обеспечивает единство дизайна и фирменного стиля.

В этой главе мы создадим для приложения CriminalIntent меню, которое будет отображаться на панели инструментов. В этом меню будет присутствовать элемент действия (action item) для добавления нового преступления. Также мы обеспечим работу кнопки Up на панели инструментов (рис. 13.1).

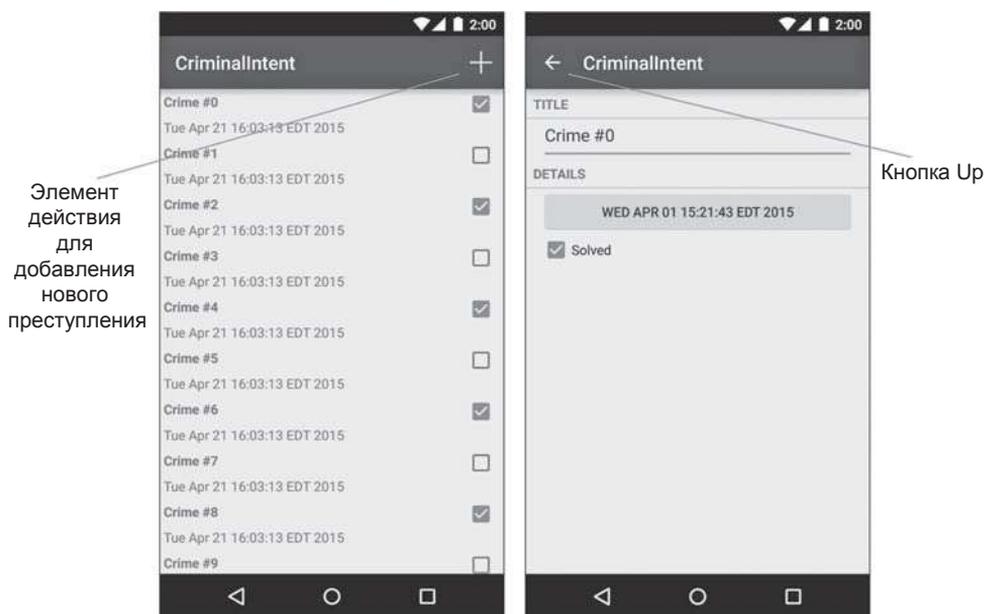


Рис. 13.1. Панель инструментов CriminalIntent

AppCompat

Компонент панели инструментов появился в Android с выходом Android 5.0 (Lollipop). До Lollipop для навигации и размещения действий в приложении рекомендовалось использовать *панель действий* (action bar).

Панель действий и панель инструментов очень похожи. Панель инструментов расширяет возможности панели действий: она обладает улучшенным пользовательским интерфейсом и отличается большей гибкостью в использовании.

Приложение `CriminalIntent` поддерживает API уровня 16+; это означает, что вы не сможете использовать встроенную реализацию панели инструментов во всех поддерживаемых версиях Android. К счастью, панель инструментов была адаптирована для библиотеки `AppCompat`. Библиотека `AppCompat` позволяет реализовать функциональность панели инструментов Lollipop в любой версии Android вплоть до API 7 (Android 2.1).

Использование библиотеки AppCompat

В главе 12 в приложение `CriminalIntent` была добавлена зависимость для библиотеки `AppCompat`. Полная интеграция с библиотекой `AppCompat` требует ряда дополнительных шагов. Возможно, некоторые из этих действий уже выполнены — это зависит от того, как создавался ваш проект.

Для использования библиотеки `AppCompat` необходимо:

1. добавить зависимость `AppCompat`;
1. использовать одну из тем `AppCompat`;
1. проследить за тем, чтобы все активности были subclasses `AppCompatActivity`.

Обновление темы

Так как зависимость для `AppCompat` уже добавлена, пора сделать следующий шаг — убедиться в том, что вы используете одну из тем (themes) `AppCompat`. Библиотека `AppCompat` включает три темы:

1. `Theme.AppCompat` — темная;
1. `Theme.AppCompat.Light` — светлая;
1. `Theme.AppCompat.Light.DarkActionBar` — светлая с темной панелью инструментов.

Тема приложения задается на уровне приложения; также существует необязательная возможность назначения темы на уровне активностей в файле `AndroidManifest.xml`. Откройте файл `AndroidManifest.xml` и найдите тег `application`. Обратите внимание на атрибут `android:theme`. Он выглядит примерно так, как показано в листинге 13.1.

Листинг 13.1. Стандартный манифест (AndroidManifest.xml)

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

```
...
```

AppTheme определяется в файле `res/values/styles.xml`. В зависимости от того, как создавался исходный проект, он может содержать несколько версий AppTheme в нескольких файлах `styles.xml`. Эти файлы уточняются спецификаторами ресурсов для разных версий Android. При использовании библиотеки AppCompatActivity переключать темы в зависимости от версии Android не придется, потому что она обеспечивает единый стиль представления на всех платформах.

Если ваш проект содержит несколько версий файла `styles.xml`, удалите лишние файлы. Должен остаться только один файл `styles.xml` в каталоге `res/values/styles.xml` (рис. 13.2).



Рис. 13.2. Дополнительные файлы `styles.xml`

После стирания дополнительных файлов откройте `res/values/styles.xml` и убедитесь в том, что атрибут `parent` элемента `AppTheme` соответствует выделенной части в листинге 13.2.

Листинг 13.2. Использование темы AppCompatActivity (`res/values/styles.xml`)

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        </style>
</resources>
```

Стили и темы более подробно рассматриваются в главе 20.

Использование AppCompatActivity

Последним шагом на пути перехода к AppCompatActivity становится преобразование всех активностей `CriminalIntent` в subclasses AppCompatActivity. До настоящего момента все активности были subclasses `FragmentActivity`, что позволяло нам использовать реализацию фрагментов из библиотеки поддержки.

Класс AppCompatActivity сам является subclasses FragmentActivity. Это означает, что вы по-прежнему можете использовать поддержку фрагментов в AppCompatActivity, что упрощает необходимые изменения в CriminalIntent.

Преобразуйте SingleFragmentActivity и CrimePagerActivity в subclasses AppCompatActivity.

(Почему не CrimeListActivity? Потому что это subclasses SingleFragmentActivity.)

Листинг 13.3. Преобразование в AppCompatActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends FragmentActivity {
public abstract class SingleFragmentActivity extends AppCompatActivity {
    ...
}
```

Листинг 13.4. Преобразование в AppCompatActivity (CrimePagerActivity.java)

```
public class CrimePagerActivity extends FragmentActivity AppCompatActivity {
    ...
}
```

Запустите CriminalIntent и убедитесь в том, что запуск происходит без сбоев.

Приложение должно выглядеть примерно так, как показано на рис. 13.3.

Теперь, когда в приложении CriminalIntent используется панель инструментов AppCompatActivity, вы можете добавлять действия на панель инструментов.



Рис. 13.3. Новая панель инструментов

Меню

Правая верхняя часть панели инструментов зарезервирована для меню. Меню состоит из *элементов действий* (иногда также называемых *элементами меню*), выполняющих действия на текущем экране или в приложении в целом. Мы добавим элемент действия, при помощи которого пользователь сможет создать описание нового преступления.

Для работы меню потребуется несколько строковых ресурсов. Добавьте их в файл `strings.xml` (листинг 13.5). Пока эти строки выглядят довольно загадочно, но лучше решить эту проблему сразу. Когда они понадобятся нам позднее, они уже будут на своем месте и нам не придется отвлекаться от текущих дел.

Листинг 13.5. Добавление строк для меню (`res/values/strings.xml`)

```
<resources>
  ...
  <string name="date_picker_title">Date of crime:</string>
  <string name="new_crime">New Crime</string>
  <string name="show_subtitle">Show Subtitle</string>
  <string name="hide_subtitle">Hide Subtitle</string>
  <string name="subtitle_format">%1$s crimes</string>
</resources>
```

Определение меню в XML

Меню определяются такими же ресурсами, как и макеты. Вы создаете описание меню в XML и помещаете файл в каталог `res/menu` своего проекта. Android генерирует идентификатор ресурса для файла меню, который затем используется для заполнения меню в коде.

В окне инструментов Project щелкните правой кнопкой мыши на каталоге `res` и выберите команду `New` ▶ `Android resource File`. Выберите тип ресурса `Menu`, присвойте ресурсу меню имя `fragment_crime_list` и щелкните на кнопке `OK`. Android Studio генерирует файл `res/menu/fragment_crime_list.xml` (рис. 13.4).

Добавьте в новый файл `fragment_crime_list.xml` элемент `item` (листинг 13.6).

Листинг 13.6. Создание ресурса меню `CrimeListFragment` (`fragment_crime_list.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto">
  <item
    android:id="@+id/menu_item_new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:title="@string/new_crime"
    app:showAsAction="ifRoom|withText"/>
</menu>
```

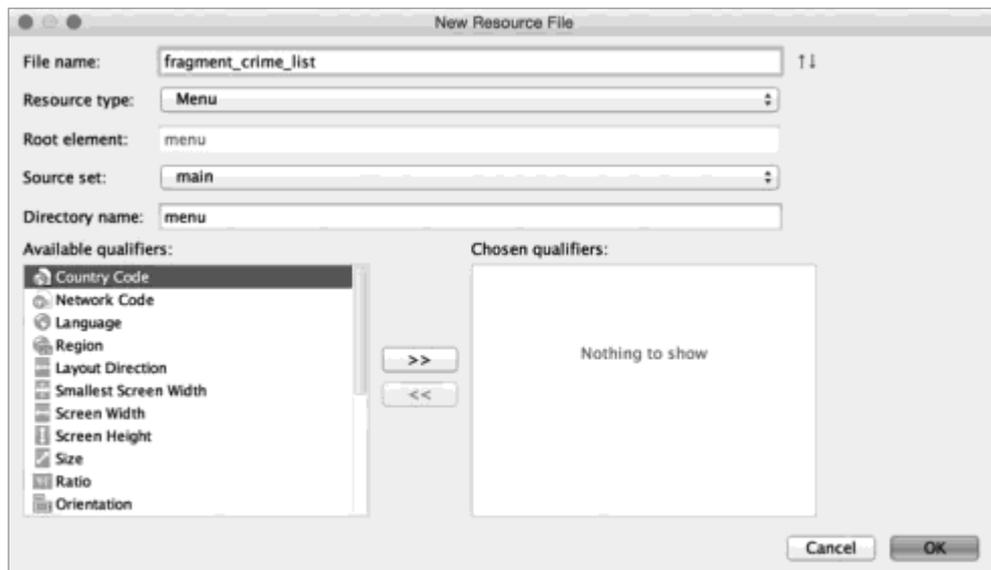


Рис. 13.4. Создание файла меню

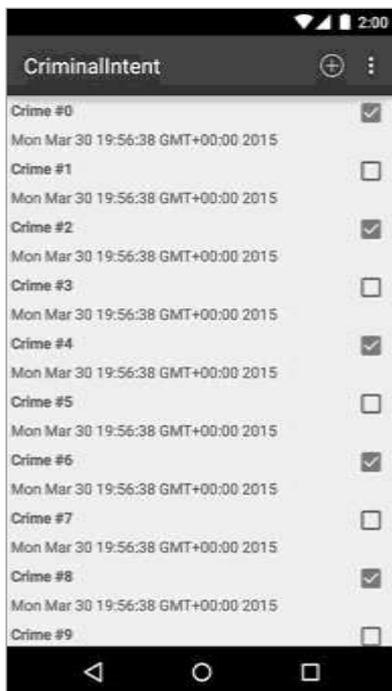


Рис. 13.5. Дополнительное меню на панели инструментов

Атрибут `showAsAction` показывает, должна команда меню отображаться на самой панели инструментов или в *дополнительном меню* (overflow menu). Мы объединили два значения, `ifRoom` и `withText`, чтобы при наличии свободного места на панели инструментов отображались значок и текст команды. Если на панели не хватает места для текста, то отображается только значок. Если места нет ни для того ни для другого, команда перемещается в дополнительное меню.

Дополнительное меню вызывается значком в виде трех точек в правой части панели инструментов (рис. 13.5).

Также атрибут `showAsAction` может принимать значения `always` и `never`. Выбирать `always` не рекомендуется; лучше использовать `ifRoom` и предоставить решение ОС. Вариант `never` хорошо подходит для редко выполняемых действий. Как правило, на панели инструментов следует размещать только часто используемые команды меню, чтобы не загромождать экран.

Пространство имен `app`

Заметьте, что в файле `fragment_crime_list.xml` тег `xmlns` используется для определения нового пространства имен `app` отдельно от обычного объявления пространства имен `android`. Затем пространство имен `app` используется для назначения атрибута `showAsAction`.

Это необычное объявление пространства имен существует для обеспечения совместимости с библиотекой `AppCompat`. API панели действий впервые появился в Android 3.0. Изначально библиотека `AppCompat` была создана для включения совместимой версии панели действий, поддерживающей более ранние версии Android, чтобы панель действий могла использоваться на любых устройствах — даже не поддерживающих встроенную панель действий.

На устройствах с Android 2.3 и более ранними версиями меню и соответствующая разметка XML не существовали, но атрибут `android:showAsAction` добавился только с выпуском панели действий.

Библиотека `AppCompat` определяет собственный атрибут `showAsAction`, игнорируя системную реализацию `showAsAction`.

Android Asset Studio

В атрибуте `android:icon` значение `@android:drawable/ic_menu_add` ссылается на *системный значок* (system icon). Системные значки находятся на устройстве, а не в ресурсах проекта.

В прототипе приложения ссылки на системные значки работают нормально. Однако в приложении, готовом к выпуску, лучше быть уверенным в том, что именно пользователь увидит на экране. Системные значки могут сильно различаться между устройствами и версиями ОС, а на некоторых устройствах системные значки могут не соответствовать дизайну приложения.

Одно из возможных решений — создание собственных значков. Вам придется подготовить версии для каждого разрешения экрана, а возможно, и для других конфигураций устройств. За дополнительной информацией обращайтесь к руководству «Android’s Icon Design Guidelines» по адресу <http://developer.android.com/design/style/iconography.html>.

Также можно действовать иначе: найти системные значки, соответствующие потребностям вашего приложения, и скопировать их прямо в графические ресурсы проекта.

Системные значки находятся в каталоге Android SDK. На Mac это обычно каталог вида `/Users/пользователь/Library/Android/sdk`. В Windows по умолчанию используется путь `\Users\пользователь\sdk`. Также для получения местонахождения SDK можно открыть окно Project Structure и выбрать категорию SDK Location.

В каталоге SDK вы найдете разные ресурсы Android, включая `ic_menu_add`. Эти ресурсы находятся в каталоге `/platforms/android-21/data/res`, где 21 — уровень API Android-версии.

Третий, и самый простой, вариант — использовать программу Android Asset Studio, включенную в Android Studio. Asset Studio позволяет создать и настроить изображение для использования на панели инструментов.

Щелкните правой кнопкой мыши на каталоге `drawable` в окне инструментов Project и выберите команду `New ▶ Image Asset`. На экране появляется Asset Studio (рис. 13.6).

В Asset Studio можно генерировать значки нескольких разных типов. Выберите в поле Asset Type: вариант `Action Bar and Tab Icons`. Затем в группе `Foreground` выберите переключатель `Clipart` и нажмите кнопку `Choose`, чтобы выбрать графическую заготовку.

Воткрывшемся окне выберите изображение, напоминающее знак+ (рис. 13.7).

Наконец, введите имя `ic_menu_add` и нажмите кнопку `Next` (рис. 13.8).

Asset Studio предлагает выбрать модуль и каталог для добавления изображения. Оставьте значения по умолчанию, чтобы добавить изображение в модуль `app`. В окне также выводится план работы, которую выполнит Asset Studio. Обратите внимание: значки `mdpi`, `hdpi`, `xhdpi` и `xxhdpi` будут созданы автоматически. Класс!

Щелкните на кнопке `Finish`, чтобы сгенерировать изображения. Затем в файле макета измените атрибут `icon` и включите в него ссылку на новый ресурс из вашего проекта.

Листинг 13.7. Ссылка на локальный ресурс (`menu/fragment_crime_list.xml`)

```
<item
    android:id="@+id/menu_item_new_crime"
    android:icon="@android:drawable/ic_menu_add"
    android:icon="@drawable/ic_menu_add"
    android:title="@string/new_crime"
    app:showAsAction="ifRoom|withText"/>
```

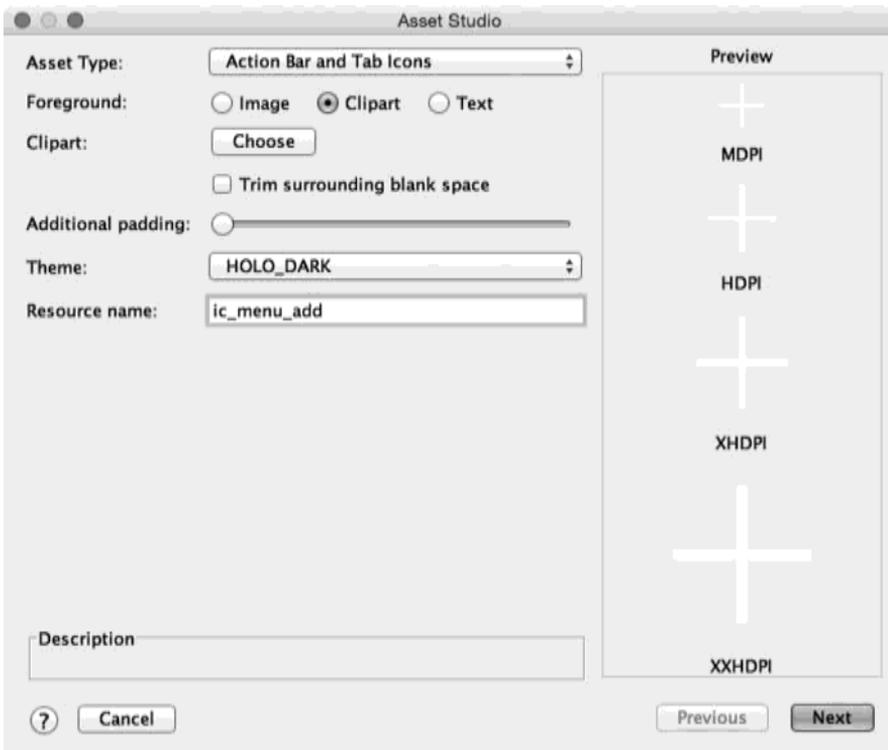


Рис. 13.6. Asset Studio

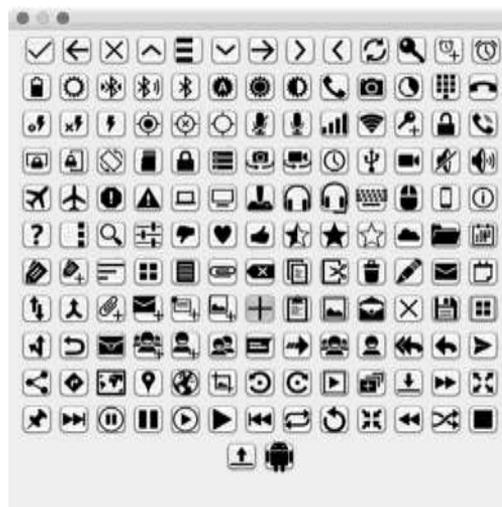


Рис. 13.7. Галерея графических заготовок: где знак +?

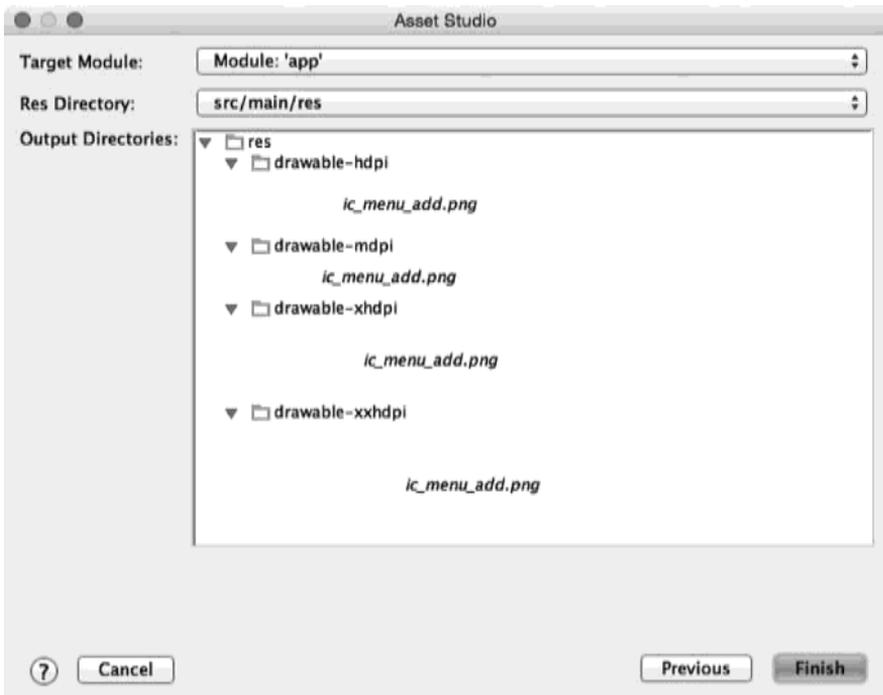


Рис. 13.8. Файлы, сгенерированные в Asset Studio

Создание меню

Для управления меню в коде используются методы обратного вызова класса Activity. Когда возникает необходимость в меню, Android вызывает метод Activity с именем `onCreateOptionsMenu(Menu)`.

Однако архитектура нашего приложения требует, чтобы реализация находилась в фрагменте, а не в активности. Класс `Fragment` содержит собственный набор методов обратного вызова для командных меню, которые мы реализуем в `CrimeListFragment`. Для создания меню и обработки выбранных команд используются следующие методы:

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater)
public boolean onOptionsItemSelected(MenuItem item)
```

В файле `CrimeListFragment.java` переопределите метод `onCreateOptionsMenu(Menu, MenuInflater)` так, чтобы он заполнял меню, определенное в файле `fragment_crime_list.xml`.

Листинг 13.8. Заполнение ресурса меню (`CrimeListFragment.java`)

```
@Override
public void onResume() {
    super.onResume();
```

```
        updateUI();
    }

    @Override
    public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
        super.onCreateOptionsMenu(menu, inflater);
        inflater.inflate(R.menu.fragment_crime_list, menu);
    }
}
```

В этом методе мы вызываем метод `MenuInflater.inflate(int, Menu)` и передаем идентификатор ресурса своего файла меню. Вызов заполняет экземпляр `Menu` командами, определенными в файле.

Обратите внимание на вызов реализации `onCreateOptionsMenu(...)` суперкласса. Он не обязателен, но мы рекомендуем вызывать версию суперкласса просто для соблюдения общепринятой схемы, чтобы работала вся функциональность командных меню, определяемая в суперклассе. Впрочем, в данном случае это лишь формальность — базовая реализация этого метода из `Fragment` не делает ничего.

`FragmentManager` отвечает за вызов `Fragment.onCreateOptionsMenu(Menu, MenuInflater)` при получении активностью обратного вызова `onCreateOptionsMenu(...)` от ОС. Вы должны явно указать `FragmentManager`, что фрагмент должен получить вызов `onCreateOptionsMenu(...)`. Для этого вызывается следующий метод:

```
public void setHasOptionsMenu(boolean hasMenu)
```

В методе `CrimeListFragment.onCreate(...)` сообщите `FragmentManager`, что экземпляр `CrimeListFragment` должен получать обратные вызовы командного меню.

Листинг 13.9. Получение обратных вызовов (`CrimeListFragment.java`)

```
...
private RecyclerView mCrimeRecyclerView;
private CrimeAdapter mAdapter;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setHasOptionsMenu(true);
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
...
}
```

В приложении `CriminalIntent` появляется командное меню (рис. 13.9).

Где текст командного меню? У большинства телефонов в книжной ориентации хватает места только для значка. Текст команды открывается долгим нажатием на значке на панели инструментов (рис. 13.10).



Рис. 13.9. Значок команды меню на панели инструментов



Рис. 13.10. Долгое нажатие на значке на панели инструментов выводит текст команды

В альбомной ориентации на панели инструментов хватает места как для значка, так и для текста (рис. 13.11).



Рис. 13.11. Значок и текст на панели инструментов

Реакция на выбор команд

Чтобы отреагировать на выбор пользователем команды New Crime, нам понадобится механизм добавления нового объекта Crime в список. Включите в файл CrimeLab.java следующий метод.

Листинг 13.10. Добавление нового объекта Crime (CrimeLab.java)

```
...  
  
public void addCrime(Crime c) {  
    mCrimes.add(c);  
}  
  
public List<Crime> getCrimes() {  
    return mCrimes;  
}  
  
...
```

Теперь, когда вы сможете вводить описания преступлений самостоятельно, программное генерирование 100 объектов становится лишним. В файле CrimeLab.java удалите код, генерирующий эти преступления.

Листинг 13.11. Долой случайные преступления! (CrimeLab.java)

```
private CrimeLab(Context context) {  
    mCrimes = new ArrayList<>();  
    for (int i = 0; i < 100; i++) {  
        Crime crime = new Crime();  
        crime.setTitle("Crime #" + i);  
        crime.setSolved(i % 2 == 0);  
        mCrimes.add(crime);  
    }  
}
```

Когда пользователь выбирает команду в командном меню, фрагмент получает обратный вызов метода onOptionsItemSelected(Menuitem). Этот метод получает экземпляр Menuitem, описывающий выбор пользователя.

И хотя наше меню состоит всего из одной команды, в реальных меню их обычно больше. Чтобы определить, какая команда меню была выбрана, проверьте идентификатор команды меню и отреагируйте соответствующим образом. Этот идентификатор соответствует идентификатору, назначенному команде в файле меню.

В файле CrimeListFragment.java реализуйте метод onOptionsItemSelected(Menuitem), реагирующий на выбор команды меню. Реализация создает новый объект Crime, добавляет его в CrimeLab и запускает экземпляр CrimePagerActivity для редактирования нового объекта Crime.

Листинг 13.12. Реакция на выбор команды меню (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
            Intent intent = CrimePagerActivity
                .newIntent(getActivity(), crime.getId());
            startActivity(intent);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Метод возвращает логическое значение. После того как команда меню будет обработана, верните true; тем самым вы сообщаете, что дальнейшая обработка не нужна. Секция default вызывает реализацию суперкласса, если идентификатор команды не известен в вашей реализации.

Запустите приложение CriminalIntent и опробуйте новую команду. Добавьте несколько преступлений и отредактируйте их. (Пустой список до начала ввода может сбить с толку пользователя. Упражнение в конце этой главы будет выдавать подсказку для пользователя при пустом списке.)

Включение иерархической навигации

До настоящего момента приложение CriminalIntent использует кнопку Back для навигации по приложению. Кнопка Back возвращает приложение к предыдущему состоянию. С другой стороны, *иерархическая навигация* осуществляет перемещение по иерархии приложения.

В иерархической навигации пользователь переходит на один уровень «наверх» к родителю текущей активности при помощи кнопки Up в левой части панели инструментов. До выхода Jelly Bean (API уровня 16) разработчику приходилось вручную отображать кнопку Up и вручную обрабатывать нажатия. Начиная с Jelly Bean эта функциональность реализуется намного проще.

Чтобы включить иерархическую навигацию в приложение CriminalIntent, добавьте атрибут parentActivityName в файл AndroidManifest.xml.

Листинг 13.13. Включение кнопки Up (AndroidManifest.xml)

```
...
<activity
    android:name=".CrimePagerActivity"
    android:label="@string/app_name"
    android:parentActivityName=".CrimeListActivity">
</activity>
...
```

Запустите приложение CriminalIntent и создайте новое преступление. Обратите внимание на появление кнопки Up (рис. 13.12). Нажатие кнопки Up переводит приложение на один уровень вверх в иерархии CriminalIntent к активности CrimeListActivity.



Рис. 13.12. Кнопка Up в CrimePagerActivity

Как работает иерархическая навигация

В приложении CriminalIntent навигация кнопкой Back и кнопкой Up выполняет одну и ту же задачу. Нажатие любой из этих кнопок в CrimePagerActivity возвращает пользователя обратно к CrimeListActivity. И хотя результат один и тот же, «за кулисами» происходят совершенно разные события. Эти различия важны, потому что в зависимости от приложения кнопка Up может вернуть пользователя на несколько активностей назад в стеке.

Когда пользователь переходит вверх по иерархии из CrimeActivity, создается интент следующего вида:

```
Intent intent = new Intent(this, CrimeListActivity.class);
intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
startActivity(intent);
finish();
```

Флаг FLAG_ACTIVITY_CLEAR_TOP приказывает Android провести поиск существующего экземпляра активности в стеке и, если он будет найден, вывести из стека все остальные активности, чтобы запускаемая активность была верхней (рис. 13.13).

Альтернативная команда меню

В этом разделе все, что мы узнали о меню, совместимости и альтернативных ресурсах, будет использовано для добавления команды меню, скрывающей и отображающей подзаголовки в панели инструментов CrimeListActivity.

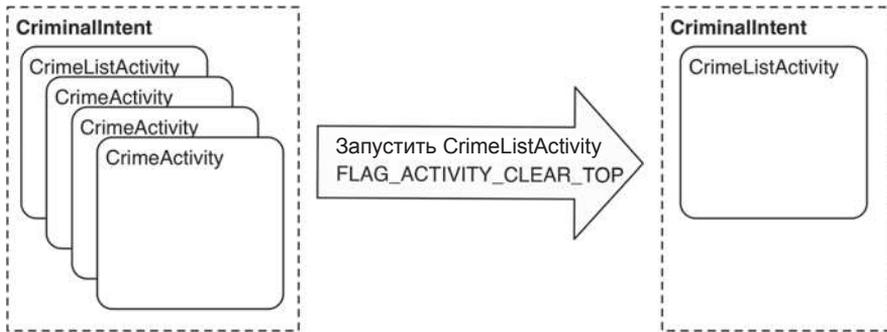


Рис. 13.13. FLAG_ACTIVITY_CLEAR_TOP в действии

В файле `res/menu/fragment_crime_list.xml` добавьте элемент действия `Show Subtitle`, который будет отображаться на панели инструментов при наличии свободного места.

Листинг 13.14. Добавление элемента действия `Show Subtitle` (`res/menu/fragment_crime_list.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/menu_item_new_crime"
        android:icon="@android:drawable/ic_menu_add"
        android:title="@string/new_crime"
        app:showAsAction="ifRoom|withText"/>

    <item
        android:id="@+id/menu_item_show_subtitle"
        android:title="@string/show_subtitle"
        app:showAsAction="ifRoom"/>
</menu>
```

Команда отображает подзаголовок с количеством преступлений в `CriminalIntent`. Создайте новый метод `updateSubtitle()`, который будет задавать подзаголовок панели инструментов.

Листинг 13.15. Назначение подзаголовка панели инструментов (`CrimeListFragment.java`)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    ...
}

private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
```

```
int crimeCount = crimeLab.getCrimes().size();
String subtitle = getString(R.string.subtitle_format, crimeCount);

AppCompatActivity activity = (AppCompatActivity) getActivity();
activity.getSupportActionBar().setSubtitle(subtitle);
}
```

Метод `updateSubtitle` генерирует строку подзаголовка при помощи метода `getString(int resId, Object... formatArgs)`, который получает значения, подставляемые на место заполнителей в строковом ресурсе.

Затем активность, являющаяся хостом для `CrimeListFragment`, преобразуется в `AppCompatActivity`. `CriminalIntent` использует библиотеку `AppCompatActivity`, поэтому все активности должны быть subclassesми `AppCompatActivity`, чтобы панель инструментов была доступной для приложения. По историческим причинам панель инструментов во многих местах библиотеки `AppCompatActivity` называется «панелью действий».

Итак, теперь метод `updateSubtitle` определен, и мы можем вызвать его при нажатии нового элемента действий.

Листинг 13.16. Обработка элемента действия Show Subtitle (`CrimeListFragment.java`)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
        case R.id.menu_item_show_subtitle:
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Запустите `CriminalIntent`, нажмите элемент Show Subtitle и убедитесь в том, что в подзаголовке отображается количество преступлений.

Переключение текста команды

Теперь подзаголовок отображается, но текст команды меню остался неизменным: Show Subtitle. Было бы лучше, если бы текст команды и функциональность команды меню изменялись в зависимости от текущего состояния подзаголовка.

При вызове `onOptionsItemSelected(...)` в параметре передается объект `MenuItem` для элемента действия, нажатого пользователем. Текст элемента Show Subtitle можно было бы обновить в этом методе, но изменения будут потеряны при повороте устройства и повторном создании панели инструментов.

Другое, более правильное решение — обновить объект `MenuItem` для Show Subtitle в `onCreateOptionsMenu(...)` и инициализировать повторное создание панели инструмен-

тов при нажатии на элементе действия. Это позволит вам заново использовать код обновления элемента действия как при выборе элемента действия пользователем, так и при повторном создании панели инструментов.

Сначала добавьте переменную для хранения признака видимости подзаголовка.

Листинг 13.17. Хранение признака видимости подзаголовка (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {

    private RecyclerView mCrimeRecyclerView;
    private CrimeAdapter mAdapter;
    private boolean mSubtitleVisible;
    ...
}
```

Затем измените подзаголовок в onCreateOptionsMenu(...) и иницилируйте повторное создание элементов действий при нажатии элемента действия Show Subtitle.

Листинг 13.18. Обновление MenuItem (CrimeListFragment.java)

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.fragment_crime_list, menu);

    MenuItem subtitleItem = menu.findItem(R.id.menu_item_show_subtitle);
    if (mSubtitleVisible) {
        subtitleItem.setTitle(R.string.hide_subtitle);
    } else {
        subtitleItem.setTitle(R.string.show_subtitle);
    }
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            ...
        case R.id.menu_item_show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Наконец, проверьте переменную mSubtitleVisible при отображении или сокрытии подзаголовка панели инструментов.

Листинг 13.19. Отображение или сокрытие подзаголовка (CrimeListFragment.java)

```
private void updateSubtitle() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    int crimeCount = crimeLab.getCrimes().size();
    String subtitle = getString(R.string.subtitle_format, crimeCount);

    if (!mSubtitleVisible) {
        subtitle = null;
    }

    AppCompatActivity activity = (AppCompatActivity) getActivity();
    activity.getSupportActionBar().setSubtitle(subtitle);
}
```

Запустите приложение CriminalIntent и убедитесь в том, что подзаголовок успешно скрывается и отображается. Обратите внимание на изменение текста команды в зависимости от состояния подзаголовка.

«Да, и еще одно...»

Программирование Android часто напоминает беседы с детективом Коломбо из сериала. Вы уже думаете, что все прошло как по маслу и у следствия нет претензий. Но Android всегда поворачивается у двери и говорит: «Да, и еще одно...»

Впрочем, в нашем случае, точнее, «еще два». Во-первых, при создании нового преступления и последующем возвращении к CrimeListActivity кнопкой Back содержимое подзаголовка не соответствует новому количеству преступлений. Во-вторых, если отобразить подзаголовок, а потом повернуть устройство, подзаголовков исчезнет.

Начнем с первой проблемы. Она решается обновлением текста подзаголовка при возвращении к CrimeListActivity. Иницилируйте вызов updateSubtitle в onResume. Ваш метод updateUI уже вызывается в onResume и onCreate; добавьте вызов updateSubtitle в метод updateUI.

Листинг 13.20. Вывод обновленного состояния (CrimeListFragment.java)

```
private void updateUI() {
    CrimeLab crimeLab = CrimeLab.get(getActivity());
    List<Crime> crimes = crimeLab.getCrimes();

    if (mAdapter == null) {
        mAdapter = new CrimeAdapter(crimes);
        mCrimeRecyclerView.setAdapter(mAdapter);
    } else {
        mAdapter.notifyDataSetChanged();
    }

    updateSubtitle();
}
```

Запустите `CriminalIntent`, отобразите подзаголовок, создайте новое преступление и нажмите на устройстве кнопку `Back`, чтобы вернуться к `CrimeListActivity`. На этот раз на панели инструментов будет отображаться правильное количество.

Повторите эти действия, но вместо кнопки `Back` воспользуйтесь кнопкой `Up`. Подзаголовок снова становится невидимым. Почему это происходит?

У реализации иерархической навигации в Android имеется неприятный побочный эффект: активность, к которой вы переходите кнопкой `Up`, полностью создается заново, с нуля. Это означает, что все переменные экземпляров теряются, а значит, все сохраненное состояние экземпляров также будет потеряно. Родительская активность воспринимается как совершенно новая активность.

Не существует простого способа обеспечить вывод подзаголовка при переходе кнопкой `Up`. Одно из возможных решений — переопределение механизма перехода. В `CriminalIntent` можно вызвать метод `finish` для `CrimePagerActivity`, чтобы вернуться к предыдущей активности. Такое решение прекрасно сработает в `CriminalIntent`, но не подойдет для решений с более реалистичной иерархией, так как возврат произойдет только на одну активность.

Другое возможное решение — передача `CrimePagerActivity` информации о видимости подзаголовка в дополнительных данных интента при запуске. Переопределите метод `getParentActivityIntent()` в `CrimePagerActivity`, чтобы добавить дополнение в интент, используемый для воссоздания `CrimeListActivity`. Это решение требует, чтобы класс `CrimePagerActivity` обладал подробной информацией о том, как работает его родитель.

Оба решения не идеальны, причем хорошей альтернативы не существует.

Теперь в подзаголовке всегда отображается правильное количество преступлений, и мы можем заняться решением проблемы с поворотом. Для этого следует сохранить переменную экземпляра `mSubtitleVisible` между поворотами при помощи механизма сохранения состояния экземпляров.

Листинг 13.21. Сохранение признака видимости подзаголовка (`CrimeListFragment.java`)

```
public class CrimeListFragment extends Fragment {

    private static final String SAVED_SUBTITLE_VISIBLE = "subtitle";
    ...

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...
        if (savedInstanceState != null) {
            mSubtitleVisible = savedInstanceState.getBoolean(
                SAVED_SUBTITLE_VISIBLE);
        }

        updateUI();
    }
}
```

```
        return view;
    }

    @Override
    public void onResume() {
        ...
    }

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }
}
```

Запустите приложение `CriminalIntent`. Отобразите подзаголовок, поверните устройство. Подзаголовок должен появиться в воссозданном представлении, как и ожидалось.

Для любознательных: панели инструментов и панели действий

Чем панель инструментов отличается от панели действий?

Наиболее очевидное различие между ними — измененный визуальный стиль панели инструментов. В левой части панели инструментов нет знака, а интервалы между элементами действий в правой части сокращаются. Другое существенное визуальное изменение — кнопка `Up`. На панели действий эта кнопка была менее заметной и играла второстепенную роль.

Помимо этих визуальных различий, панели инструментов проектировались прежде всего с расчетом на большую гибкость, чем у панели действий. Панель действий должна подчиняться множеству ограничений. Она всегда отображается у верхнего края экрана. У приложения может быть только одна панель действий. Размер панели действий фиксирован, и изменить его невозможно. У панелей инструментов этих ограничений нет.

В этой главе мы использовали панель инструментов, позаимствованную из тем `AppBarCompat`. Также можно вручную включить панель инструментов как обычное представление в файл макета активности или фрагмента. Панель инструментов можно разместить где угодно, и на экране могут одновременно находиться несколько панелей инструментов. Эта гибкость открывает интересные возможности: например, представьте, что каждый фрагмент, используемый в вашем приложении, поддерживает собственную панель инструментов. При одновременном размещении на экране нескольких фрагментов каждый из них может отображать собственную панель инструментов вместо того, чтобы совместно использовать одну панель инструментов у верхнего края экрана.

Упражнение. Удаление преступлений

Приложение CriminalIntent дает возможность создать новое преступление, но не предоставляет средств для удаления его из «протокола». В этом упражнении добавьте в CrimeFragment новый элемент действия для удаления текущего преступления. После того как пользователь нажимает элемент удаления, не забудьте вернуть его к предыдущей активности вызовом метода finish для активности-хоста CrimeFragment.

Упражнение. Множественное число в строках

Если список содержит одно преступление, подзаголовок «1 crimes» становится грамматически неправильным. В этом упражнении вам предлагается исправить текст подзаголовка.

Вы можете создать две разные строки и выбрать нужную в коде, но такое решение быстро разваливается при попытке локализовать приложение для разных языков. Лучше использовать строковые ресурсы (иногда называемые *количественными строками*) во множественном числе.

Сначала определите в файле strings.xml элемент plurals:

```
<plurals name="subtitle_plural">
    <item quantity="one">%1$s crime</item>
    <item quantity="other">%1$s crimes</item>
</plurals>
```

Затем используйте метод getQuantityString для правильного формирования множественного числа строки:

```
int crimeSize = crimeLab.getCrimes().size();
String subtitle = getResources()
    .getQuantityString(R.plurals.subtitle_plural, crimeSize, crimeSize);
```

Упражнение. Пустое представление для списка

В настоящее время при запуске CriminalIntent отображает пустой виджет RecyclerView — большую черную пустоту. Мы должны предоставить пользователям что-то для взаимодействия при отсутствии элементов в списке.

Пусть в пустом представлении выводится сообщение (например, «Список пуст»). Добавьте в представление кнопку, которая будет инициировать создание нового преступления.

Для отображения и сокрытия нового представления-заполнителя используйте метод setVisibility, существующий в каждом классе View.

14

Базы данных SQLite

Почти каждому приложению необходимо место для долгосрочного хранения данных — более длительного, чем позволяет `savedInstanceState`. Android предоставляет вам такое место: локальную файловую систему во флеш-памяти телефона или планшета.

Каждое приложение на устройстве Android имеет каталог в своей *песочнице* (`sandbox`). Хранение файлов в песочнице защищает их от других приложений и даже от любопытных глаз пользователей (если только устройство не было «взломяно» — в этом случае пользователь сможет делать все, что ему заблагорассудится).

Песочница каждого приложения представляет собой подкаталог каталога `/data/data`, имя которого соответствует имени пакета приложения. Для `CriminalIntent` полный путь к каталогу песочницы имеет вид `/data/data/com.bignerdranch.android.criminalintent`.

Тем не менее большинство данных приложений не хранится в простых файлах. И на то есть веская причина: допустим, у вас имеется файл, в котором записаны данные всех объектов `Crime`. Чтобы изменить краткое описание преступления в начале файла, вам придется прочитать весь файл и записать его заново. При большом количестве записей эта процедура займет много времени.

На помощь приходит SQLite — реляционная база данных с открытым кодом, как и MySQL или PostgreSQL. В отличие от других баз данных, SQLite хранит свои данные в простых файлах, для чтения и записи которых может использоваться библиотека SQLite. Библиотека SQLite входит в стандартную библиотеку Android вместе с другими вспомогательными классами Java.

В этой главе описаны не все возможности SQLite. Более подробную информацию можно найти в полной документации SQLite по адресу <http://www.sqlite.org>. Здесь будет показано, как работают основные классы поддержки SQLite в Android. Вы сможете выполнять операции открытия, чтения и записи с базами данных SQLite в песочнице приложения, даже не зная, где именно они хранятся.

Определение схемы

Прежде чем создавать базу данных, необходимо решить, какая информация будет в ней храниться. В `CriminalIntent` хранится только список преступлений, поэтому мы определим одну таблицу с именем `crimes` (рис. 14.1).

<code>_id</code>	<code>uuid</code>	<code>title</code>	<code>date</code>	<code>solved</code>
1	13090636733242	Stolen yoghurt	13090636733242	0
2	13090732131909	Dirty sink	13090732131909	1

Рис. 14.1. Таблица `crimes`

В мире программирования у задач такого рода существует много разных решений, но разработчик должен неизменно руководствоваться принципом DRY. Это сокращение (от «Don't Repeat Yourself», то есть «Не повторяйтесь») обозначает одно из практических правил, которыми следует руководствоваться при написании программы: когда вы пишете какой-то код, запишите его в одном месте. В этом случае вместо того, чтобы плодить дубликаты, вы всегда будете обращаться к одному централизованному источнику информации.

Соблюдение этого принципа при работе с базами данных может быть весьма непростой задачей. Существуют сложные программы, называемые средствами *объектно-реляционного отображения* (Object-Relational Mappers, сокращенно ORM), которые позволяют использовать объекты модели (такие, как `Crime`) как Единственно Верное Определение. В этой главе мы пойдем по более простому пути и определим в коде Java упрощенную схему базы данных, в которой будут храниться имя таблицы и описания ее столбцов.

Начните с создания класса для хранения схемы. Этот класс будет называться `CrimeDbSchema`, но вы должны ввести в диалоговом окне New Class имя `database.CrimeDbSchema`. Файл `CrimeDbSchema.java` будет помещен в отдельный пакет `database`, который будет использоваться для организации всего кода, относящегося к базам данных.

В классе `CrimeDbSchema` определите внутренний класс `CrimeTable` для описания таблицы.

Листинг 14.1. Определение `CrimeTable` (`CrimeDbSchema.java`)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";
    }
}
```

Класс `CrimeTable` существует только для определения строковых констант, необходимых для описания основных частей определения таблицы. Определение начинается с имени таблицы в базе данных `CrimeTable.NAME`, за которым следуют описания столбцов.

Листинг 14.2. Определение столбцов таблицы (CrimeDbSchema.java)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

При наличии такого определения вы сможете обращаться к столбцу с именем `title` в синтаксисе, безопасном для кода Java: `CrimeTable.Cols.TITLE`. Такой синтаксис существенно снижает риск изменения программы, если вам когда-нибудь понадобится изменить имя столбца или добавить новые данные в таблицу.

Построение исходной базы данных

После определения схемы можно переходить к созданию базы данных. Android предоставляет в классе `Context` низкоуровневые методы для открытия файла базы данных в экземпляр `SQLiteDatabase`: `openOrCreateDatabase(...)` и `databaseList()`.

Тем не менее на практике при открытии базы данных всегда следует выполнить ряд простых действий:

1. Проверить, существует ли база данных.
2. Если база данных не существует, создать ее, создать таблицы и заполнить их необходимыми исходными данными.
3. Если база данных существует, открыть ее и проверить версию `CrimeDbSchema` (возможно, в будущих версиях `CriminalIntent` вы захотите добавить или удалить какие-то аспекты).
4. Если это старая версия, выполнить код преобразования ее в новую версию.

Android предоставляет класс `SQLiteOpenHelper`, который сделает это все за вас. Создайте в пакете `database` класс с именем `CrimeBaseHelper`.

Листинг 14.3. Создание `CrimeBaseHelper` (`CrimeBaseHelper.java`)

```
public class CrimeBaseHelper extends SQLiteOpenHelper {
    private static final int VERSION = 1;
    private static final String DATABASE_NAME = "crimeBase.db";

    public CrimeBaseHelper(Context context) {
        super(context, DATABASE_NAME, null, VERSION);
    }
}
```

```

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}

```

Класс SQLiteOpenHelper избавляет разработчика от рутинной работы при открытии SQLiteDatabase. Используйте его в CrimeLab для создания базы данных.

Листинг 14.4. Открытие SQLiteDatabase (CrimeLab.java)

```

public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;
    ...

    private CrimeLab(Context context) {
        mContext = context.getApplicationContext();
        mDatabase = new CrimeBaseHelper(mContext)
            .getWritableDatabase();
        mCrimes = new ArrayList<>();
    }

    ...
}

```

(Интересуетесь, зачем контекст сохраняется в переменной экземпляра? Он будет использоваться CrimeLab в главе 16.)

При вызове getWritableDatabase() класс CrimeBaseHelper:

- 1) открывает /data/data/com.bignerdranch.android.criminalintent/databases/crimeBase.db. Если файл базы данных не существует, то он создается;
- 2) если база данных открывается впервые, вызывает метод onCreate(SQLiteDatabase) с последующим сохранением последнего номера версии;
- 3) если база данных открывается не впервые, проверяет номер ее версии. Если версия базы данных в CrimeOpenHelper выше, то вызывается метод onUpgrade(SQLiteDatabase, int, int).

Мораль: код создания исходной базы данных размещается в onCreate(SQLiteDatabase), код обновления — в onUpgrade(SQLiteDatabase, int, int), а дальше все работает само собой.

Пока приложение CriminalIntent существует только в одной версии, так что на onUpgrade(...) можно не обращать внимания. Нужно только создать таблицы базы данных в onCreate(...). Для этого будет использоваться класс CrimeTable, являющийся внутренним классом CrimeDbSchema.

Процедура импортирования состоит из двух шагов. Сначала запишите начальную часть кода создания SQL:

Листинг 14.5. Первая часть onCreate(...) (CrimeBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeDbSchema.CrimeTable.NAME);
}
```

Наведите курсор на слово CrimeTable и нажмите Option+Return (Alt+Enter). Затем выберите первый вариант Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable', как показано на рис. 14.2.

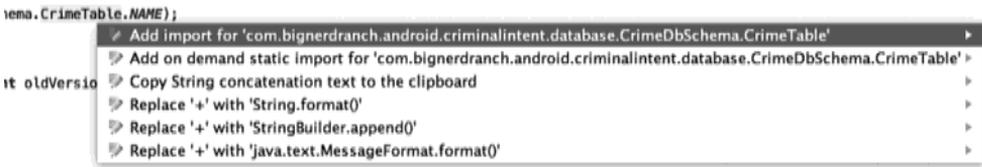


Рис. 14.2. Добавление директивы import для CrimeTable

Android Studio генерирует директиву import следующего вида:

```
...
import com.bignerdranch.android.criminalintent.database.CrimeDbSchema.
CrimeTable;

public class CrimeBaseHelper extends SQLiteOpenHelper {
    ...
```

Директива позволяет ссылаться на строковые константы из CrimeDbSchema.CrimeTable в форме CrimeTable.Cols.UUID (вместо того, чтобы вводить полное имя CrimeDbSchema.CrimeTable.Cols.UUID). Используйте это обстоятельство, чтобы завершить ввод кода определения таблицы.

Листинг 14.6. Создание таблицы (CrimeBaseHelper.java)

```
@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
        "_id integer primary key autoincrement, " +
        CrimeTable.Cols.UUID + ", " +
        CrimeTable.Cols.TITLE + ", " +
        CrimeTable.Cols.DATE + ", " +
        CrimeTable.Cols.SOLVED +
        ")"
    );
}
```

Создание таблицы в SQLite происходит проще, чем в других базах данных: вам не нужно задавать тип столбца в момент создания. Сделать это желательно, но в нашем случае упрощенное создание сэкономит не много времени.

Запустите `CriminalIntent`; приложение создает базу данных (рис. 14.3). Если приложение выполняется в эмуляторе или на «рутованном» устройстве, вы сможете увидеть созданную базу данных. (На физическом устройстве это невозможно — база данных хранится в закрытой области.) Выполните команду `Tools ▶ Android ▶ Android Device Monitor` и загляните в каталог `/data/data/com.bignerdranch.android.criminalintent/databases/`.



Рис. 14.3. Ваша база данных

Решение проблем при работе с базами данных

При написании кода для работы с базами данных SQLite иногда требуется слегка изменить структуру базы данных. Например, в следующей главе для каждого преступления будет добавлено новое поле подозреваемого. Для этого в таблицу преступлений придется добавить новый столбец. «Правильный» способ решения этой задачи заключается во включении в `SQLiteOpenHelper` кода повышения номера версии с последующим обновлением таблиц в `onUpgrade(...)`.

И этот «правильный» способ потребует изрядного объема кода — совершенно смехотворного, когда вы просто пытаетесь довести до ума первую или вторую версию базы данных. На практике лучше всего уничтожить базу данных и начать все заново, чтобы метод `SQLiteOpenHelper.onCreate(...)` был вызван снова.

Самый простой способ уничтожения базы — удаление приложения с устройства. А самый простой способ удаления приложений в стандартном варианте Android — открыть менеджер приложений и перетащить значок `CriminalIntent` к области `Uninstall` у верхнего края экрана. (Если на вашем устройстве используется нестандартная версия Android, процесс может выглядеть иначе.) Открывается экран вроде изображенного на рис. 14.4.

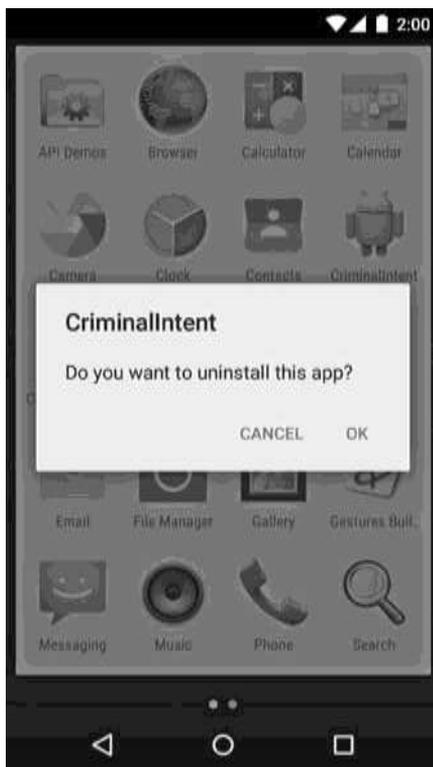


Рис. 14.4. Удаление приложения

Помните об этом приеме, если у вас возникнут проблемы при работе с базами данных в этой главе.

Изменение кода CrimeLab

Итак, теперь у вас есть база данных, и нам предстоит изменить довольно большой объем кода в CrimeLab, чтобы для хранения данных вместо mCrimes использовалась база данных mDatabase.

Для начала расчистим место для работы. Удалите из CrimeLab весь код, относящийся к mCrimes.

Листинг 14.7. Удаляем лишнее (CrimeLab.java)

```
public class CrimeLab {
    private static CrimeLab sCrimeLab;

    private List<Crime> mCrimes;
    private Context mContext;
    private SQLiteDatabase mDatabase;
```

```

public static CrimeLab get(Context context) {
    ...
}

private CrimeLab(Context context) {
    mContext = context.getApplicationContext();
    mDatabase = new CrimeBaseHelper(mContext)
        .getWritableDatabase();
    mCrimes = new ArrayList<>();
}

public void addCrime(Crime c) {
    mCrimes.add(c);
}

public List<Crime> getCrimes() {
    return mCrimes;
    return new ArrayList<>();
}

public Crime getCrime(UUID id) {
    for (Crime crime : mCrimes) {
        if (crime.getId().equals(id)) {
            return crime;
        }
    }
    return null;
}
}

```

Приложение `CriminalIntent` остается в состоянии, которое вряд ли можно назвать работоспособным; вы видите пустой список преступлений, но при добавлении преступления отображается пустая активность `CrimePagerActivity`. Неприятно, но пока сойдет.

Запись в базу данных

Работа с `SQLiteDatabase` начинается с записи данных. Ваше приложение должно вставлять новые записи в существующую таблицу, а также обновлять уже существующие данные при изменении информации.

Использование ContentValues

Запись и обновление баз данных осуществляются с помощью класса `ContentValues`. Класс `ContentValues` обеспечивает хранение пар «ключ-значение», как и контейнер `Java HashMap` или объекты `Bundle`, уже встречавшиеся вам ранее. Однако в отличие от `HashMap` или `Bundle`, он предназначен для хранения типов данных, которые могут содержаться в базах данных `SQLite`.

Экземпляры `ContentValues` будут несколько раз создаваться из `Crime` в коде `CrimeLab`. Добавьте закрытый метод, который будет преобразовывать объект `Crime` в `ContentValues`. (Не забудьте описанный ранее прием для добавления директивы `import` для `CrimeTable`: добравшись до `CrimeTable.Cols.UUID`, нажмите `Option+Return` (`Alt+Enter`) и выберите команду `Add import for 'com.bignerdranch.android.criminalintent.database.CrimeDbSchema.CrimeTable'`.)

Листинг 14.8. Создание `ContentValues` (`CrimeLab.java`)

```
public getCrime(UUID id) {
    return null;
}

private static ContentValues getContentValues(Crime crime) {
    ContentValues values = new ContentValues();
    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
    values.put(CrimeTable.Cols.TITLE, crime.getTitle());
    values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
    values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);

    return values;
}
}
```

В качестве ключей используйте имена столбцов. Эти имена не выбираются произвольно; они определяют столбцы, которые должны вставляться или обновляться в базе данных. Если имя отличается от содержащегося в базе данных (например, из-за опечатки), операция вставки или обновления завершится неудачей. В приведенном фрагменте указаны все столбцы, кроме столбца `_id`, который генерируется автоматически для однозначной идентификации записи.

Вставка и обновление записей

Объект `ContentValues` создан, можно переходить к добавлению записи в базу данных. Заполните метод `addCrime(Crime)` новой реализацией.

Листинг 14.9. Вставка записи (`CrimeLab.java`)

```
public void addCrime(Crime c) {
    ContentValues values = getContentValues(c);

    mDatabase.insert(CrimeTable.NAME, null, values);
}
```

Метод `insert(String, String, ContentValues)` получает два важных аргумента; еще один аргумент используется относительно редко. Первый аргумент определяет таблицу, в которую выполняется вставка, — в нашем случае `CrimeTable.NAME`. Последний аргумент содержит вставляемые данные.

А второй аргумент, который называется `nullColumnHack`? Что же он делает?

Представьте, что вы решили вызвать `insert(...)` с пустым объектом `ContentValues`. SQLite такую вставку не поддерживает, поэтому попытка вызова `insert(...)` завершится неудачей.

Но если передать в `nullColumnHack` значение `uuid`, пустой объект `ContentValues` будет проигнорирован. Вместо него будет передан объект `ContentValues` с полем `uuid`, содержащим `null`. Это позволит методу `insert(...)` успешно выполниться и создать новую запись.

Пригодится? Когда-нибудь — возможно... Хотя не сегодня. Но по крайней мере, теперь вы знаете о такой возможности.

Продолжим использование `ContentValues` и напишем метод обновления строк в базе данных.

Листинг 14.10. Обновление записи (`CrimeLab.java`)

```
public Crime getCrime(UUID id) {
    return null;
}

public void updateCrime(Crime crime) {
    String uuidString = crime.getId().toString();
    ContentValues values = getContentValues(crime);

    mDatabase.update(CrimeTable.NAME, values,
        CrimeTable.Cols.UUID + " = ?",
        new String[] { uuidString });
}

private static ContentValues getContentValues(Crime crime) {
    ContentValues values = new ContentValues();
    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
    ...
}
```

Метод `update(String, ContentValues, String, String[])` начинается так же, как `insert(...)`, — при вызове передается имя таблицы и объект `ContentValues`, который должен быть присвоен каждой обновляемой записи. Однако последняя часть отличается, потому что в этом случае необходимо указать, какие именно записи должны обновляться. Для этого строится условие `WHERE` (третий аргумент), за которым следуют значения аргументов в условии `WHERE` (завершающий массив `String[]`).

Почему мы не вставили `uuidString` прямо в условие `WHERE`? Ведь это проще, чем использовать `?` и передавать значение в `String[]`. Дело в том, что в некоторых случаях сама строка может содержать код SQL. Если вставить ее содержимое прямо в запрос, этот код может изменить смысл запроса или даже модифицировать базу данных. Подобные ситуации, называемые *атаками внедрения SQL*, безусловно нежелательны. Но при использовании `?` ваш код будет делать именно то, что положено: значение интерпретируется как строковые данные, а не как код. Таким образом, лучше перестраховаться и привыкнуть к использованию заполнителя `?`,

который всегда приведет к желаемому результату независимо от содержимого строки.

Экземпляры `Crime`, изменяемые в `CrimeFragment`, должны быть записаны в базу данных при завершении `CrimeFragment`. Добавьте переопределение `CrimeFragment.onPause()`, которое обновляет копию `Crime` из `CrimeLab`.

Листинг 14.11. Запись обновлений (`CrimeFragment.java`)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
}

@Override
public void onPause() {
    super.onPause();

    CrimeLab.get(getActivity())
        .updateCrime(mCrime);
}
```

К сожалению, проверить работоспособность этого кода пока не удастся. С проверкой придется подождать, пока приложение не научится читать обновленные данные. Чтобы убедиться в том, что программа нормально компилируется, запустите `CriminalIntent` еще один раз, прежде чем переходить к следующему разделу. В приложении должен отображаться пустой список.

Чтение из базы данных

Чтение данных из `SQLite` осуществляется методом `query(...)`. При вызове `SQLiteDatabase.query(...)` происходит много всего; метод существует в нескольких перегруженных версиях. Версия, которую вы будете использовать, выглядит так:

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

Если у вас уже есть опыт работы с `SQL`, многие имена покажутся знакомыми по аргументам команды `SELECT`. Если же вы не имели дела с `SQL`, ограничьтесь только теми аргументами, которые вы будете использовать:

```
public Cursor query(
    String table,
    String[] columns,
    String where,
    String[] whereArgs,
    String groupBy,
    String having,
    String orderBy,
    String limit)
```

Аргумент `table` содержит таблицу, к которой обращен запрос. Аргумент `columns` определяет столбцы, значения которых вам нужны, и порядок их извлечения. Наконец, `where` и `whereArgs` работают так же, как и в команде `update(...)`.

Создайте вспомогательный метод, в котором будет вызываться метод `query(...)` для таблицы `CrimeTable`.

Листинг 14.12. Запрос для получения данных `Crime` (`CrimeLab.java`)

```
...
    values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
    values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
    return values;
}

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
    Cursor cursor = mDatabase.query(
        CrimeTable.NAME,
        null, // Columns - null выбирает все столбцы
        whereClause,
        whereArgs,
        null, //groupBy
        null, // having
        null // orderBy
    );
    return cursor;
}
```

Использование `CursorWrapper`

Класс `Cursor` как средство работы с данными таблиц оставляет желать лучшего. По сути, он просто возвращает низкоуровневые значения столбцов. Процедура получения данных из `Cursor` выглядит так:

```
String uuidString = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.UUID));
String title = cursor.getString(
    cursor.getColumnIndex(CrimeTable.Cols.TITLE));
long date = cursor.getLong(
    cursor.getColumnIndex(CrimeTable.Cols.DATE));
int isSolved = cursor.getInt(
    cursor.getColumnIndex(CrimeTable.Cols.SOLVED));
```

Каждый раз, когда вы извлекаете Crime из курсора, этот код придется записывать снова. (Не говорю уже о коде создания экземпляра Crime с этими значениями!)

Вспомните правило DRY: «Не повторяйтесь». Вместо того чтобы записывать этот код при каждом чтении данных из курсора, вы можете создать собственный субкласс Cursor, который выполняет эту операцию в одном месте. Для написания субкласса курсора проще всего воспользоваться CursorWrapper — этот класс позволяет дополнить класс Cursor, полученный извне, новыми методами.

Создайте новый класс в пакете базы данных с именем CrimeCursorWrapper.

Листинг 14.13. Создание CrimeCursorWrapper (CrimeCursorWrapper.java)

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }
}
```

Класс создает тонкую «обертку» для Cursor. Он содержит те же методы, что и инкапсулированный класс Cursor, и вызов этих методов приводит ровно к тем же последствиям. Все это не имело бы смысла, если бы не возможность добавления новых методов для работы с инкапсулированным классом Cursor.

Добавьте метод getCrime() для извлечения данных столбцов. (Не забудьте использовать для CrimeTable прием импортирования, состоящий из двух шагов, как это было сделано ранее.)

Листинг 14.14. Добавление метода getCrime() (CrimeCursorWrapper.java)

```
public class CrimeCursorWrapper extends CursorWrapper {
    public CrimeCursorWrapper(Cursor cursor) {
        super(cursor);
    }

    public Crime getCrime() {
        String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
        String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
        long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
        int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

        return null;
    }
}
```

Метод должен возвращать объект Crime с соответствующим значением UUID. Добавьте в Crime конструктор для выполнения этой операции.

Листинг 14.15. Добавление конструктора Crime (Crime.java)

```
public Crime() {
    this(UUID.randomUUID());
    mId = UUID.randomUUID();
    mDate = new Date();
}
```

```

}

public Crime(UUID id) {
    mId = id;
    mDate = new Date();
}

```

А затем доработайте метод `getCrime()`.

Листинг 14.16. Окончательная версия `getCrime()` (`CrimeCursorWrapper.java`)

```

public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));

    Crime crime = new Crime(UUID.fromString(uuidString));
    crime.setTitle(title);
    crime.setDate(new Date(date));
    crime.setSolved(isSolved != 0);

    return crime;
return null;
}

```

(Android Studio предлагает выбрать между `java.util.Date` и `java.sql.Date`. И хотя вы работаете с базой данных, здесь следует выбрать `java.util.Date`.)

Преобразование в объекты модели

С `CrimeCursorWrapper` процесс получения `List<Crime>` от `CrimeLab` достаточно прямолинейен. Курсор, полученный при запросе, упаковывается в `CrimeCursorWrapper`, после чего его содержимое перебирается методом `getCrime()` для получения объектов `Crime`.

Для первой части переведите `queryCrimes(...)` на использование `CrimeCursorWrapper`.

Листинг 14.17. Использование `CursorWrapper` (`CrimeLab.java`)

```

private Cursor queryCrimes(String whereClause, String[] whereArgs) {
private CrimeCursorWrapper queryCrimes(String whereClause, String[] whereArgs) {
    Cursor cursor = mDatabase.query(
        CrimeTable.NAME,
        null, // Columns - null выбирает все столбцы
        whereClause,
        whereArgs,
        null, // groupBy
        null, // having
        null // orderBy
    );
}

```

```

return cursor;
return new CrimeCursorWrapper(cursor);
}

```

Метод `getCrimes()` принял нужную форму. Добавьте код запроса всех преступлений, организуйте перебор курсора и заполнение списка `Crime`.

Листинг 14.18. Возвращение списка преступлений (`CrimeLab.java`)

```

public List<Crime> getCrimes() {
return new ArrayList<>();
    List<Crime> crimes = new ArrayList<>();

    CrimeCursorWrapper cursor = queryCrimes(null, null);

    try {
        cursor.moveToFirst();
        while (!cursor.isAfterLast()) {
            crimes.add(cursor.getCrime());
            cursor.moveToNext();
        }
    } finally {
        cursor.close();
    }

    return crimes;
}

```

Курсоры базы данных всегда устанавливаются в определенную позицию в результатах запроса. Таким образом, чтобы извлечь данные из курсора, его следует перевести к первому элементу вызовом `moveToFirst()`, а затем прочитать данные строки. Каждый раз, когда потребуется перейти к следующей записи, мы вызываем `moveToNext()`, пока `isAfterLast()`, наконец, не сообщит, что указатель вышел за пределы набора данных.

Последнее, что осталось сделать, — вызвать `close()` для объекта `Cursor`. Не забывайте об этой служебной операции, это важно. Если вы забудете закрыть курсор, устройство Android начнет выдавать в журнал сообщения об ошибках. Что еще хуже, если ваша забывчивость будет проявляться хронически, со временем это приведет к исчерпанию файловых дескрипторов и сбою приложения. Итак, помните: курсоры нужно закрывать.

Метод `CrimeLab.getCrime(UUID)` похож на `getCrimes()`, не считая того, что он должен извлечь только первый элемент данных (если тот присутствует).

Листинг 14.19. Переработка `getCrime(UUID)` (`CrimeLab.java`)

```

public Crime getCrime(UUID id) {
return null;
    CrimeCursorWrapper cursor = queryCrimes(
        CrimeTable.Cols.UUID + " = ?",

```

```

        new String[] { id.toString() }
    );
    try {
        if (cursor.getCount() == 0) {
            return null;
        }

        cursor.moveToFirst();
        return cursor.getCrime();
    } finally {
        cursor.close();
    }
}

```

Нам удалось сделать следующее:

- ❶ Вы можете вставлять новые преступления, так что код, добавляющий  в CrimeLab при нажатии элемента действия New Crime, теперь работает.
- ❷ Приложение обращается с запросами к базе данных, так что  видит все объекты Crime в CrimeLab.
- ❸ Метод CrimeLab.getCrime(UUID) тоже работает, так что каждый  CrimeFragment, отображаемый в CrimePagerActivity, отображает существующий объект Crime.

Теперь при нажатии New Crime в CrimePagerActivity появляется новый объект Crime.

Запустите CrimInalIntent и убедитесь в том, что эта функциональность работает. Если что-то пошло не так, перепроверьте свои реализации из этой главы.

Обновление данных модели

Впрочем, работа еще не закончена. Преступления сохраняются в базе данных, но данные не читаются из нее. Таким образом, если вы нажмете кнопку Back в процессе редактирования нового преступления, оно не появится в CrimeListActivity.

Дело в том, что CrimeLab теперь работает немного иначе. Прежде существовал только один список List<Crime> и один объект для каждого преступления: тот, что содержится в List<Crime>. Это объяснялось тем, что переменная mCrimes была единственным авторитетным источником данных о преступлениях, известных вашему приложению.

Сейчас ситуация изменилась. Переменная mCrimes исчезла, так что список List<Crime>, возвращаемый getCrimes(), представляет собой «моментальный снимок» данных Crime на некоторый момент времени. Чтобы обновить CrimeListActivity, необходимо обновить этот снимок.

Большинство необходимых составляющих уже готово. CrimeListActivity уже вызывает updateUI() для обновления других частей интерфейса. Остается лишь заставить этот метод обновить его представление CrimeLab.

Сначала добавьте в `CrimeAdapter` метод `setCrimes(List<Crime>)`, чтобы закрепить отображаемые в нем данные.

Листинг 14.20. Добавление `setCrimes(List<Crime>)` (`CrimeListFragment.java`)

```
private class CrimeAdapter extends RecyclerView.Adapter<CrimeHolder> {  
    ...  
  
    @Override  
    public int getItemCount() {  
        return mCrimes.size();  
    }  
  
    public void setCrimes(List<Crime> crimes) {  
        mCrimes = crimes;  
    }  
}
```

Вызовите `setCrimes(List<Crime>)` в `updateUI()`.

Листинг 14.21. Вызов `setCrime(List<>)` (`CrimeListFragment.java`)

```
private void updateUI() {  
    CrimeLab crimeLab = CrimeLab.get(getActivity());  
    List<Crime> crimes = crimeLab.getCrimes();  
  
    if (mAdapter == null) {  
        mAdapter = new CrimeAdapter(crimes);  
        mCrimeRecyclerView.setAdapter(mAdapter);  
    } else {  
        mAdapter.setCrimes(crimes);  
        mAdapter.notifyDataSetChanged();  
    }  
  
    updateSubtitle();  
}
```

Теперь все должно работать правильно. Запустите `CriminalIntent` и убедитесь в том, что вы можете добавить преступление, нажать кнопку `Back`, и это преступление появится в `CrimeListActivity`.

Заодно можно проверить, что вызовы `updateCrime(Crime)` в `CrimeFragment` работают. Нажмите на преступлении и отредактируйте его краткое описание в `CrimePagerActivity`. Нажмите кнопку `Back` и убедитесь в том, что новый текст появился в списке.

Для любознательных: другие базы данных

Ради простоты и краткости мы не стали углубляться во все подробности, которые могут встретиться при работе с базами данных в профессиональном приложении.

Разработчики не зря пользуются такими инструментами, как ORM: все это может оказаться весьма непростым делом.

В более серьезном приложении в базах данных обычно реализуются следующие аспекты:

- ❶ *Типы данных столбцов.* Строго говоря, в SQLite типизация столбцов отсутствует, так что вы можете обойтись без этой информации. Впрочем, дать SQLite полезную подсказку не помешает.
- ❷ *Индексы.* Запросы к столбцам, для которых построены индексы, намного быстрее, чем запросы к неиндексированным столбцам.
- ❸ *Внешние ключи.* В нашем примере база данных содержит всего одну таблицу, но в реальных приложениях могут понадобиться ограничения внешнего ключа.

Также стоит учитывать и более глубокие факторы эффективности. Ваше приложение при каждом запросе к базе данных создает новый список объектов `Crime`. Действительно эффективное приложение оптимизирует такие обращения: оно использует заново старые экземпляры `Crime` или организует из них хранилище объектов в памяти (как это делалось в предшествующих главах). Это приводит к увеличению объема кода — еще одной проблеме, которую часто пытаются решить инструменты ORM.

Для любознательных: контекст приложения

Ранее в этой главе при вызове конструктора `CrimeLab` использовался *контекст приложения*:

```
private CrimeLab(Context context) {
    mContext = context.getApplicationContext();
    ...
}
```

Почему именно контекст приложения? Когда следует использовать его вместо активности в качестве контекста?

Важно учитывать жизненный цикл каждого из этих объектов. Если в приложении существуют какие-либо из ваших активностей, `Android` также создает объект `Application`. Активности появляются и исчезают в процессе работы пользователя с приложением, но объект `Application` продолжает существовать. Его срок жизни значительно превышает срок жизни любой отдельной активности.

Объект `CrimeLab` является синглетом, то есть после того, как он будет создан, он продолжит существовать до тех пор, пока не будет уничтожен весь процесс приложения. В `CrimeLab` хранится ссылка на объект `mContext`.

Если сохранить в `mContext` активность, эта активность никогда не будет уничтожена уборщиком мусора, потому что ссылка на нее хранится в `CrimeLab`. Даже если пользователь покинет активность, она не уничтожается.

Чтобы избежать столь неэффективного поведения, мы используем контекст приложения: активности приходят и уходят, а CrimeLab поддерживает ссылку на объект Context. Всегда учитывайте срок жизни своих активностей, если вы сохраняете ссылки на них.

Упражнение. Удаление преступлений

Если ранее вы добавили элемент действия Delete Crime, в этом упражнении он дополняется возможностью удаления информации из базы данных. Для этого вызывается метод `deleteCrime(Crime)` для CrimeLab, который вызывает метод `mDatabase.delete(...)` для завершения работы.

А если элемент действия отсутствует? Так добавьте его! Добавьте на панель инструментов CrimeFragment элемент действия, который вызывает метод CrimeLab.`deleteCrime(Crime)` с последующим вызовом `finish()` для его активности.

Модуль 3. Решение прикладных задач на мобильных устройствах

3.1. Добавление кнопок

Начнем с включения в макеты CrimeFragment новых кнопок. Прежде всего добавьте строки, которые будут отображаться на кнопках.

Листинг 15.1. Добавление строк для надписей на кнопках (strings.xml)

```
...
<string name="subtitle_format">%1$s crimes</string>
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
</resources>
```

Добавьте в файл layout/fragment_crime.xml два виджета Button, представленных на рис. 15.2. Обратите внимание: на диаграмме не показан первый виджет

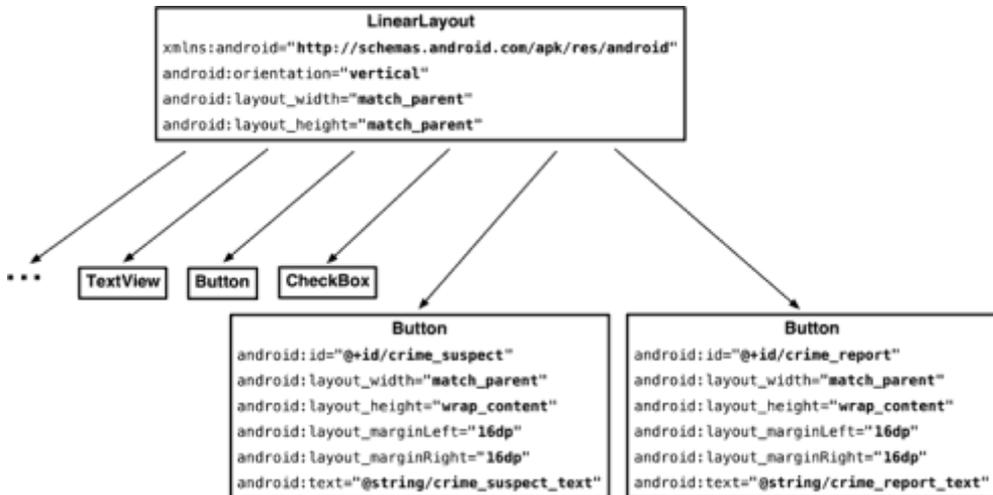


Рис. 15.2. Добавление кнопок для выбора контактов и отправки отчетов (layout/fragment_crime.xml)

LinearLayout и все его потомки, чтобы вы могли сосредоточиться на новых и интересных частях диаграммы.

В альбомном макете мы назначим новые кнопки потомками нового горизонтального виджета LinearLayout, расположенного под виджетом с кнопкой даты и флажком. Новый макет изображен на рис. 15.3.

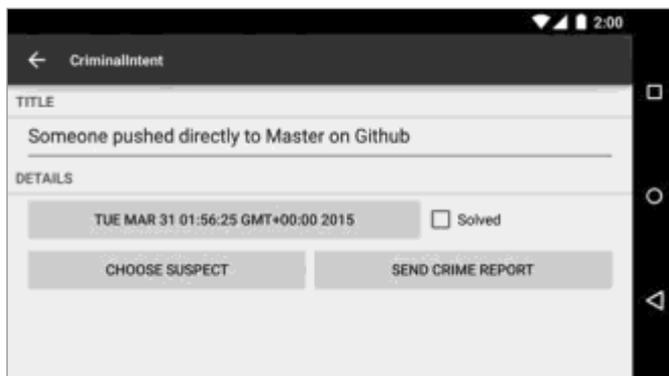


Рис. 15.3. Новый альбомный макет

Добавьте в файл `layout-land/fragment_crime.xml` виджет `LinearLayout` и два виджета кнопок, как показано на рис. 15.4.

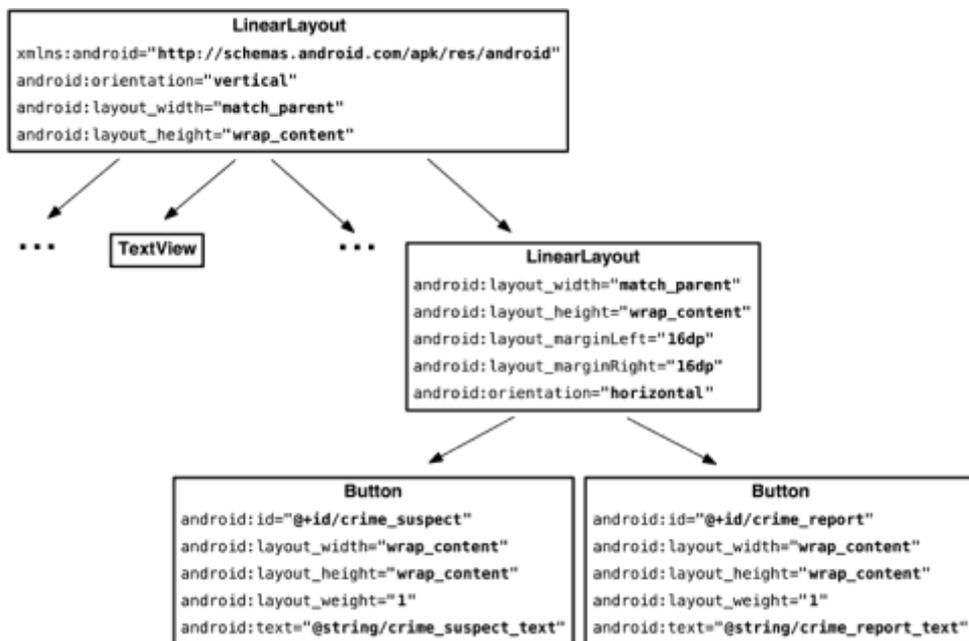


Рис. 15.4. Добавление кнопок для выбора контактов и отправки отчетов (`layout-land/fragment_crime.xml`)

На этой стадии вы можете проверить макеты в области предварительного просмотра или запустить приложение `CriminalIntent`, чтобы убедиться в правильности расположения новых кнопок.

Добавление подозреваемого в уровень модели

Откройте файл `Crime.java` и добавьте новую переменную для хранения имени подозреваемого.

Листинг 15.2. Добавление поля для имени подозреваемого (`Crime.java`)

```
public class Crime {
    ...
    private boolean mSolved;
    private String mSuspect;

    public Crime() {
        this(UUID.randomUUID());
    }
    ...

    public void setSolved(boolean solved) {
        mSolved = solved;
    }

    public String getSuspect() {
        return mSuspect;
    }

    public void setSuspect(String suspect) {
        mSuspect = suspect;
    }
}
```

Затем дополнительное поле добавляется в базу данных. Сначала добавьте в `CrimeDbSchema` столбец `suspect`.

Листинг 15.3. Добавление столбца `suspect` (`CrimeDbSchema.java`)

```
public class CrimeDbSchema {
    public static final class CrimeTable {
        public static final String NAME = "crimes";

        public static final class Cols {
            public static final String UUID = "uuid";
            public static final String TITLE = "title";
            public static final String DATE = "date";
            public static final String SOLVED = "solved";
        }
    }
}
```

```

        public static final String SUSPECT = "suspect";
    }
}

```

Также добавьте столбец в CrimeBaseHelper. (Обратите внимание: новый код начинается с запятой после CrimeTable.Cols.SOLVED.)

Листинг 15.4. Добавление столбца suspect в другом месте (CrimeBaseHelper.java)

```

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("create table " + CrimeTable.NAME + "(" +
        " _id integer primary key autoincrement, " +
        CrimeTable.Cols.UUID + ", " +
        CrimeTable.Cols.TITLE + ", " +
        CrimeTable.Cols.DATE + ", " +
        CrimeTable.Cols.SOLVED + ", " +
        CrimeTable.Cols.SUSPECT +
        ")");
}

```

Запишите новый столбец в CrimeLab.getContentValues(Crime).

Листинг 15.5. Запись в столбец suspect (CrimeLab.java)

```

---
private static ContentValues getContentValues(Crime crime) {
    ContentValues values = new ContentValues();
    values.put(CrimeTable.Cols.UUID, crime.getId().toString());
    values.put(CrimeTable.Cols.TITLE, crime.getTitle());
    values.put(CrimeTable.Cols.DATE, crime.getDate().getTime());
    values.put(CrimeTable.Cols.SOLVED, crime.isSolved() ? 1 : 0);
    values.put(CrimeTable.Cols.SUSPECT, crime.getSuspect());

    return values;
}
---

```

Теперь прочитайте из него данные в CrimeCursorWrapper.

Листинг 15.6. Чтение из столбца suspect (CrimeCursorWrapper.java)

```

---
public Crime getCrime() {
    String uuidString = getString(getColumnIndex(CrimeTable.Cols.UUID));
    String title = getString(getColumnIndex(CrimeTable.Cols.TITLE));
    long date = getLong(getColumnIndex(CrimeTable.Cols.DATE));
    int isSolved = getInt(getColumnIndex(CrimeTable.Cols.SOLVED));
    String suspect = getString(getColumnIndex(CrimeTable.Cols.SUSPECT));

    Crime crime = new Crime(UUID.fromString(uuidString));

```

```
        crime.setTitle(title);
        crime.setDate(new Date(date));
        crime.setSolved(isSolved != 0);
        crime.setSuspect(suspect);
    }
    return crime;
}
```

Если приложение `CriminalIntent` уже установлено на вашем устройстве, существующая база данных не содержит столбец `suspect`, и новый метод `onCreate(SQLiteDatabase)` не будет выполнен для добавления нового столбца. В такой ситуации проще всего стереть старую базу данных для создания новой. (Это довольно часто происходит при разработке приложений.)

Сначала удалите приложение `CriminalIntent`: откройте экран лаунчера и перетащите значок `CriminalIntent` в верхнюю часть экрана. В процессе удаления будут уничтожены все данные приложения в песочнице вместе с устаревшей схемой базы данных. Затем запустите `CriminalIntent` из `Android Studio`. Новая база данных создается вместе с новым столбцом в процессе установки приложения.

Форматные строки

Последним подготовительным шагом станет создание шаблона отчета о преступлении, который заполняется информацией о конкретном преступлении. Так как подробная информация недоступна до стадии выполнения, необходимо использовать форматную строку с заполнителями, которые будут заменяться во время выполнения. Форматная строка будет выглядеть так:

```
<string name="crime_report">%1$s! The crime was discovered on %2$s. %3$s,
    and %4$s
```

Поля `%1$s`, `%2$s` и т. д. — заполнители для строковых аргументов. В коде вы вызываете `getString(...)` и передаете форматную строку и еще четыре строки в том порядке, в каком они должны заменять заполнители.

Сначала добавьте в `strings.xml` строки из листинга 15.7.

Листинг 15.7. Добавление строковых ресурсов (`strings.xml`)

```
<string name="crime_suspect_text">Choose Suspect</string>
<string name="crime_report_text">Send Crime Report</string>
<string name="crime_report">%1$s!
    The crime was discovered on %2$s. %3$s, and %4$s
</string>
<string name="crime_report_solved">The case is solved</string>
<string name="crime_report_unsolved">The case is not solved</string>
<string name="crime_report_no_suspect">there is no suspect.</string>
<string name="crime_report_suspect">the suspect is %s.</string>
<string name="crime_report_subject">CriminalIntent Crime Report</string>
```

```
<string name="send_report">Send crime report via</string>
</resources>
```

В файле `CrimeFragment.java` добавьте метод, который создает четыре строки, соединяет их и возвращает полный отчет.

Листинг 15.8. Добавление метода `getCrimeReport()` (`CrimeFragment.java`)

```
...
private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}

private String getCrimeReport() {
    String solvedString = null;
    if (mCrime.isSolved()) {
        solvedString = getString(R.string.crime_report_solved);
    } else {
        solvedString = getString(R.string.crime_report_unsolved);
    }

    String dateFormat = "EEE, MMM dd";
    String dateString = DateFormat.format(dateFormat,
        mCrime.getDate().toString());

    String suspect = mCrime.getSuspect();
    if (suspect == null) {
        suspect = getString(R.string.crime_report_no_suspect);
    } else {
        suspect = getString(R.string.crime_report_suspect, suspect);
    }

    String report = getString(R.string.crime_report,
        mCrime.getTitle(), dateString, solvedString, suspect);

    return report;
}
```

(Обратите внимание: класс `DateFormat` существует в двух версиях, `android.text.format.DateFormat` и `java.text.DateFormat`. Используйте `android.text.format.DateFormat`.)

Приготовления завершены, теперь можно непосредственно заняться неявными интен­тами.

Использование неявных интен­тов

Объект `Intent` описывает для ОС некую операцию, которую вы хотите выполнить. Для *явных* интен­тов, использовавшихся до настоящего момента, разработчик явно указывает активность, которую должна запустить ОС.

```
Intent intent = new Intent(getActivity(), CrimePagerActivity.class);
intent.putExtra(EXTRA_CRIME_ID, crimeId);
startActivity(intent);
```

Для *неявных* интенгов разработчик описывает выполняемую операцию, а ОС запускает активность, которая ранее сообщила о том, что она способна выполнять эту операцию. Если ОС находит несколько таких активностей, пользователю предлагается выбрать нужную.

Строение неявного интенга

Ниже перечислены важнейшие составляющие интенга, используемые для определения выполняемой операции.

- ① Выполняемое *действие* (action) — обычно определяется константами из класса Intent. Так, для просмотра URL-адреса используется константа Intent.ACTION_VIEW, а для отправки данных — константа Intent.ACTION_SEND.
- ① Местонахождение *данных* — это может быть как ссылка на данные, находящиеся за пределами устройства (скажем, URL веб-страницы), так и URI файла или URI контента, ссылающийся на запись ContentProvider.
- ① *Тип* данных, с которыми работает действие, — тип MIME (например, `audio/mpeg3`). Если в интенг включено местонахождение данных, то тип обычно удается определить по этим данным.
- ① Необязательные *категории* — если действие указывает, что нужно сделать, категория обычно описывает, где, когда или как вы пытаетесь использовать операцию. Android использует категорию `android.intent.category.LAUNCHER` для обозначения активностей, которые должны отображаться в лаунчере приложений верхнего уровня. С другой стороны, категория `android.intent.category.INFO` обозначает активность, которая выдает пользователю информацию о папке, но не отображается в лаунчере.

Простой неявный интенг для просмотра веб-сайта включает действие Intent.ACTION_VIEW и объект данных Uri с URL-адресом сайта.

На основании этой информации ОС запускает соответствующую активность соответствующего приложения. (Если ОС обнаруживает более одного кандидата, пользователю предлагается принять решение.)

Активность сообщает о себе как об исполнителе для ACTION_VIEW при помощи фильтра интенгов в манифесте. Например, если вы пишете приложение-браузер, вы включаете следующий фильтр интенгов в объявление активности, реагирующей на ACTION_VIEW.

```
<activity
  android:name=".BrowserActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category android:name="android.intent.category.DEFAULT" />
```

```
<data android:scheme="http" android:host="www.bignerdranch.com" />
</intent-filter>
</activity>
```

Категория DEFAULT должна явно задаваться в фильтрах интентов. Элемент action в фильтре интентов сообщает ОС, что активность способна выполнять операцию, а категория DEFAULT — что она желает рассматриваться среди кандидатов на выполнение операции. Категория DEFAULT неявно добавляется к почти любому неявному интенту. (Единственное исключение составляет категория LAUNCHER, с которой мы будем работать в главе 22.)

Неявные интенты, как и явные, также могут включать дополнения. Однако дополнения неявного интенга не используются ОС для поиска соответствующей активности.

Также следует отметить, что компоненты действия и данных интенга могут использоваться в сочетании с явными интентами. Результат равнозначен тому, как если бы вы приказали конкретной активности выполнить конкретную операцию.

Отправка отчета

Чтобы увидеть на практике, как работает эта схема, мы создадим неявный интенг для отправки отчета о преступлении в приложении CriminalIntent. Операция, которую нужно выполнить, — отправка простого текста; отчет представляет собой строку. Таким образом, действие неявного интенга будет представлено константой ACTION_SEND. Интенг не содержит ссылок на данные и не имеет категорий, но определяет тип text/plain.

В методе CrimeFragment.onCreateView(...) получите ссылку на кнопку Send Crime Report и назначьте для нее слушателя. В реализации слушателя создайте неявный интенг и передайте его startActivity(Intent).

Листинг 15.9. Отправка отчета о преступлении (CrimeFragment.java)

```
private Crime mCrime;
private EditText mTitleField;
private Button mDateButton;
private CheckBox mSolvedCheckbox;
private Button mReportButton;
...

public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...

    mReportButton = (Button) v.findViewById(R.id.crime_report);
    mReportButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            Intent i = new Intent(Intent.ACTION_SEND);
            i.setType("text/plain");
            i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
```

```
        i.putExtra(Intent.EXTRA_SUBJECT,
                   getString(R.string.crime_report_subject));
        startActivity(i);
    });
}

return v;
}
```

Здесь мы используем конструктор `Intent`, который получает строку с константой, описывающей действие. Также существуют другие конструкторы, которые могут использоваться в зависимости от вида создаваемого неявного интенга. Информацию о них можно найти в справочной документации `Intent`. Конструктора, получающего тип, не существует, поэтому мы задаем его явно.

Текст отчета и строка темы включаются в дополнения. Обратите внимание на использование в них констант, определенных в классе `Intent`. Любая активность, реагирующая на интенг, знает эти константы и то, что следует делать с ассоциированными значениями.

Запустите приложение `CriminalIntent` и нажмите кнопку `Send Crime Report`. Так как этот интенг с большой вероятностью совпадет со многими активностями на устройстве, скорее всего, на экране появится список активностей (рис. 15.5).

Если на экране появился список, выберите нужный вариант. Вы увидите, что отчет о преступлении загружается в выбранном вами приложении. Вам остается лишь ввести адрес и отправить его.

Если список не появился, это может означать одно из двух: либо вы уже назначили приложение по умолчанию для идентичного неявного интенга, либо на вашем устройстве имеется всего одна активность, способная реагировать на этот интенг.

Часто лучшим вариантом оказывается использование приложения по умолчанию, выбранного пользователем для действия `ACTION_SEND`. Впрочем, в приложении `CriminalIntent` лучше всегда предоставлять пользователю выбор: сегодня пользователь предпочтет не поднимать шум и отправит отчет по электронной почте, а завтра выставит нарушителя на общественное осуждение в Твиттере.

Вы можете создать список, который будет отображаться каждый раз при использовании неявного интенга для запуска активности. После создания неявного интенга способом, показанным ранее, вы вызываете следующий метод `Intent` и передаете ему неявный интенг и строку с заголовком:

```
public static Intent createChooser(Intent target, String title)
```

Затем интенг, возвращенный `createChooser(...)`, передается `startActivity(...)`.

В файле `CrimeFragment.java` создайте список выбора для отображения активностей, реагирующих на неявный интенг.

Листинг 15.10. Использование списка выбора (`CrimeFragment.java`)

```
public void onClick(View v) {
    Intent i = new Intent(Intent.ACTION_SEND);
```

```

i.setType("text/plain");
i.putExtra(Intent.EXTRA_TEXT, getCrimeReport());
i.putExtra(Intent.EXTRA_SUBJECT,
           getString(R.string.crime_report_subject));
i = Intent.createChooser(i, getString(R.string.send_report));
startActivity(i);
}

```

Запустите приложение CriminalIntent и нажмите кнопку Send Crime Report. Если в системе имеется несколько активностей, способных обработать ваш интент, на экране появляется список для выбора (рис. 15.6).

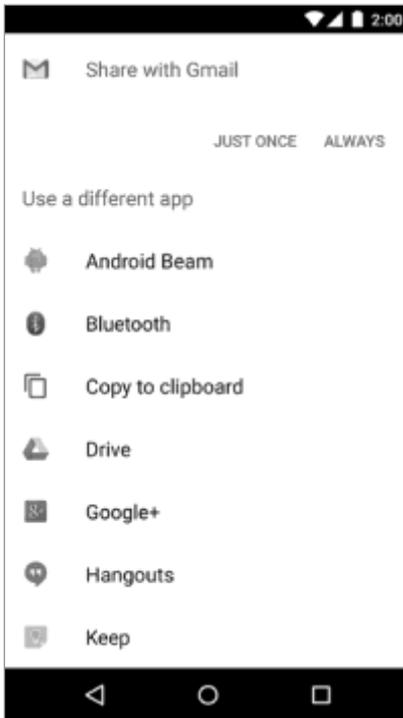


Рис. 15.5. Приложения, готовые отправить ваш отчет

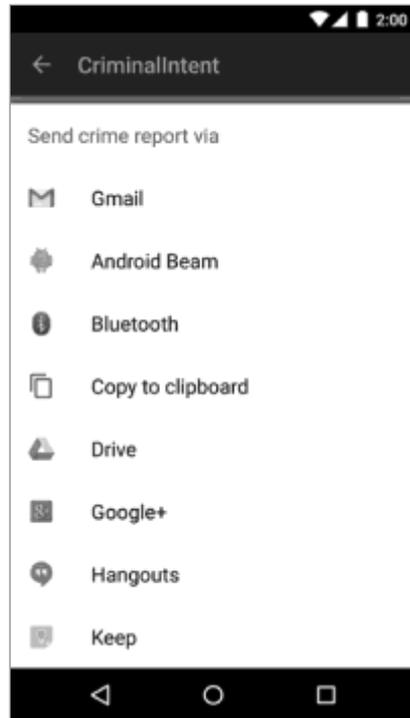


Рис. 15.6. Отправка текста с выбором активности

Запрос контакта у Android

Теперь мы создадим другой неявный интент, который предлагает пользователю выбрать подозреваемого из списка контактов. Для этого неявного интента будет определено действие и местонахождение соответствующих данных. Действие задается константой `Intent.ACTION_PICK`, а местонахождение данных — `ContactsContract.Contacts.CONTENT_URI`. Короче говоря, вы просите Android помочь с выбором записи из базы данных контактов.

Запущенная активность должна вернуть результат, поэтому мы передаем интент через `startActivityResult(...)` вместе с кодом запроса. Добавьте в файл `CrimeFragment.java` константу для кода запроса и поле для кнопки.

Листинг 15.11. Добавление поля для кнопки подозреваемого (`CrimeFragment.java`)

```
...
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
...

private CheckBox mSolvedCheckbox;
private Button mSuspectButton;
...
```

В конце `onCreateView(...)` получите ссылку на кнопку и назначьте ей слушателя. В реализации слушателя создайте неявный интент и передайте его `startActivityResult(...)`. Также выведите на кнопке имя подозреваемого (если оно содержится в `Crime`).

Листинг 15.12. Отправка неявного интента (`CrimeFragment.java`)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...

    final Intent pickContact = new Intent(Intent.ACTION_PICK,
        ContactsContract.Contacts.CONTENT_URI);
    mSuspectButton = (Button) v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivityResult(pickContact, REQUEST_CONTACT);
        }
    });

    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    return v;
}
```

Мы еще воспользуемся интентом `pickContact`, поэтому он размещается за пределами слушателя `OnClickListener` кнопки `mSuspectButton`.

Запустите приложение `CriminalIntent` и нажмите кнопку `ChooseSuspect`. На экране появляется список контактов (рис. 15.7).

Если у вас установлено другое контактное приложение, экран будет выглядеть иначе. Это еще одно преимущество неявных интентов: вам не нужно знать название контактного приложения, чтобы использовать его из своего приложения. Соответственно пользователь может установить то приложение, которое считает нужным, а ОС найдет и запустит его.

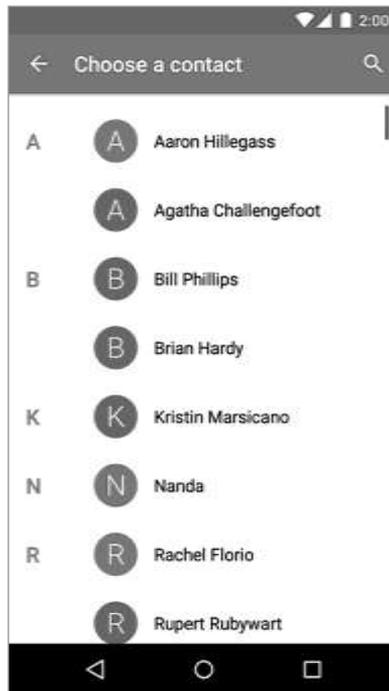


Рис. 15.7. Список подозреваемых

Получение данных из списка контактов

Теперь необходимо получить результат от контактного приложения. Контактная информация совместно используется многими приложениями, поэтому Android предоставляет расширенный API для работы с контактными данными через ContentProvider. Экземпляры этого класса инкапсулируют базы данных и предоставляют доступ к ним другим приложениям. Обращение к ContentProvider осуществляется через ContentResolver.

Так как активность запускалась с возвращением результата с использованием ACTION_PICK, вы можете получить интент вызовом onActivityResult(...). Интент включает URI данных — ссылку на конкретный контакт, выбранный пользователем.

В файле CrimeFragment.java добавьте следующий код в реализацию onActivityResult(...) из CrimeFragment.

Листинг 15.13. Получение имени контакта (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {
        return;
    }
    if (requestCode == REQUEST_DATE) {
```

```

    ...
    updateDate();
} else if (requestCode == REQUEST_CONTACT && data != null) {
    Uri contactUri = data.getData();

    // Определение полей, значения которых должны быть
    // возвращены запросом.
    String[] queryFields = new String[] {
        ContactsContract.Contacts.DISPLAY_NAME
    };
    // Выполнение запроса - contactUri здесь выполняет функции
    // условия "where"
    Cursor c = getActivity().getContentResolver()
        .query(contactUri, queryFields, null, null, null);

    try {
        // Проверка получения результатов
        if (c.getCount() == 0) {
            return;
        }
        // Извлечение первого столбца данных - имени подозреваемого.
        c.moveToFirst();
        String suspect = c.getString(0);
        mCrime.setSuspect(suspect);
        mSuspectButton.setText(suspect);
    } finally {
        c.close();
    }
}
}
}
}

```

В листинге 15.13 создается запрос, который запрашивает все отображаемые имена контактов из возвращаемых данных. Затем мы выдаем запрос к базе данных контактов и получаем объект `Cursor` для работы с ней. Так как мы знаем, что курсор содержит всего один элемент, мы переходим к первому элементу и используем его как строку. Эта строка содержит имя подозреваемого, которое мы используем для задания подозреваемого в `Crime` и текста кнопки `Choose Suspect`.

(База данных контактов сама по себе является достаточно обширной темой. Здесь она не рассматривается. Если вам захочется узнать больше, обратитесь к руководству по `Contacts Provider API`: <http://developer.android.com/guide/topics/providers/contacts-provider.html>.)

Запустите приложение. На некоторых устройствах нет контактного приложения, которое могло бы использоваться для тестирования. В этом случае воспользуйтесь эмулятором.

Разрешения контактов

Как получить разрешение на чтение из базы данных контактов? Контактное приложение распространяет свои разрешения на вас. Оно обладает полными разреше-

ниями на обращение к базе данных. Когда контактное приложение возвращает родительской активности URI данных в интенте, оно также добавляет флаг `Intent.FLAG_GRANT_READ_URI_PERMISSION`. Этот флаг сообщает Android, что родительской активности в `CriminalIntent` следует разрешить однократное использование этих данных. Такой подход работает хорошо, потому что фактически нам нужен доступ не ко всей базе данных контактов, а к одному контакту в этой базе.

Проверка реагирующих активностей

На первый неявный интент, созданный в этой главе, кто-то гарантированно отреагирует: даже если способа отправки отчета не существует, окно выбора все равно будет отображено. Со вторым интентом дело обстоит иначе: на некоторых устройствах (или у некоторых пользователей) может не оказаться контактного приложения. Если ОС не найдет подходящую активность, в приложении происходит сбой.

Проблема решается предварительной проверкой того, от какой части ОС поступил вызов `PackageManager`. Это удобно сделать в `onCreateView(...)`.

Листинг 15.14. Защита от отсутствия контактных приложений (`CrimeFragment.java`)

```
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...

    if (mCrime.getSuspect() != null) {
        mSuspectButton.setText(mCrime.getSuspect());
    }

    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
        PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    return v;
}
```

`PackageManager` известно все о компонентах, установленных на устройстве Android, включая все его активности. (Другие компоненты встретятся вам позднее в этой книге.) Вызывая `resolveActivity(Intent, int)`, вы приказываете найти активность, соответствующую переданному интенту.

Флаг `MATCH_DEFAULT_ONLY` ограничивает поиск активностями с флагом `CATEGORY_DEFAULT` (по аналогии с `startActivity(Intent)`).

Если поиск прошел успешно, возвращается экземпляр `ResolveInfo`, который сообщает полную информацию о найденной активности. С другой стороны, если поиск вернул `null`, все кончено — контактного приложения нет, поэтому бесполезная кнопка просто блокируется.

Если вы хотите убедиться в том, что фильтр работает, но не располагаете устройством без контактного приложения, временно добавьте в интент дополнительную категорию. Эта категория ничего не делает, а только предотвращает возможные совпадения контактных приложений с вашим интентом.

Листинг 15.15. Фиктивный код для проверки фильтра (CrimeFragment.java)

```
...  
  
final Intent pickContact = new Intent(Intent.ACTION_PICK,  
    ContactsContract.Contacts.CONTENT_URI);  
pickContact.addCategory(Intent.CATEGORY_HOME);  
mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);  
mSuspectButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        startActivityForResult(pickContact, REQUEST_CONTACT);  
    }  
});  
  
...
```

На этот раз кнопка выбора подозреваемого недоступна (рис. 15.8).



Рис. 15.8. Зabloкированная кнопка выбора подозреваемого

После завершения проверки удалите фиктивный код.

Листинг 15.16. Удаление фиктивного кода (CrimeFragment.java)

```

...

    final Intent pickContact = new Intent(Intent.ACTION_PICK,
        ContactsContract.Contacts.CONTENT_URI);
pickContact.addCategory(Intent.CATEGORY_HOME);
    mSuspectButton = (Button)v.findViewById(R.id.crime_suspect);
    mSuspectButton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            startActivityForResult(pickContact, REQUEST_CONTACT);
        }
    });

...

```

Упражнение. ShareCompat

Первое упражнение очень простое. В библиотеку поддержки Android входит класс ShareCompat с внутренним классом IntentBuilder. ShareCompat.IntentBuilder упрощает построение интен­тов точно такого вида, какой мы использовали для кнопки отчета.

Итак, первое упражнение: в слушателе OnClickListener кнопки mReportButton используйте для построения интен­та класс ShareCompat.IntentBuilder (вместо того, чтобы строить его вручную).

Упражнение. Другой неявный интен­т

Возможно, вместо отправки отчета разгне­ванный пользователь предпочтет разоб­раться с подозреваемым по телефону. Добавьте новую кнопку для звонка указан­ному подозреваемому.

Вам понадобится извлечь номер телефона из базы данных контактов. Для этого необходимо обратиться с запросом к другой таблице базы данных ContactsContract, которая называется CommonDataKinds.Phone. За дополнительными све­дениями о том, как получить эту информацию, обращайтесь к документации ContactsContract и ContactsContract.CommonDataKinds.Phone.

Пара подсказок: для запроса дополнительных данных можно воспользоваться разрешением android.permission.READ_CONTACTS. С этим разрешением вы сможете прочитать ContactsContract.Contacts._ID для получения идентификатора контакта из исходного запроса. Затем полученный идентификатор используется для получения данных из таблицы CommonDataKinds.Phone.

После получения телефонного номера можно создать неявный интент с URI телефона:

```
Uri number = Uri.parse("tel:5551234");
```

При этом может использоваться действие `Intent.ACTION_DIAL` или `Intent.ACTION_CALL`. `ACTION_CALL` запускает телефонное приложение и немедленно осуществляет звонок по номеру, отправленному в интенте; `ACTION_DIAL` только вводит номер и ждет, пока пользователь инициирует звонок.

Мы рекомендуем использовать `ACTION_DIAL`. Режим `ACTION_CALL` может быть ограничен, и для него определенно потребуются разрешения. Кроме того, у пользователя будет возможность немного остыть перед нажатием кнопки вызова.

16

Интенты при работе с камерой

Итак, вы научились работать с неявными интентами, и теперь преступления будут документироваться еще точнее. Располагая снимком места преступления, вы сможете поделиться жуткими подробностями со всеми желающими.

Для создания снимков вам понадобится пара новых инструментов, которые используются в сочетании с уже знакомыми вам неявными интентами. Неявный интент используется при запуске любимого приложения для работы с камерой и получения от него нового снимка.

Неявный интент может создать снимок, но где его разместить? И как вывести на экран созданную фотографию? В этой главе даны ответы на оба вопроса.

Место для хранения фотографий

Прежде всего следует обустроить место, в котором будет «жить» ваша фотография. Для этого понадобятся два новых объекта View: `ImageView` для отображения фотографии на экране и `Button` для создания снимка (рис. 16.1).

Если выделить отдельную строку под миниатюру и кнопку, приложение будет выглядеть убого и непрофессионально. Чтобы этого не произошло, необходимо аккуратно разместить все компоненты на экране.



Рис. 16.1. Новый интерфейс

Включение файлов макетов

Новый макет будет включать большой раздел, который будет выглядеть одинаково как в книжной, так и в альбомной версии `fragment_crime.xml`. Конечно, можно просто продублировать этот раздел `res/layout/fragment_crime.xml` и `res/layout-land/fragment_crime.xml`. Такое решение работает, но оно не единственное — также можно воспользоваться механизмом включения.

Включение (`include`) позволяет встроить один файл макета в другой. В нашем случае встраивается раздел, содержащий общие элементы. Работа начинается с создания файла макета для части представления, изображенной на рис. 16.2.

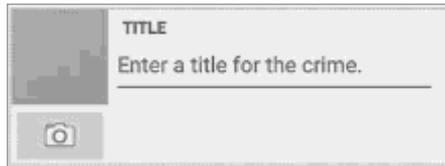


Рис. 16.2. Камера и текст

Присвойте файлу макета имя `view_camera_and_title.xml`. Начните с построения левой стороны (рис. 16.3).

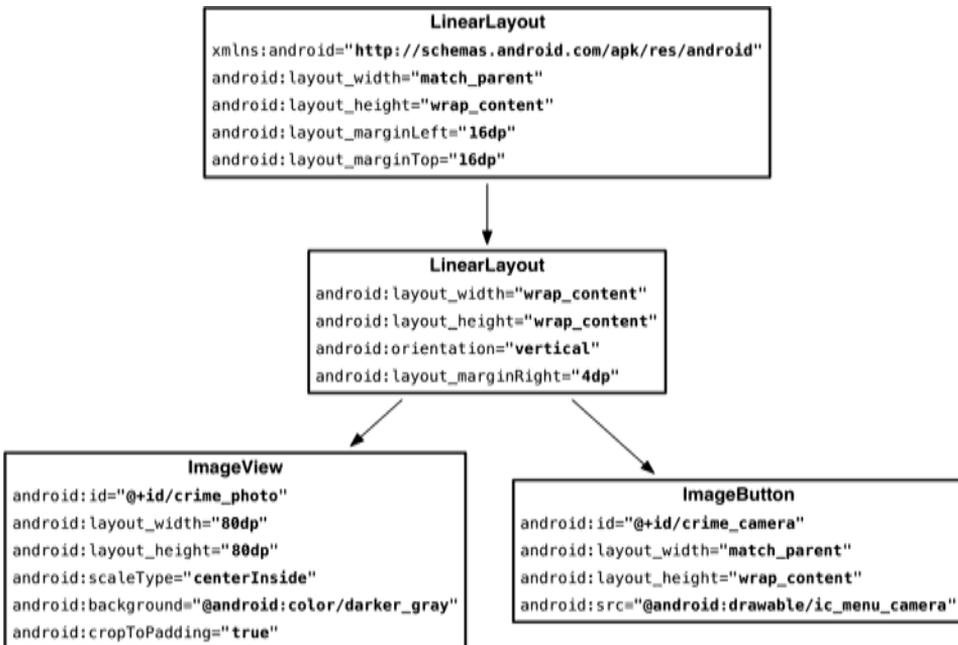


Рис. 16.3. Макет представления камеры (`res/layout/view_camera_and_title.xml`)

Затем создайте правую сторону (рис. 16.4).

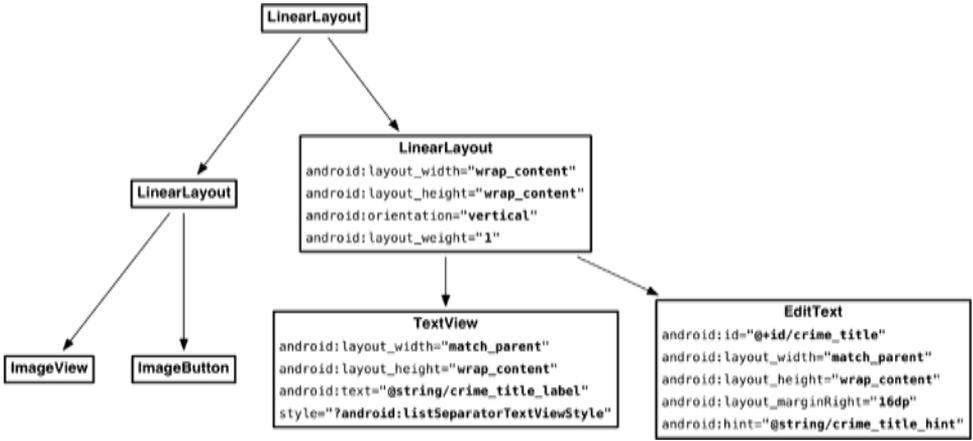


Рис. 16.4. Макет текстовой части (res/layout/view_camera_and_title.xml)

Перейдите в режим графического конструктора и убедитесь в том, что ваш файл макета выглядит так, как показано на рис. 16.2.

Для включения этого макета в другие файлы макетов используются теги `include`. Обратите внимание: в теге `include` атрибут `layout` не использует обычный префикс `android`.

Начните с изменения главного файла макета (рис. 16.5).

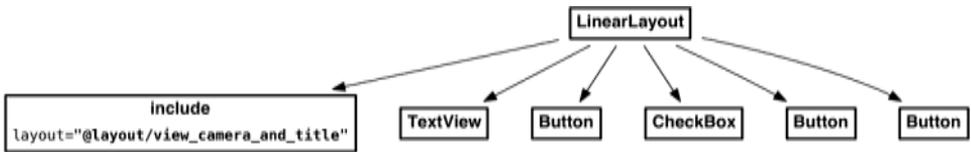


Рис. 16.5. Включение макета камеры для книжной ориентации (res/layout/fragment_crime.xml)

Затем проделайте то же для альбомного макета (рис. 16.6).

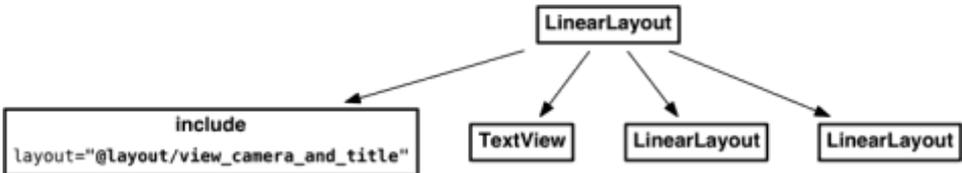


Рис. 16.6. Включение макета камеры для альбомной ориентации (res/layout-land/fragment_crime.xml)

Запустите приложение CriminalIntent; новый пользовательский интерфейс должен выглядеть так, как показано на рис. 16.1.

Выглядит замечательно, но для реагирования на нажатия ImageButton и управления содержимым ImageView нам понадобятся переменные экземпляров со ссылками на оба виджета. Поиск представлений во включаемых макетах не требует никаких особых действий. Как обычно, вызовите findViewById(int) для заполняемого макета fragment_crime.xml, и в поиск также будут включены представления из view_camera_and_title.xml.

Листинг 16.1. Добавление переменных экземпляров (CrimeFragment.java)

```
...
private CheckBox mSolvedCheckbox;
private Button mSuspectButton;
private ImageButton mPhotoButton;
private ImageView mPhotoView;
...

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    ...

    PackageManager packageManager = getActivity().getPackageManager();
    if (packageManager.resolveActivity(pickContact,
        PackageManager.MATCH_DEFAULT_ONLY) == null) {
        mSuspectButton.setEnabled(false);
    }

    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}

...
```

На этом ненадолго оставим пользовательский интерфейс (через несколько страниц эти кнопки будут подключены к логике приложения).

Внешнее хранилище

Одного лишь места на экране вашим фотографиям недостаточно. Полноразмерная фотография слишком велика для хранения в базе данных SQLite, не говоря уже об интернете. Ей необходимо место для хранения в файловой системе устройства.

Обычно такие данные размещаются в закрытом (приватном) хранилище. Помните, что именно в закрытой области хранится наша база данных SQLite. Такие методы, как Context.getFileStreamPath(String) и Context.getFilesDir(), позво-

ляют хранить в этой же области и обычные файлы (в папке по соседству с папкой `databases`, в которой размещается база данных SQLite).

Таблица 16.1. Методы для работы с внешними файлами и каталогами в Context

Метод	Назначение
File <code>getFilesDir()</code>	Возвращает дескриптор каталога для закрытых файлов приложения
FileInputStream <code>openFileInput(String name)</code>	Открывает существующий файл для ввода (относительно каталога файлов)
FileOutputStream <code>openFileOutput(String name, int mode)</code>	Открывает существующий файл для вывода, возможно, с созданием (относительно каталога файлов)
File <code>getDir(String name, int mode)</code>	Получает (и, возможно, создает) подкаталог в каталоге файлов
String[] <code>fileList()</code>	Получает список имен файлов в главном каталоге файлов (например, для использования с <code>openFileInput(String)</code>)
File <code>getCacheDir()</code>	Возвращает дескриптор каталога, используемого для хранения кэш-файлов. Будьте внимательны, поддерживайте порядок в этом каталоге и старайтесь использовать как можно меньше пространства

Если вы сохраняете файлы, которые должны использоваться *только текущим приложением*, — это именно то, что вам нужно.

С другой стороны, если запись в файлы должна осуществляться другим приложением, вам не повезло: хотя существует флаг `Context.MODE_WORLD_READABLE`, который можно передать при вызове `openFileOutput(String, int)`, он официально считается устаревшим, а на новых устройствах его надежность не гарантирована. Если вы сохраняете файлы, которые должны использоваться другими приложениями, или получаете файлы от других приложений (как, например, сохраненные фотографии), хранение должно осуществляться во внешнем хранилище.

Память внешнего хранилища делится на два вида: первичная область и все остальное. На всех устройствах Android присутствует по крайней мере одна область для внешнего хранения: *первичная* (`primary`) область, которая находится в папке, возвращаемой `Environment.getExternalStorageDirectory()`. Это может быть и SD-карта, но в наши дни она чаще интегрируется в само устройство. На некоторых устройствах имеется дополнительная внешняя память; она относится к категории «все остальное».

Класс `Context` тоже предоставляет методы для получения доступа к внешнему хранилищу. Эти методы предоставляют простые средства для обращения к первичной области, а также *относительно* простые средства для обращения ко всему остальному. Все эти методы сохраняют файлы в общедоступных местах, так что будьте осторожны.

Таблица 16.2. Основные методы для работы с файлами и каталогами в Context

Метод	Назначение
File getExternalCacheDir()	Возвращает дескриптор папки кэша в первичной внешней области. Используйте по аналогии с getCacheDir(), разве что с большей осторожностью. Android очищает содержимое этой папки с еще меньшей вероятностью, чем содержимое кэша в закрытой области
File[] getExternalCacheDirs()	Возвращает папки кэша для нескольких разделов внешнего хранилища
File getExternalFilesDir(String)	Возвращает дескриптор папки в первичном внешнем хранилище, предназначенной для хранения обычных файлов. При передаче типа String вы сможете обратиться к папке, предназначенной для конкретного типа содержимого. Константы типов определяются в Environment с префиксом DIRECTORY_ . Например, изображения хранятся в Environment.DIRECTORY_PICTURES
File[] getExternalFilesDirs(String)	То же, что getExternalFilesDir(String), но возвращает все возможные папки для заданного типа
File[] getExternalMediaDirs()	Возвращает дескрипторы для всех внешних папок, предоставляемых Android для хранения изображений, видео и музыки. От вызова getExternalFilesDir(Environment.DIRECTORY_PICTURES) этот метод отличается тем, что папка автоматически обрабатывается медиасканером, соответственно файлы становятся доступными для приложений, которые воспроизводят музыку, отображают графику или видеоролики. Соответственно все, что размещается в папке, возвращаемой getExternalMediaDirs(), автоматически появляется в этих приложениях

Теоретически внешние папки, предоставляемые этими методами, могут оказаться недоступными, потому что на некоторых устройствах для внешнего хранилища используется съемная SD-карта. На практике такая проблема встречается редко, потому что почти на всех современных устройствах в качестве «внешнего» хранилища используется несъемная внутренняя память. Таким образом, прилагать слишком значительные усилия для обработки такой возможности не стоит. Тем не менее мы рекомендуем включить простой код для защиты от подобных ситуаций; вскоре мы это сделаем.

Выбор места для хранения фотографии

Пора выделить фотографиям место, где они будут существовать. Сначала добавьте в Crime метод для получения имени файла.

Листинг 16.2. Добавление свойства для получения имени файла (Crime.java)

...

```
public void setSuspect(String suspect) {
    mSuspect = suspect;
}
```

```

    }

    public String getPhotoFilename() {
        return "IMG_" + getId().toString() + ".jpg";
    }
}

```

Метод `Crime.getPhotoFilename()` не знает, в какой папке будет храниться фотография. Однако имя файла будет уникальным, поскольку оно строится на основании идентификатора `Crime`.

Затем следует найти место, в котором будут располагаться фотографии. Класс `CrimeLab` отвечает за все, что относится к долгосрочному хранению данных в `CriminalIntent`, поэтому он становится наиболее естественным кандидатом. Добавьте в `CrimeLab` метод `getPhotoFile(Crime)`, который будет возвращать эту информацию.

Листинг 16.3. Определение местонахождения файла фотографии (`CrimeLab.java`)

```

public class CrimeLab {
    ...

    public Crime getCrime(UUID id) {
        ...
    }

    public File getPhotoFile(Crime crime) {
        File externalFilesDir = mContext
            .getExternalFilesDir(Environment.DIRECTORY_PICTURES);

        if (externalFilesDir == null) {
            return null;
        }

        return new File(externalFilesDir, crime.getPhotoFilename());
    }

    ...
}

```

Этот код не создает никакие файлы в файловой системе. Он только возвращает объекты `File`, представляющие нужные места. При этом он проверяет наличие внешнего хранилища для сохранения данных. Если внешнее хранилище недоступно, `getExternalFilesDir(String)` возвращает `null` — как и весь метод.

Использование интента камеры

Следующий шаг — непосредственное создание снимка. Здесь все просто: необходимо снова воспользоваться неявным интенментом.

Начнем с сохранения местонахождения файла фотографии. (Эта информация будет использоваться еще в нескольких местах, поэтому сохранение избавит от лишней работы.)

Листинг 16.4. Сохранение местонахождения файла фотографии (CrimeLab.java)

```
...
private Crime mCrime; private
File mPhotoFile; private
EditText mTitleField;
...

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID crimeId = (UUID) getArguments().getSerializable(ARG_CRIME_ID);
    mCrime = CrimeLab.get(getActivity()).getCrime(crimeId);
    mPhotoFile = CrimeLab.get(getActivity()).getPhotoFile(mCrime);
}
...
```

На следующем шаге мы подключим кнопку, которая будет непосредственно создавать снимок. Интент камеры определяется в MediaStore — верховном повелителе всего, что относится к аудиовизуальной информации в Android. Интент отправляется действием MediaStore.ACTION_IMAGE_CAPTURE, по которому Android запускает активность камеры и делает снимок за вас.

Но не стоит торопить события.

Разрешения для работы с внешним хранилищем

Как правило, для выполнения операций чтения или записи с внешним хранилищем требуется соответствующее разрешение. Разрешения представляют собой заранее известные строковые значения, которые включаются в манифест с использованием тега `<uses-permission>`. Они сообщают Android, что вы собираетесь выполнить некую операцию, для которой необходимо запросить разрешение у Android.

В данном случае Android ожидает, что вы запросите разрешение, потому что хочет обеспечить ответственность при выполнении операций. Вы сообщаете Android, что вам нужно обратиться к внешнему хранилищу, а Android указывает эту информацию пользователю при перечислении того, что делает ваше приложение, когда пользователь пытается установить его. При таком подходе никто не будет удивляться, когда ваше приложение начнет сохранять информацию на SD-карту.

В Android 4.4 (KitKat) это ограничение было ослаблено. Так как вызов `Context.getExternalFilesDir(String)` возвращает папку, относящуюся к вашему приложению, будет логично, что вы сможете читать и записывать находящиеся в этой

папке файлы. Соответственно в Android 4.4 (API 19) и выше для работы с этой папкой указанное разрешение не требуется. (Хотя оно по-прежнему необходимо для других видов внешнего хранилища.)

Добавьте в манифест строку, которая запрашивает разрешение для чтения из внешней памяти, но только до API 18.

Листинг 16.5. Запрос разрешения для работы с внешним хранилищем (AndroidManifest.xml)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.criminalintent" >

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18"
    />
    ...
```

С атрибутом `maxSdkVersion` приложение запрашивает это разрешение только в версиях Android, превышающих API 19 (Android KitKat).

Обратите внимание: мы запрашиваем только доступ к внешнему хранилищу для чтения. Также существует разрешение записи `WRITE_EXTERNAL_STORAGE`, но оно в данном случае не нужно. Мы ничего не записываем во внешнее хранилище: приложение камеры сделает это за нас.

Отправка интента

Теперь все готово к отправке интента камеры. Нужно действие `ACTION_CAPTURE_IMAGE` определяется в классе `MediaStore`. Этот класс определяет открытые интерфейсы, используемые в Android при работе с основными аудиовизуальными материалами — изображениями, видео и музыкой. К этой категории относится и интент, запускающий камеру.

По умолчанию `ACTION_CAPTURE_IMAGE` послушно запускает приложение камеры и делает снимок, но результат не является фотографией в полном разрешении. Вместо нее создается миниатюра с малым разрешением, которая упаковывается в объект `Intent`, возвращаемый в `onActivityResult(...)`.

Чтобы получить выходное изображение в высоком разрешении, необходимо сообщить, где должно храниться изображение в файловой системе. Эта задача решается передачей URI для места, в котором должен сохраняться файл, в `MediaStore.EXTRA_OUTPUT`.

Напишите неявный интент для сохранения фотографии в месте, определяемом `mPhotoFile`. Добавьте код, блокирующий кнопку при отсутствии приложения камеры или недоступности места, в котором должна сохраняться фотография. (Чтобы проверить доступность приложения камеры, запросите у `PackageManager` активности, реагирующие на неявный интент камеры. О том, как обращаться за информацией в `PackageManager`, более подробно рассказано в разделе «Проверка реагирующих активностей» главы 15.)

Листинг 16.6. Отправка интента камеры (CrimeFragment.java)

```
---
private static final int REQUEST_DATE = 0;
private static final int REQUEST_CONTACT = 1;
private static final int REQUEST_PHOTO= 2;
---

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                        Bundle savedInstanceState) {
    ---

    mPhotoButton = (ImageButton) v.findViewById(R.id.crime_camera);
    final Intent captureImage = new Intent
        (MediaStore.ACTION_IMAGE_CAPTURE);

    boolean canTakePhoto = mPhotoFile != null &&
        captureImage.resolveActivity(packageManager) != null;
    mPhotoButton.setEnabled(canTakePhoto);

    if (canTakePhoto) {
        Uri uri = Uri.fromFile(mPhotoFile);
        captureImage.putExtra(MediaStore.EXTRA_OUTPUT, uri);
    }

    mPhotoButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivityForResult(captureImage, REQUEST_PHOTO);
        }
    });

    mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);

    return v;
}
```

Запустите CriminalIntent и нажмите кнопку запуска приложения камеры (рис. 16.7).

Масштабирование и отображение растровых изображений

После всего, что было сделано, приложение успешно делает снимки, которые сохраняются в файловой системе для дальнейшего использования.

Следующий шаг — поиска файла с изображением, его загрузка и отображение для пользователя. Для этого необходимо загрузить данные изображения в объект Bitmap достаточного размера. Чтобы построить объект Bitmap на базе файла, достаточно воспользоваться классом BitmapFactory:

```
Bitmap bitmap = BitmapFactory.decodeFile(mPhotoFile.getPath());
```

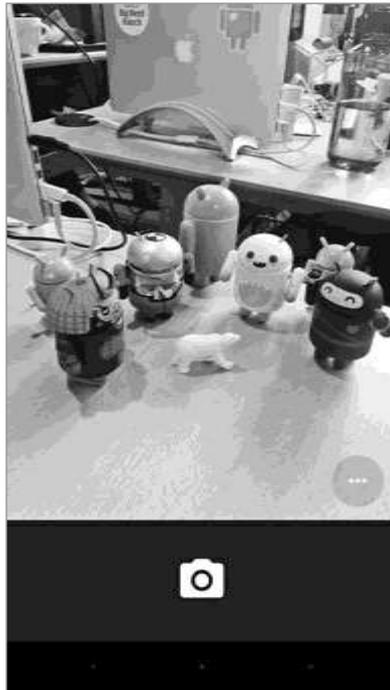


Рис. 16.7. [Подставьте ваше приложение для работы с камерой]

Но ведь должна же быть какая-то загвоздка, верно? Иначе мы бы просто напечатали эту строку жирным шрифтом, вы бы ввели ее — и на этом все закончилось.

Да, загвоздка существует: «достаточный размер» следует понимать буквально. Bitmap — простой объект для хранения необработанных данных пикселей. Таким образом, даже если исходный файл был сжат, в объекте Bitmap никакого сжатия не будет. И так, 24-битовое изображение с камеры на 16 мегапикселей, которое может занимать всего 5 Мбайт в формате JPG, в объекте Bitmap разрастается до 48(!) Мбайт.

Найти обходное решение возможно, но это означает, что изображение придется масштабировать вручную. Для этого можно сначала просканировать файл и определить его размер, затем вычислить, насколько его нужно масштабировать для того, чтобы он поместился в заданную область, и, наконец, заново прочитать файл для создания уменьшенного объекта Bitmap.

Создайте для этого метода новый класс с именем PictureUtils.java и добавьте в него статический метод с именем getScaledBitmap(String, int, int).

Листинг 16.7. Создание метода getScaledBitmap(...) (PictureUtils.java)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, int destWidth,
        int destHeight) {
```

```
// Чтение размеров изображения на диске
BitmapFactory.Options options = new BitmapFactory.Options();
options.inJustDecodeBounds = true;
BitmapFactory.decodeFile(path, options);

float srcWidth = options.outWidth;
float srcHeight = options.outHeight;

// Вычисление степени масштабирования
int inSampleSize = 1;
if (srcHeight > destHeight || srcWidth > destWidth) {
    if (srcWidth > srcHeight) {
        inSampleSize = Math.round(srcHeight / destHeight);
    } else {
        inSampleSize = Math.round(srcWidth / destWidth);
    }
}

options = new BitmapFactory.Options();
options.inSampleSize = inSampleSize;

// Чтение данных и создание итогового изображения
return BitmapFactory.decodeFile(path, options);
}
}
```

Ключевой параметр `inSampleSize` определяет величину «образца» для каждого пиксела исходного изображения: образец с размером 1 содержит один горизонтальный пиксел для каждого горизонтального пиксела исходного файла, а образец с размером 2 содержит один горизонтальный пиксел для каждых двух горизонтальных пикселов исходного файла. Таким образом, если значение `inSampleSize` равно 2, количество пикселов в изображении составляет четверть от количества пикселов оригинала.

И последняя неприятная новость: при запуске фрагмента вы еще не знаете величину `PhotoView`. До обработки макета никаких экранных размеров не существует. Первый проход этой обработки происходит после выполнения `onCreate(...)`, `onStart()` и `onResume()`, поэтому `PhotoView` и не знает своих размеров.

У проблемы есть два решения: либо подождать, пока будет сделан первый проход, либо воспользоваться консервативной оценкой. Второй способ менее эффективен, но более прямолинеен.

Напишите еще один статический метод с именем `getScaledBitmap(String, Activity)` для масштабирования `Bitmap` под размер конкретной активности.

Листинг 16.8. Метод масштабирования с консервативной оценкой (`PictureUtils.java`)

```
public class PictureUtils {
    public static Bitmap getScaledBitmap(String path, Activity activity) {
        Point size = new Point();
```

```

        activity.getWindowManager().getDefaultDisplay()
            .getSize(size);

        return getScaledBitmap(path, size.x, size.y);
    }

    ...

```

Метод проверяет размер экрана и уменьшает изображение до этого размера. Виджет `ImageView`, в который загружается изображение, всегда меньше размера экрана, так что эта оценка весьма консервативна.

Чтобы загрузить объект `Bitmap` в `ImageView`, добавьте в `CrimeFragment` метод для обновления `mPhotoView`.

Листинг 16.9. Обновление `mPhotoView` (`CrimeFragment.java`)

```

...
private String getCrimeReport() {
    ...
}

private void updatePhotoView() {
    if (mPhotoFile == null || !mPhotoFile.exists()) {
        mPhotoView.setImageDrawable(null);
    } else {
        Bitmap bitmap = PictureUtils.getScaledBitmap(
            mPhotoFile.getPath(), getActivity());
        mPhotoView.setImageBitmap(bitmap);
    }
}
}
}

```

Затем вызовите этот метод из `onCreateView(...)` и `onActivityResult(...)`.

Листинг 16.10. Вызов `updatePhotoView()` (`CrimeFragment.java`)

```

mPhotoButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivityForResult(captureImage, REQUEST_PHOTO);
    }
});

mPhotoView = (ImageView) v.findViewById(R.id.crime_photo);
updatePhotoView();

return v;
}

@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (resultCode != Activity.RESULT_OK) {

```

```
        return;
    }
    if (requestCode == REQUEST_DATE) {
        ...
    } else if (requestCode == REQUEST_CONTACT && data != null) {
        ...

    } else if (requestCode == REQUEST_PHOTO) {
        updatePhotoView();
    }
}
```

Запустите приложение снова. Изображение выводится в уменьшенном виде.

Объявление функциональности

Наша реализация камеры сейчас отлично работает. Остается решить еще одну задачу: сообщить о ней потенциальным пользователям. Когда приложение использует некоторое оборудование (например, камеру или NFC) или любой другой аспект, который может отличаться от устройства к устройству, настоятельно рекомендуется сообщить о нем Android. Это позволит другим приложениям (например, магазину Google Play) заблокировать установку приложения, если в нем используются возможности, не поддерживаемые вашим устройством.

Чтобы объявить, что в приложении используется камера, включите тег `<uses-feature>` в `AndroidManifest.xml`:

Листинг 16.11. Добавление тега `uses-feature` (`AndroidManifest.xml`)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.criminalintent" >
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
        android:maxSdkVersion="18"
        />
    <uses-feature android:name="android.hardware.camera"
        android:required="false"
        />
    ...
</manifest>
```

В этом примере в тег добавляется необязательный атрибут `android:required`. Почему? По умолчанию объявление об использовании некоторой возможности означает, что без нее приложение может работать некорректно. К `CriminalIntent` это не относится. Мы вызываем `resolveActivity(...)`, чтобы проверить наличие приложения камеры, после чего корректно блокируем кнопку, если приложение не найдено.

Атрибут `android:required="false"` корректно обрабатывает эту ситуацию. Мы сообщаем Android, что приложение может нормально работать без камеры, но некоторые части приложения окажутся недоступными.

Для любознательных: использование включений

В этой главе мы использовали механизм включения, чтобы нам не пришлось повторять большую часть разметки макета как в альбомной, так и в книжной ориентации. Этот пример наглядно показывает, чем удобно включение: оно сокращает объем вводимого текста и способствует соблюдению принципа DRY. Однако это не означает, что вы должны использовать включение во всех случаях, когда в макетах альбомной и книжной ориентации присутствуют общие элементы.

Для начала одно замечание относительно того, как работает включение: в этой главе мы использовали тег включения без атрибутов `android`. В таком случае включаемое представление получает все атрибуты, которые оно имело в исходном файле макета.

Однако ваши возможности этим не ограничиваются; вы также можете добавлять дополнительные атрибуты. Тогда эти атрибуты будут добавляться прямо в заполняемое корневое представление, при этом заменяются исходные значения таких атрибутов. Таким образом, если вы хотите изменить значение `layout_width`, — это возможно.

Следующее, о чем необходимо сказать в отношении включения, — это осторожность. В данном случае оно экономит время и уменьшает сложность, и это хорошо. Тем не менее включение не идеальный инструмент.

В представлениях `CriminalIntent` также дублируются некоторые кнопки. Почему бы не избавиться от дублирования посредством включения? Ответ: потому что мы так поступать не рекомендуем.

Одно из достоинств файлов макетов — их авторитетность; вы можете открыть файл макета и увидеть, какую структуру должно иметь представление. Включение файлов нарушает этот принцип. Чтобы понять, что происходит, вам придется внимательно просматривать файл макета и все включаемые в него файлы. Все это быстро начинает раздражать.

Визуальное оформление часто оказывается той частью приложения, которая изменяется чаще всего. В таких случаях соблюдение принципа DRY может означать, что вам придется потратить больше сил на его сохранение, чем на построение интерфейса. Итак, при применении включения на уровне представления постарайтесь действовать осмотрительно, обдуманно, сознательно и сдержанно.

Упражнение. Вывод увеличенного изображения

Конечно, вы видите уменьшенное изображение, но вряд ли вам удастся рассмотреть его во всех подробностях.

Создайте новый фрагмент `DialogFragment`, в котором отображается увеличенная версия фотографии места преступления. Когда пользователь нажимает в какой-

то точке миниатюры, на экране должен появиться DialogFragment с увеличенным изображением.

Упражнение. Эффективная загрузка миниатюры

В этой главе нам пришлось использовать довольно грубую оценку размера, до которого должно быть уменьшено изображение. Такое решение не идеально, но оно работает и быстро реализуется.

С существующими API можно использовать `ViewTreeObserver` — объект, который можно получить от любого представления в иерархии Activity:

```
ViewTreeObserver observer = mImageView.getViewTreeObserver();
```

Вы можете зарегистрировать для `ViewTreeObserver` разнообразных слушателей, включая `OnGlobalLayoutListener`. Этот слушатель инициирует событие каждый раз, когда происходит проход обработки макета.

Измените свой код так, чтобы он использовал размеры `mPhotoView`, когда они действительны, и ожидал прохода обработки макета перед первым вызовом `updatePhotoView()`.

17

Двухпанельные интерфейсы

В этой главе мы создадим для CriminalIntent планшетный интерфейс, в котором пользователь может одновременно видеть и взаимодействовать со списком преступлений и подробным описанием конкретного преступления. На рис. 17.1 изображен такой интерфейс, также часто называемый интерфейсом типа «список-детализация».

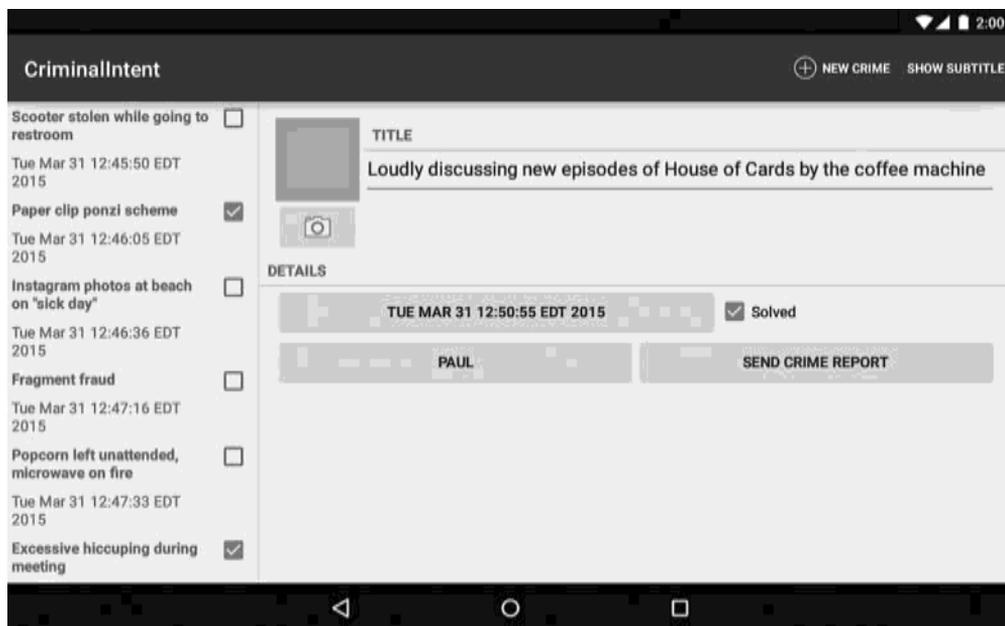


Рис. 17.1. Главное и детализированное представления одновременно находятся на экране

Для тестирования программ этой главы вам понадобится планшетное устройство или AVD. Чтобы создать виртуальный планшет AVD, выполните команду

Tools ▶ Android ▶ Android Virtual Device Manager. Щелкните на кнопке Create Virtual Device... и выберите слева категорию Tablet. Выберите аппаратный профиль, щелкните на кнопке Next и задайте для AVD целевой API не менее уровня 21 (рис. 17.2).

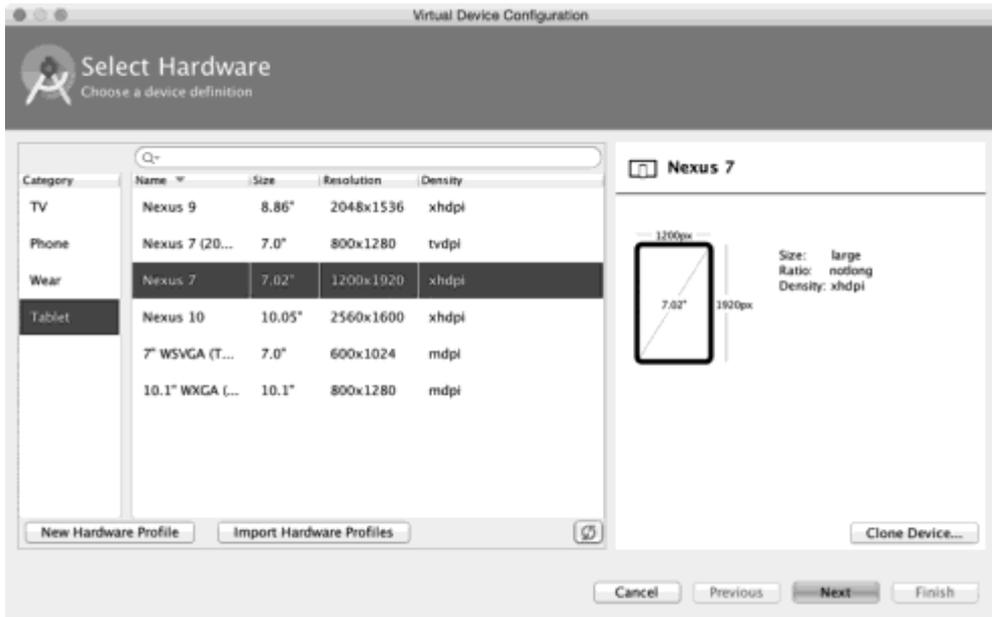


Рис. 17.2. Выбор устройства для планшетного AVD

Гибкость макета

На телефоне активность `CrimeListActivity` должна заполнять однопанельный макет, как она делает в настоящее время. На планшете она должна заполнять двухпанельный макет, способный одновременно отображать главное и детализированное представления.

В двухпанельном макете `CrimeListActivity` будет отображать как `CrimeListFragment`, так и `CrimeFragment`, как показано на рис. 17.3.

Для этого необходимо:

- ❶ изменить `SingleFragmentActivity`, чтобы выбор заполняемого макета не был жестко фиксирован в программе;
- ❶ создать новый макет, состоящий из двух контейнеров фрагментов;
- ❶ изменить `CrimeListActivity`, чтобы на телефонах заполнялся **один** макет, а на планшетах — двухпанельный.



Рис. 17.3. Разновидности макетов

Модификация SingleFragmentActivity

CrimeListActivity является субклассом SingleFragmentActivity. В настоящее время класс SingleFragmentActivity настроен таким образом, чтобы он всегда заполнял activity_fragment.xml. Чтобы класс SingleFragmentActivity стал более гибким, мы сделаем так, чтобы субкласс мог предоставлять свой идентификатор ресурса макета.

В файле SingleFragmentActivity.java добавьте защищенный метод, который возвращает идентификатор макета, заполняемого активностью.

Листинг 17.1. Обеспечение гибкости SingleFragmentActivity (SingleFragmentActivity.java)

```
public abstract class SingleFragmentActivity extends AppCompatActivity {
    protected abstract Fragment createFragment();

    @LayoutRes
    protected int getLayoutResId() {
        return R.layout.activity_fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_fragment);
        setContentView(getLayoutResId());
        FragmentManager fm = getSupportFragmentManager();
        Fragment fragment = fm.findFragmentById(R.id.fragment_container);
        if (fragment == null) {
            fragment = createFragment();
            fm.beginTransaction()

```

```

        .add(R.id.fragment_container, fragment)
        .commit();
    }
}
}

```

Реализация класса `SingleFragmentActivity` по умолчанию будет работать так же, как и прежде, но теперь его subclasses могут переопределить `getLayoutResId()` для возвращения макета, отличного от `activity_fragment.xml`. Метод `getLayoutResId()` помечается аннотацией `@LayoutRes`, чтобы сообщить Android Studio, что любая реализация этого метода должна возвращать действительный идентификатор ресурса макета.

Создание макета с двумя контейнерами фрагментов

В окне инструментов Project щелкните правой кнопкой мыши на каталоге `res/layout/` и создайте новый файл Android в формате XML. Убедитесь в том, что для файла выбран тип ресурса `Layout`, присвойте файлу имя `activity_twopane.xml` и назначьте его корневым элементом `LinearLayout`.

Используйте рис. 17.4 для построения разметки XML макета с двумя панелями.

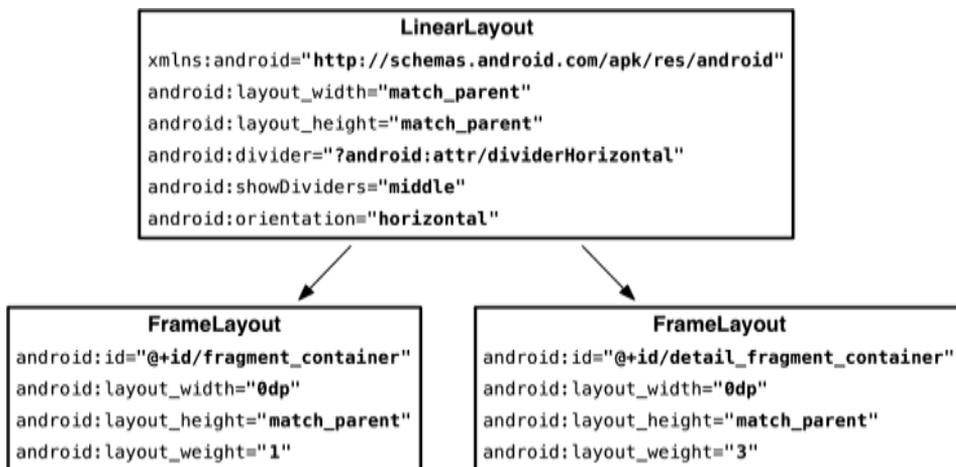


Рис. 17.4. Макет с двумя контейнерами фрагментов (`layout/activity_twopane.xml`)

Обратите внимание: у первого виджета `FrameLayout` задан идентификатор макета `fragmentContainer`, поэтому код `SingleFragmentActivity.onCreate(...)` может работать так же, как прежде. При создании активности фрагмент, возвращаемый `createFragment()`, появится на левой панели.

Протестируйте макет в CrimeListActivity; для этого переопределите метод `getLayoutResId()` так, чтобы он возвращал `R.layout.activity_twopane`.

Листинг 17.2. Переход к файлу двухпанельного макета (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity {  
  
    @Override  
    protected Fragment createFragment() {  
        return new CrimeListFragment();  
    }  
  
    @Override  
    protected int getLayoutResId() {  
        return R.layout.activity_twopane;  
    }  
}
```

Запустите приложение CriminalIntent на планшетном устройстве и убедитесь в том, что на экране отображаются две панели (рис. 17.5). Большая панель детализации пуста, а нажатие на элементе списка не отображает подробную информацию о преступлении. Контейнер детализированного представления будет подключен позднее в этой главе.

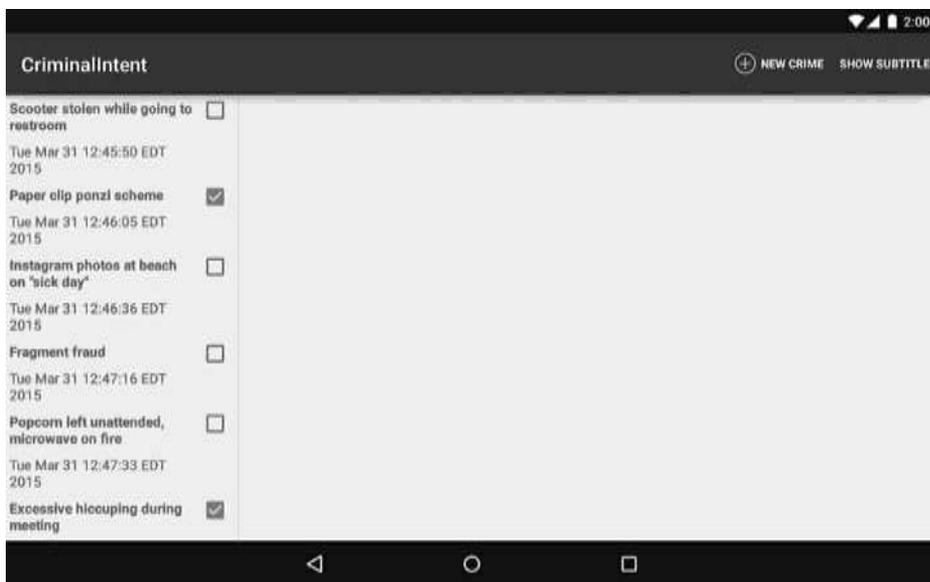


Рис. 17.5. Двухпанельный макет на планшете

В своей текущей версии CrimeListActivity также заполняет двухпанельный интерфейс при запуске на телефоне. В следующем разделе мы исправим этот недостаток при помощи ресурса-псевдонима.

Использование ресурса-псевдонима

Ресурс-псевдоним (alias resource) представляет собой ресурс, указывающий на другой ресурс. Ресурсы-псевдонимы находятся в каталоге `res/values/` и по умолчанию определяются в файле `refs.xml`.

В этом разделе мы создадим ресурс-псевдоним, который на телефонах ссылается на макет `activity_fragment.xml` и на планшетах — на макет `activity_twopane.xml`.

На следующем шаге мы должны добиться того, чтобы в `CrimeListActivity` отображались разные файлы макетов в зависимости от того, на каком устройстве работает приложение. Это делается точно также, как мы отображаем разные макеты для книжной и альбомной ориентации: при помощи квалификатора ресурса.

Решение на уровне файлов в `res/layout` работает, но у него есть свои недостатки. Каждый файл макета должен содержать полную копию отображаемого макета, а это может привести к заметной избыточности. Чтобы создать файл макета `activity_masterdetail.xml`, пришлось бы скопировать все содержимое `activity_fragment.xml` в `res/layout/activity_masterdetail.xml`, а все содержимое `activity_twopane.xml` — в `res/layout-sw600dp/activity_masterdetail.xml`. (Вскоре вы увидите, что означает `sw600dp`.)

Вместо этого мы воспользуемся ресурсом-псевдонимом. В этом разделе мы создадим ресурс-псевдоним, который ссылается на `activity_fragment.xml` на телефонах и макет `activity_twopane.xml` на планшетах.

На панели `Package Explorer` щелкните правой кнопкой мыши на каталоге `res/layout/` и создайте новый ресурсный файл значений. Присвойте файлу имя `refs.xml`, а каталогу — имя `values`. Квалификаторов в именах быть не должно. Щелкните на кнопке `Finish`. Затем добавьте элемент, приведенный в листинге 17.3.

Листинг 17.3. Создание значения по умолчанию для ресурса-псевдонима (`res/values/refs.xml`)

```
<resources>
  <item name="activity_masterdetail" type="layout">@layout/
    activity_fragment</item>
</resources>
```

Значение ресурса представляет собой ссылку на однопанельный макет. Ресурс также обладает идентификатором: `R.layout.activity_masterdetail`. Обратите внимание: внутренний класс идентификатора определяется атрибутом `type` псевдонима. И хотя сам псевдоним находится в `res/values/`, его идентификатор хранится в `R.layout`.

Теперь этот идентификатор ресурса может использоваться вместо `R.layout.activity_fragment`. Внесите следующее изменение в `CrimeListActivity`.

Листинг 17.4. Повторная замена макета (`CrimeListActivity.java`)

```
@Override
protected int getLayoutResId() {
  return R.layout.activity_twopane;
  return R.layout.activity_masterdetail;
}
```

Запустите приложение CriminalIntent и убедитесь в том, что псевдоним работает правильно. Активность CrimeListActivity снова должна заполнять однопанельный макет.

Создание альтернативы для планшета

Так как псевдоним находится в res/values/, он используется по умолчанию. Следовательно, по умолчанию CrimeListActivity заполняет однопанельный макет.

Теперь мы создадим альтернативный ресурс, чтобы псевдоним activity_masterdetail на планшетных устройствах ссылался на activity_twopane.xml.

В окне инструментов Project щелкните правой кнопкой мыши на каталоге res/values и создайте новый файл ресурсов значений. Присвойте ему имя refs.xml и снова присвойте каталогу имя values. Но на этот раз выберите в категории Available qualifiers вариант Smallest Screen Width и щелкните на кнопке >>, чтобы переместить квалификатор вправо (рис. 17.6).

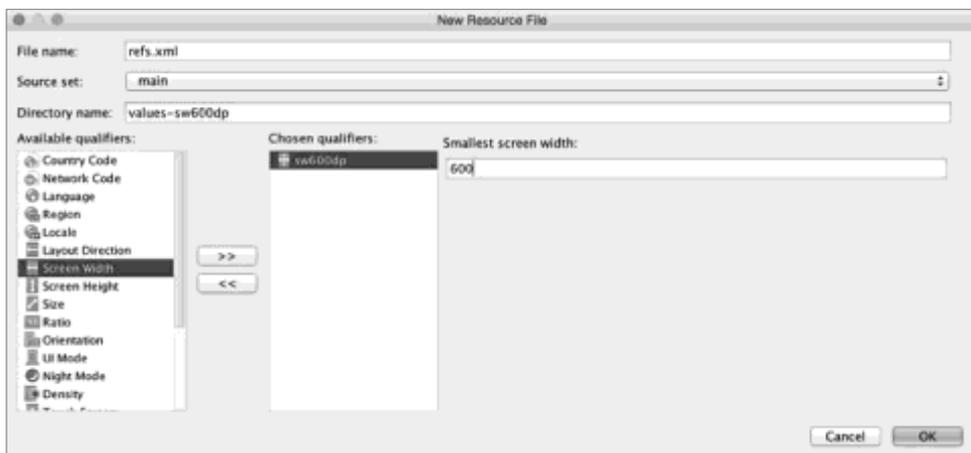


Рис. 17.6. Добавление квалификатора

Этот квалификатор работает несколько иначе: вам предлагается задать значение минимальной длины экрана. Введите 600 и щелкните на кнопке ОК. После открытия нового файла ресурсов добавьте псевдоним activity_masterdetail и в этот файл, но свяжите его с другим файлом макета.

Листинг 17.5. Альтернативный псевдоним для устройств с экраном большего размера (res/values-sw600dp/refs.xml)

```
<resources>
  <item name="activity_masterdetail" type="layout">@layout/
    activity_twopane</item>
</resources>
```

Разберемся, что же здесь происходит. Наша цель — сформировать логику, которая работает следующим образом:

- ❶ для устройств с экраном меньше заданного размера использовать `activity_fragment.xml`;
- ❷ для устройств с экраном более заданного размера использовать `activity_twopane.xml`.

Android не предоставляет возможности использовать ресурс только в том случае, если экран устройства меньше определенного размера, но предлагает неплохую альтернативу. Конфигурационный квалификатор `-sw600dp` позволяет предоставить ресурсы только в том случае, если экран устройства больше определенного размера. Сокращение «sw» происходит от «Smallest Width» (наименьшая ширина), но относится к меньшему *размеру* экрана, а следовательно, не зависит от текущей ориентации устройства.

Используя квалификатор `-sw600dp`, вы указываете: «Этот ресурс должен использоваться на любых устройствах, у которых меньший размер составляет `600dp` и выше». Это хороший критерий для определения экранов планшетных устройств.

А что делать с другой частью, где нужно использовать `activity_fragment.xml` на меньших устройствах? Меньшие устройства не соответствуют квалификатору `-sw600dp`, поэтому для них будет использован вариант по умолчанию: `activity_fragment.xml`.

Запустите приложение `CriminalIntent` на телефоне и на планшете. Убедитесь в том, что одно- и двухпанельные макеты отображаются там, где предполагалось.

Активность: управление фрагментами

Итак, макеты ведут себя так, как положено, и мы можем перейти к добавлению `CrimeFragment` в контейнер фрагмента детализации при использовании двухпанельного макета `CrimeListActivity`.

На первый взгляд кажется, что для этого достаточно написать альтернативную реализацию `CrimeHolder.onClick(View)` для планшетов. Вместо запуска нового экземпляра `CrimePagerActivity` метод `onClick(View)` получает экземпляр `FragmentManager`, принадлежащий `CrimeListActivity`, и закрепляет транзакцию, которая добавляет `CrimeFragment` в контейнер фрагмента детализации.

Код из `CrimeListFragment.CrimeHolder` выглядит примерно так:

```
public void onClick(View v) {
    // Включение нового экземпляра CrimeFragment в макете активности
    Fragment fragment = CrimeFragment.newInstance(mCrime.getId());
    FragmentManager fm = getActivity().getSupportFragmentManager();
    fm.beginTransaction()
        .add(R.id.detail_fragment_container, fragment)
        .commit();
}
```

Такое решение работает, но оно противоречит хорошему стилю программирования Android. Предполагается, что фрагменты представляют собой автономные компоуемые блоки. Если написанный вами фрагмент добавляет фрагменты в `FragmentManager` активности, значит, он делает допущения относительно того, как работает активность-хост, и перестает быть автономным компоуемым блоком.

Например, в приведенном выше коде `CrimeListFragment` добавляет `CrimeFragment` в `CrimeListActivity` и предполагает, что в макете `CrimeListActivity` присутствует контейнер `detail_fragment_container`. Такими вопросами должна заниматься активность-хост `CrimeListFragment`, а не `CrimeListActivity`.

Для сохранения независимости фрагментов мы делегируем выполнение работы активности-хосту, определяя интерфейсы обратного вызова в ваших фрагментах. Активности-хосты реализуют эти интерфейсы для выполнения операций по управлению фрагментами и обеспечения макетно-зависимого поведения.

Интерфейсы обратного вызова фрагментов

Чтобы делегировать функциональность активности-хосту, фрагмент обычно определяет интерфейс обратного вызова с именем `Callbacks`. Этот интерфейс определяет работу, которая должна быть выполнена для фрагмента его «начальником» — активностью-хостом. Любая активность, выполняющая функции хоста фрагментов, должна реализовать этот интерфейс.

С интерфейсом обратного вызова фрагмент может вызывать методы активности-хоста, не располагая никакой информацией о ней.

Реализация `CrimeListFragment.Callbacks`

Чтобы реализовать интерфейс `Callbacks`, следует сначала определить переменную для хранения объекта, реализующего `Callbacks`. Затем активность-хост преобразуется к `Callbacks`, а результат присваивается этой переменной.

Активность назначается в методе жизненного цикла `Fragment`:

```
public void onAttach(Activity activity)
```

Этот метод вызывается при присоединении фрагмента к активности (независимо от того, был он сохранен или нет).

Аналогичным образом переменной присваивается `null` в соответствующем завершающем методе жизненного цикла:

```
public void onDetach()
```

Переменной присваивается `null`, потому что в дальнейшем вы не сможете обратиться к активности или рассчитывать на то, что активность продолжит существовать.

В файле `CrimeListFragment.java` включите в класс `CrimeListFragment` интерфейс `Callbacks`. Также добавьте переменную `mCallbacks` и переопределите методы `onAttach(Activity)` и `onDetach()`, в которых задается и сбрасывается ее значение.

Листинг 17.6. Добавление интерфейса обратного вызова (CrimeListFragment.java)

```
public class CrimeListFragment extends Fragment {
    ---
    private boolean mSubtitleVisible;
    private Callbacks mCallbacks;

    /**
     * Обязательный интерфейс для активности-хоста.
     */
    public interface Callbacks {
        void onCrimeSelected(Crime crime);
    }

    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        mCallbacks = (Callbacks) activity;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }
    ---

    @Override
    public void onSaveInstanceState(Bundle outState) {
        super.onSaveInstanceState(outState);
        outState.putBoolean(SAVED_SUBTITLE_VISIBLE, mSubtitleVisible);
    }

    @Override
    public void onDetach() {
        super.onDetach();
        mCallbacks = null;
    }
}
```

Теперь у CrimeListFragment имеется механизм вызова методов активности-хоста. Неважно, какая активность является хостом, — если она реализует CrimeListFragment.Callbacks, внутренняя реализация CrimeListFragment будет работать одинаково.

Обратите внимание на то, как CrimeListFragment выполняет непроверяемое преобразование своей активности к CrimeListFragment.Callbacks. Это означает, что активность-хост *должна* реализовать CrimeListFragment.Callbacks. В самой зависимости нет ничего плохого, но ее важно документировать.

Затем в классе CrimeListActivity реализуйте CrimeListFragment.Callbacks. Метод onCrimeSelected(Crime) пока оставьте пустым.

Листинг 17.7. Реализация обратных вызовов (CrimeListActivity.java)

```
public class CrimeListActivity extends SingleFragmentActivity
implements CrimeListFragment.Callbacks {
```

```

@Override
protected Fragment createFragment() {
    return new CrimeListFragment();
}

@Override
protected int getLayoutResId() {
    return R.layout.activity_masterdetail;
}

@Override
public void onCrimeSelected(Crime crime) {
}
}

```

CrimeListFragment будет вызывать этот метод в CrimeHolder.onClick(...), а также тогда, когда пользователь выбирает команду создания новой записи преступления. Для начала нужно понять, как должна быть устроена реализация CrimeListActivity.onCrimeSelected(Crime).

При вызове onCrimeSelected(Crime) класс CrimeListActivity должен выполнить одну из двух операций:

- ❶ если используется телефонный интерфейс — запустить новый `CrimePagerActivity`;
- ❷ если используется планшетный интерфейс — поместить `CrimeFragment` в `detail_fragment_container`.

Чтобы определить, интерфейс какого типа был заполнен, можно проверить конкретный идентификатор интерфейса. Впрочем, лучше проверить наличие в макете `detail_fragment_container`. Такая проверка будет более точной и надежной. Имена файлов могут изменяться, и вас на самом деле не интересует, по какому файлу заполнялся макет; необходимо знать лишь то, имеется ли у него контейнер `detail_fragment_container` для размещения `CrimeFragment`.

Если макет содержит `detail_fragment_container`, мы создадим транзакцию фрагмента, которая удаляет существующий экземпляр `CrimeFragment` из `detail_fragment_container` (если он имеется) и добавляет экземпляр `CrimeFragment`, который мы хотим там видеть.

В файле `CrimeListActivity.java` реализуйте метод `onCrimeSelected(Crime)`, который будет обрабатывать выбор преступления в любом варианте интерфейса.

Листинг 17.8. Условный запуск CrimeFragment (CrimeListActivity.java)

```

@Override
public void onCrimeSelected(Crime crime) {
    if (findViewById(R.id.detail_fragment_container) == null) {
        Intent intent = CrimePagerActivity.newIntent(this, crime.getId());
        startActivity(intent);
    } else {
        Fragment newDetail = CrimeFragment.newInstance(crime.getId());
        getSupportFragmentManager().beginTransaction()

```

```

        .replace(R.id.detail_fragment_container, newDetail)
        .commit();
    }
}

```

Наконец, в классе `CrimeListFragment` мы будем вызывать `onCrimeSelected(Crime)` в тех местах, где сейчас запускается новый экземпляр `CrimePagerActivity`.

В файле `CrimeListFragment.java` измените методы `CrimeHolder.onClick(View)` и `onOptionsItemSelected(Menuitem)` так, чтобы в них вызывался метод `Callbacks.onCrimeSelected(Crime)`.

ЛИСТИНГ 17.9. Активизация обратных вызовов (`CrimeListFragment.java`)

```

@Override
public boolean onOptionsItemSelected(Menuitem item) {
    switch (item.getItemId()) {
        case R.id.menu_item_new_crime:
            Crime crime = new Crime();
            CrimeLab.get(getActivity()).addCrime(crime);
Intent intent = CrimePagerActivity
newIntent(getActivity(), crime.getId());
startActivity(intent);
            updateUI();
            mCallbacks.onCrimeSelected(crime);
            return true;
        case R.id.menu_item_show_subtitle:
            mSubtitleVisible = !mSubtitleVisible;
            getActivity().invalidateOptionsMenu();
            updateSubtitle();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
...
...
private class CrimeHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    ...
    @Override
    public void onClick(View v) {
Intent intent = CrimePagerActivity.newIntent(getActivity(), mCrime
startActivity(intent);
        mCallbacks.onCrimeSelected(mCrime);
    }
}

```

При обратном вызове в `onOptionsItemSelected(...)` содержимое списка также немедленно перезагружается после добавления нового преступления. Это необходимо, потому что на планшетах при добавлении нового преступления список остается видимым на экране (прежде его закрывал экран детализации).

Запустите приложение CriminalIntent на планшете. Создайте новое преступление; экземпляр CrimeFragment добавляется и отображается в detail_fragment_container. Затем просмотрите какое-либо старое преступление и убедитесь в том, что CrimeFragment заменяется новым экземпляром (рис. 17.7).

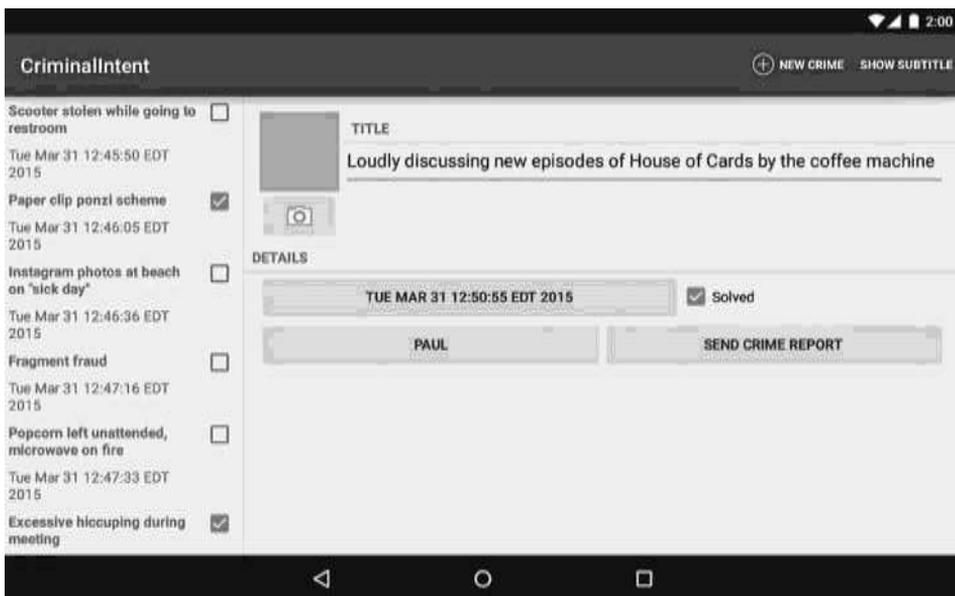


Рис. 17.7. Главное и детализированное представления связаны между собой

Выглядит замечательно! Но существует одна маленькая проблема: внесение изменений в преступление не приводит к обновлению списка. В настоящее время список перезагружается только после добавления преступления и в CrimeListFragment.onResume(). При этом на планшетах экземпляр CrimeListFragment остается видимым рядом с CrimeFragment. CrimeListFragment не приостанавливается при появлении CrimeFragment, поэтому и возобновления не происходит, а список не перезагружается.

Для решения этой проблемы мы используем другой интерфейс обратного вызова из CrimeFragment.

Реализация CrimeFragment.Callbacks

CrimeFragment определяет следующий интерфейс:

```
public interface Callbacks {  
    void onCrimeUpdated(Crime crime);  
}
```

Чтобы класс CrimeFragment «проталкивал» обновления в другой Fragment, должны выполняться два условия. Во-первых, так как единственным источником до-

стоверной информации для `CriminalIntent` является база данных `SQLite`, он должен сохранить объект `Crime` в `CrimeLab`. Затем `CrimeFragment` будет вызывать метод `onCrimeUpdated(Crime)` активности-хоста при сохранении любых изменений в `Crime`. `CrimeListActivity` реализует `onCrimeUpdated(Crime)` для перезагрузки списка `CrimeListFragment`, что приводит к извлечению обновленной информации из базы данных и ее отображению.

Прежде чем браться за интерфейс в `CrimeFragment`, измените видимость метода `CrimeListFragment.updateUI()`, чтобы он мог вызываться из `CrimeListFragment`.

Листинг 17.10. Изменение видимости `updateUI()` (`CrimeListFragment.java`)

```
private public void updateUI() {  
    ...  
}
```

В файле `CrimeListFragment.java` добавьте интерфейс обратного вызова вместе с переменной `mCallbacks` и реализациями `onAttach(...)` и `onDetach()`.

Листинг 17.11. Добавление обратных вызовов `CrimeFragment` (`CrimeFragment.java`)

```
...  
private ImageButton mPhotoButton;  
private ImageView mPhotoView;  
private Callbacks mCallbacks;  
  
/**  
 * Необходимый интерфейс для активности-хоста.  
 */  
public interface Callbacks {  
    void onCrimeUpdated(Crime crime);  
}  
  
public static CrimeFragment newInstance(UUID crimeId) {  
    ...  
}  
  
@Override  
public void onAttach(Activity activity) {  
    super.onAttach(activity);  
    mCallbacks = (Callbacks)activity;  
}  
  
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
}  
  
@Override  
public void onPause() {  
    ...  
}  
  
@Override  
public void onDetach() {
```

```

    super.onDetach();
    mCallbacks = null;
}

```

Затем в классе `CrimeListActivity` реализуйте `CrimeListFragment.Callbacks` для перезагрузки списка в `onCrimeUpdated(Crime)`.

Листинг 17.12. Обновление списка преступлений (`CrimeListActivity.java`)

```

public class CrimeListActivity extends SingleFragmentActivity
    implements CrimeListFragment.Callbacks, CrimeFragment.Callbacks {
    ...
    public void onCrimeUpdated(Crime crime) {
        CrimeListFragment listFragment = (CrimeListFragment)
            getSupportFragmentManager()
                .findFragmentById(R.id.fragment_container);
        listFragment.updateUI();
    }
}

```

Интерфейс `CrimeFragment.Callbacks` должен быть реализован во всех активностях, выполняющих функции хоста для `CrimeFragment`. Следовательно, пустую реализацию также следует включить и в `CrimePagerActivity`.

Листинг 17.13. Реализация пустых обратных вызовов (`CrimePagerActivity.java`)

```

public class CrimePagerActivity extends AppCompatActivity
    implements CrimeFragment.Callbacks {
    ...
    @Override
    public void onCrimeUpdated(Crime crime) {
    }
}

```

`CrimeFragment` в своей внутренней работе часто будет выполнять этот хитрый маневр из двух шагов: шаг влево, сохранить `mCrime` в `CrimeLab`. Шаг вправо, вызвать `mCallbacks.onCrimeUpdated(Crime)`. Добавим метод, чтобы упростить его выполнение.

Листинг 17.14. Добавление метода `updateCrime()` (`CrimeFragment.java`)

```

...
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
}

private void updateCrime() {
    CrimeLab.get(getActivity()).updateCrime(mCrime);
    mCallbacks.onCrimeUpdated(mCrime);
}

```

```
private void updateDate() {
    mDateButton.setText(mCrime.getDate().toString());
}
...

```

Добавьте в CrimeFragment.java вызовы updateCrime() при изменении краткого описания или состояния раскрытия преступления.

Листинг 17.15. Вызов onCrimeUpdated(Crime) (CrimeFragment.java)

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    ...
    mTitleField.addTextChangedListener(new TextWatcher() {
        ...
        @Override
        public void onTextChanged(CharSequence s, int start, int before,
            int count) {
            mCrime.setTitle(s.toString());
            updateCrime();
        }
    });
    ...
    mSolvedCheckbox.setOnCheckedChangeListener(new OnCheckedChangeListener() {
        @Override
        public void onCheckedChanged(CompoundButton buttonView,
            boolean isChecked) {
            mCrime.setSolved(isChecked);
            updateCrime();
        }
    });
    ...
}

```

Также необходимо вызвать onCrimeUpdated(Crime) в onActivityResult(...) при возможном изменении даты, фотографии и подозреваемого. В настоящее время фотография и подозреваемый не отображаются в представлении элемента списка, но CrimeFragment может сообщить об этих обновлениях.

Листинг 17.16. Повторный вызов onCrimeUpdated(Crime) (CrimeFragment.java)

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    ...
    if (requestCode == REQUEST_DATE) {
        Date date = (Date) data
            .getSerializableExtra(DatePickerFragment.EXTRA_DATE);
        mCrime.setDate(date);
    }
}

```

```

    updateCrime();
    updateDate();
} else if (requestCode == REQUEST_CONTACT && data != null) {
    ...
    try {
        ...

        String suspect = c.getString(0);
        mCrime.setSuspect(suspect);
        updateCrime();
        mSuspectButton.setText(suspect);
    } finally {
        c.close();
    }
} else if (requestCode == REQUEST_PHOTO) {
    updateCrime();
    updatePhotoView();
}
}
}

```

Запустите приложение CriminalIntent на планшете и убедитесь в том, что RecyclerView обновляется при внесении изменений в CrimeFragment (рис. 17.8). Затем запустите его на телефоне и убедитесь, что приложение работает как прежде.

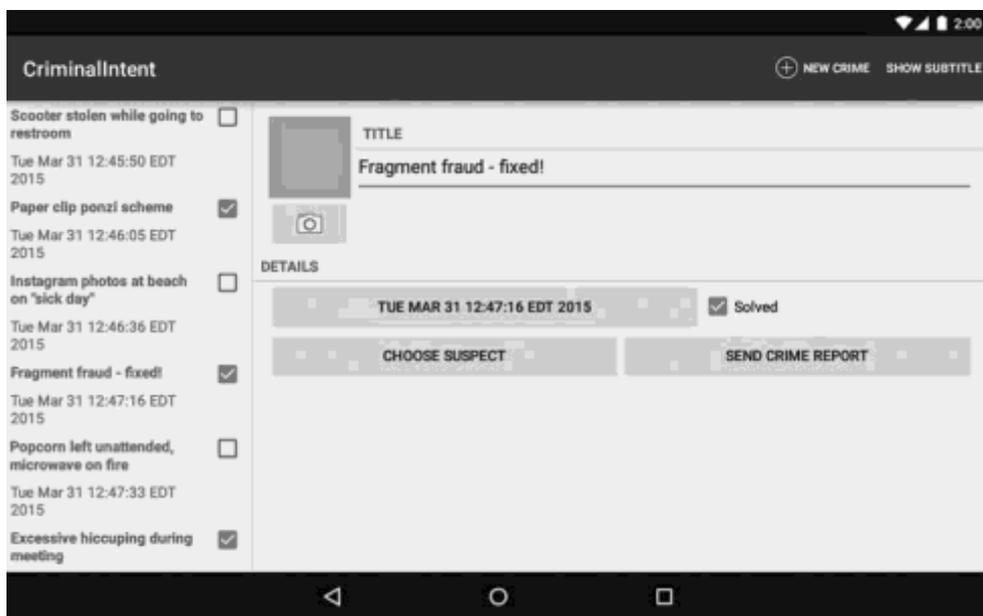


Рис. 17.8. Изменения, внесенные в детализированном представлении, отражаются в списке

На этом наше знакомство с CriminalIntent подходит к концу. На протяжении 11 глав мы создали сложное приложение, которое использует фрагменты, взаимо-

действует с другими приложениями, делает снимки и сохраняет данные. Почему бы не отпраздновать окончание работы куском торта?

Только не забудьте убрать за собой крошки, чтобы ваш проступок не попал в `CriminalIntent`.

Для любознательных: подробнее об определении размера экрана

До выхода Android 3.2 для предоставления альтернативных ресурсов в зависимости от размера устройства использовался квалификатор размера экрана. Этот квалификатор группирует разные устройства на четыре категории — `small`, `normal`, `large` и `xlarge`.

В табл. 17.1 приведены минимальные размеры для каждого квалификатора.

Таблица 17.1. Квалификаторы размера экрана

Имя	Минимальный размер экрана
<code>small</code>	320 × 426dp
<code>normal</code>	320 × 470dp
<code>large</code>	480 × 640dp
<code>xlarge</code>	720 × 960dp

Квалификаторы размера экрана были объявлены устаревшими в Android 3.2. Они были заменены дискретными квалификаторами, позволяющими определять размеры устройства. В табл. 17.2 перечислены эти новые квалификаторы.

Таблица 17.2. Дискретные квалификаторы размера экрана

Формат квалификатора	Описание
<code>wXXXdp</code>	Доступная ширина: ширина больше либо равна XXX dp
<code>hXXXdp</code>	Доступная высота: высота больше либо равна XXX dp
<code>swXXXdp</code>	Минимальная ширина: ширина или высота (меньшая из двух) больше либо равна XXX dp

Предположим, вы хотите задать макет, который должен использоваться только в том случае, если ширина экрана не менее 300dp. В этом случае можно поместить файл макета в каталог `res/layout-w300dp` («w» — сокращение от «width», то есть «ширина»). То же самое можно сделать для высоты при помощи префикса «h» («height», то есть «высота»).

Впрочем, ширина и высота могут меняться местами в зависимости от ориентации устройства. Для обнаружения конкретного размера экрана используется префикс «sw» («smallest width», то есть «минимальная ширина»). Он задает наименьший размер экрана, которым в зависимости от ориентации устройства может быть как ширина, так и высота. Если размеры экрана равны 1024 × 800, то метрика `sw` равна 800. Если размеры экрана равны 800 × 1024, то метрика `sw` все равно равна 800.

18

АКТИВЫ

До настоящего момента мы имели дело с основным механизмом, используемым в Android для упаковки графики, XML и других материалов, не связанных с Java: системой ресурсов. В этой главе будет рассмотрен другой способ упаковки контента для распространения с приложением: *активы* (assets).

В этой главе также начнется работа над новым приложением BeatBox (рис. 18.1), предназначенным для воспроизведения всевозможных угрожающих звуков.



Рис. 18.1. Приложение BeatBox к концу этой главы

Почему активы, а не ресурсы

Ресурсы могут использоваться для хранения звуков. Сохраните файл вида 79_long_scream.wav в res/raw, и вы сможете обратиться к нему по идентификатору R.raw.79_long_scream. При хранении звуков в ресурсах вы сможете делать с ними все, что обычно делается с ресурсами: например, использовать разные звуки для разных ориентаций, языков, версий Android и т.д.

Однако в приложении BeatBox используется большой набор звуков: свыше 20 разных файлов. Работать с ними по одному в системе ресурсов будет весьма неудобно. Было бы гораздо удобнее отправить их все в одной большой папке, однако ресурсы такой возможности не дают, как и возможности использовать что-либо помимо плоской структуры.

Именно в таких ситуациях прекрасно работает система активов. Активы представляют собой некое подобие миниатюрной файловой системы, которая поставляется вместе с приложением. С активами вы можете использовать такую структуру папок, которая вам нужна. Благодаря своим организационным возможностям активы часто применяются для загрузки графики и звуков в приложениях, в которых они интенсивно используются, например в играх.

Создание приложения BeatBox

Пора браться за дело. Начнем с создания приложения BeatBox. Выполните в Android Studio команду File ▶ New Project... для создания нового проекта. Присвойте ему имя BeatBox и введите домен компании android.bignerdranch.com. Выберите минимальный уровень SDK API 16 и начните с пустой активности с именем BeatBoxActivity. Оставьте остальным параметрам значения по умолчанию.

В приложении снова будет использоваться виджет RecyclerView, поэтому откройте настройки проекта и добавьте зависимость com.android.support:recyclerview-v7.

Перейдем к построению основы приложения. На главном экране будет отображаться сетка из кнопок, каждая из которых воспроизводит определенный звук. Нам понадобятся два файла макета: один для сетки, другой для кнопок.

Сначала создайте файл макета для RecyclerView. Файл res/layout/activity_beat_box.xml нам не понадобится, поэтому переименуйте его в fragment_beat_box.xml и заполните следующим образом:

Листинг 18.1. Создание главного файла макета (res/layout/fragment_beat_box.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v7.widget.RecyclerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_beat_box_recycler_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
 />
```

Затем создайте файл макета для кнопок res/layout/list_item_sound.xml.

Листинг 18.2. Создание макета для звуковых кнопок (res/layout/list_item_sound.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<Button
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

Теперь создайте в com.bignerdranch.android.beatbox новый фрагмент с именем BeatBoxFragment и свяжите его с только что созданным макетом.

Листинг 18.3. Создание BeatBoxFragment (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_beat_box, container,
            false);

        RecyclerView recyclerView = (RecyclerView)view
            .findViewById(R.id.fragment_beat_box_recycler_view);
        recyclerView.setLayoutManager(new GridLayoutManager
            (getActivity(), 3));

        return view;
    }
}
```

Обратите внимание: здесь используется не та реализация `LayoutManager`, которая встречалась в главе 9. Эта разновидность `LayoutManager` размещает элементы в виде сетки, так что в каждой строке отображаются сразу несколько элементов. Передаваемое значение 3 указывает, что сетка состоит из трех столбцов.

Создайте виджет `ViewHolder`, связанный с `list_item_sound.xml`.

Листинг 18.4. Создание класса `SoundHolder` (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ...
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
        ...
    }

    private class SoundHolder extends RecyclerView.ViewHolder {
        private Button mButton;
```

```

public SoundHolder(LayoutInflater inflater, ViewGroup container) {
    super(inflater.inflate(R.layout.list_item_sound, container,
        false));

    mButton = (Button)itemView.findViewById
        (R.id.list_item_sound_button);
}
}
}

```

Затем создайте адаптер, связанный с SoundHolder. (Если вы поместите курсор в RecyclerView.Adapter, прежде чем вводить любые из приведенных ниже методов, и нажмете Option+Return (Alt+Enter), Android Studio сгенерирует большую часть кода за вас.)

Листинг 18.5. Создание класса SoundAdapter (BeatBoxFragment.java)

```

public class BeatBoxFragment extends Fragment {
    ---
    private class SoundHolder extends RecyclerView.ViewHolder {
        ---
    }

    private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
        @Override
        public SoundHolder onCreateViewHolder
            (ViewGroup parent, int viewType) {
            LayoutInflater inflater = LayoutInflater.from(getActivity());
            return new SoundHolder(inflater, parent);
        }

        @Override
        public void onBindViewHolder(SoundHolder soundHolder, int position) {
        }

        @Override
        public int getItemCount() {
            return 0;
        }
    }
}

```

Подключите SoundAdapter в методе onCreateView(...).

Листинг 18.6. Подключение SoundAdapter (BeatBoxFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_beat_box, container,
        false);
}

```

```

RecyclerView recyclerView = (RecyclerView)view
    .findViewById(R.id.fragment_beat_box_recycler_view);
recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
recyclerView.setAdapter(new SoundAdapter());

return view;
}

```

Теперь нужно связать `BeatBoxFragment` с `BeatBoxActivity`. Мы используем ту же архитектуру `SingleFragmentActivity`, которая использовалась в приложении `CriminalIntent`.

Скопируйте файл `SingleFragmentActivity.java` из приложения `CriminalIntent` в `app/java/com.bignerdranch.android.beatbox`, а затем скопируйте `activity_fragment.xml` в `app/src/res/layout`. (Скопируйте эти файлы либо из папки проекта `CriminalIntent` на своем компьютере, либо из архива решений. За информацией о том, как получить доступ к файлам решений, обращайтесь к разделу «Добавление значка» главы 2.)

Теперь удалите весь код из тела класса `BeatBoxActivity`, назначьте его субклассом `SingleFragmentActivity` и переопределите метод `createFragment()`:

Листинг 18.7. Заполнение `BeatBoxActivity` (`BeatBoxActivity.java`)

```

public class BeatBoxActivity extends SingleFragmentActivity {
    @Override
    protected Fragment createFragment() {
        return BeatBoxFragment.newInstance();
    }
}

```

Этого должно быть достаточно для того, чтобы основной код приложения заработал. Ваша реализация `BeatBoxFragment` пока ничего не отображает, но вы все же запустите приложение и убедитесь в том, что все компоненты связаны правильно (рис. 18.2).

Импортирование активов

Используемые активы следует импортировать в проект. Создайте в проекте папку для активов: щелкните правой кнопкой мыши на модуле `app` и выберите команду `New ▶ Folder ▶ Assets Folder` (рис. 18.3). Оставьте флажок `Change Folder Location` снятым, а в списке `Target Source Set` выберите вариант `main`.

Щелкните на кнопке `Finish`, чтобы создать папку для активов.

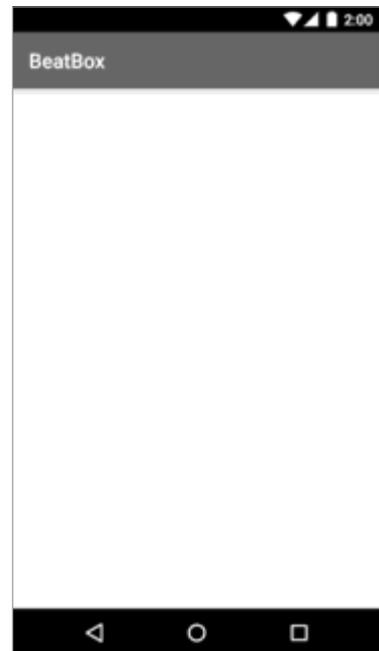


Рис. 18.2. Пустое представление `BeatBox`

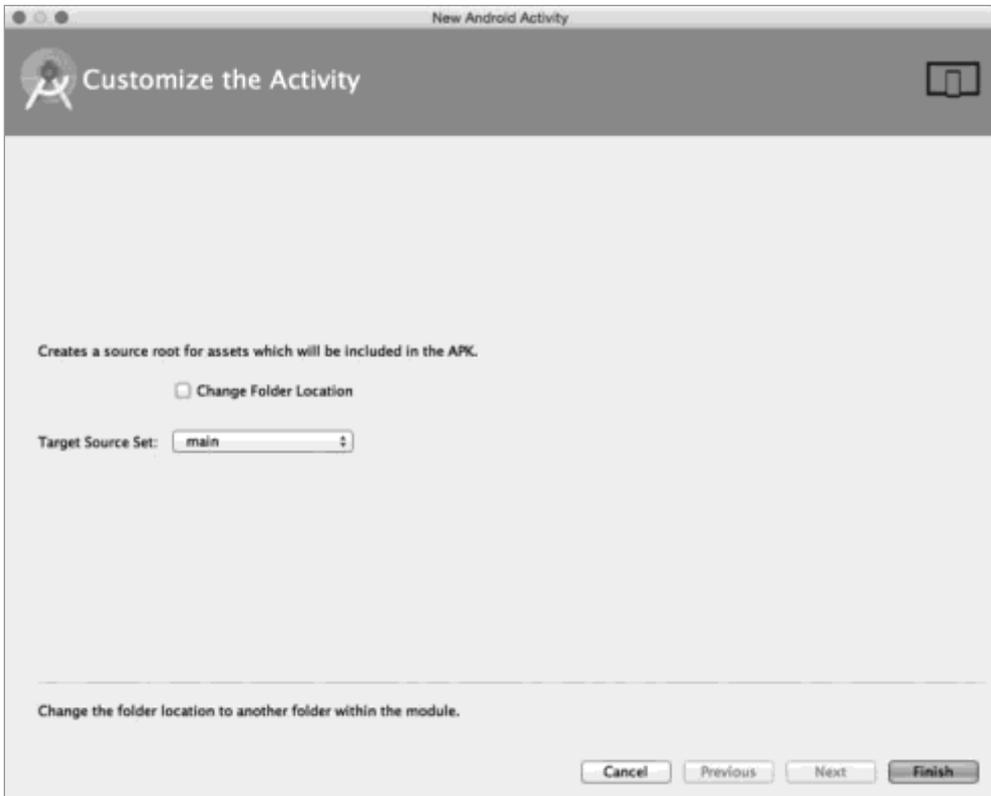


Рис. 18.3. Создание папки assets

Щелкните правой кнопкой мыши на папке assets и выберите команду New ▶ Directory. Введите имя каталога sample_sounds (рис. 18.4).



Рис. 18.4. Создание папки sample_sounds

Все содержимое папки assets интегрируется в приложение. Для удобства и порядка мы создали вложенную папку с именем sample_sounds. Впрочем, в отличие от ресурсов, делать это было не обязательно.

Где найти звуки? Мы воспользуемся подборкой, распространяемой на условиях лицензии Creative Commons, которую мы впервые увидели у пользователя

plagasul (<http://www.freesound.org/people/plagasul/packs/3/>). Мы поместили все звуки в один zip-архив по адресу:

https://www.bignerdranch.com/solutions/sample_sounds.zip

Загрузите архив и распакуйте его содержимое в папку `assets/sample_sounds` (рис. 18.5).

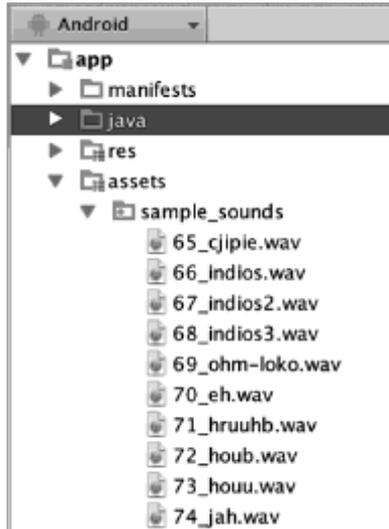


Рис. 18.5. Импортированные активы

(Кстати, проследите за тем, чтобы там находились файлы `.wav`, а не только файл `.zip`, из которого они были извлечены.)

Постройте приложение заново. Следующим шагом станет получение списка активов и вывод его для пользователя.

Получение информации об активах

Приложение будет выполнять множество операций, относящихся к управлению активами: поиск, отслеживание и в конечном итоге воспроизведение их как звуков. Для выполнения этих операций создайте новый класс с именем `BeatBox` в пакете `com.bignerdranch.android.beatbox`. Добавьте пару констант: для вывода информации в журнал и для имени папки, в которой были сохранены звуки.

Листинг 18.8. Новый класс `BeatBox` (`BeatBox.java`)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
}
```

Для обращения к активам используется класс `AssetManager`. Экземпляр этого класса можно получить для любой разновидности `Context`. Так как `BeatBox` понадобится такой экземпляр, предоставьте ему конструктор, который получает `Context`, извлекает `AssetManager` и сохраняет на будущее.

Листинг 18.9. Сохранение `AssetManager` (`BeatBox.java`)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";

    private AssetManager mAssets;

    public BeatBox(Context context) {
        mAssets = context.getAssets();
    }
}
```

Как правило, при работе с активами вам не нужно беспокоиться о том, какой именно объект `Context` будет использоваться. Во всех ситуациях, которые вам встретятся на практике, объект `AssetManager` всех разновидностей `Context` будет связан с одним набором активов.

Чтобы получить список доступных активов, используйте метод `list(String)`. Напишите метод `loadSounds()`, который обращается к активам при помощи метода `list(String)`.

Листинг 18.10. Получение списка активов (`BeatBox.java`)

```
public BeatBox(Context context) {
    mAssets = context.getAssets();
    loadSounds();
}

private void loadSounds() {
    String[] soundNames;
    try {
        soundNames = mAssets.list(SOUNDS_FOLDER);
        Log.i(TAG, "Found " + soundNames.length + " sounds");
    } catch (IOException ioe) {
        Log.e(TAG, "Could not list assets", ioe);
        return;
    }
}
```

Метод `AssetManager.list(String)` возвращает список имен файлов, содержащихся в заданной папке. Передав ему путь к папке со звуками, вы получите информацию обо всех файлах `.wav` в этой папке.

Чтобы убедиться в том, что система активов работает правильно, создайте экземпляр `BeatBox` в `BeatBoxFragment`.

Листинг 18.11. Создание экземпляра BeatBox (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {

    private BeatBox mBeatBox;

    public static BeatBoxFragment newInstance() {
        return new BeatBoxFragment();
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mBeatBox = new BeatBox(getActivity());
    }

    ...
}
```

Запустите приложение. В журнале должна появиться информация о том, сколько звуковых файлов было обнаружено. Мы использовали 22 файла в формате .wav, и если вы использовали наши файлы, результат должен выглядеть так:

```
...1823-1823/com.bignerdranch.android.beatbox I/BeatBox: Found 22 sounds
```

Подключение активов для использования

Имена файлов активов получены, теперь нужно вывести их для пользователя. В конечном итоге файлы должны воспроизводиться, поэтому стоит создать объект для хранения имени файла, того имени, которое видит пользователь, и всей остальной информации, относящейся к данному звуку.

Создайте класс Sound для хранения всей этой информации. (Не забудьте поручить Android Studio сгенерировать get-методы.)

Листинг 18.12. Создание объекта Sound (Sound.java)

```
public class Sound {
    private String mAssetPath;
    private String mName;

    public Sound(String assetPath) {
        mAssetPath = assetPath;
        String[] components = assetPath.split("/");
        String filename = components[components.length - 1];
        mName = filename.replace(".wav", "");
    }

    public String getAssetPath() {
        return mAssetPath;
    }
}
```

```
    }  
  
    public String getName() {  
        return mName;  
    }  
}
```

В конструкторе выполняется небольшая подготовительная работа для построения удобочитаемого имени звука. Сначала имя файла отделяется вызовом `String.split(String)`, после чего вызов `String.replace(String, String)` удаляет расширение.

Затем в методе `BeatBox.loadSounds()` строится список объектов `Sound`.

Листинг 18.13. Создание списка объектов `Sound` (`BeatBox.java`)

```
public class BeatBox {  
    ...  
    private AssetManager mAssets;  
    private List<Sound> mSounds = new ArrayList<>();  
  
    public BeatBox(Context context) {  
        ...  
    }  
  
    private void loadSounds() {  
        String[] soundNames;  
        try {  
            ...  
        } catch (IOException ioe) {  
            ...  
        }  
  
        for (String filename : soundNames) {  
            String assetPath = SOUNDS_FOLDER + "/" + filename;  
            Sound sound = new Sound(assetPath);  
            mSounds.add(sound);  
        }  
    }  
  
    public List<Sound> getSounds() {  
        return mSounds;  
    }  
}
```

Затем в `BeatBoxFragment` добавьте в `SoundHolder` код, связывающий экземпляр с объектом `Sound`.

Листинг 18.14. Связывание с объектом `Sound` (`BeatBoxFragment.java`)

```
private class SoundHolder extends RecyclerView.ViewHolder {  
    private Button mButton;  
    private Sound mSound;
```

```

public SoundHolder(LayoutInflater inflater, ViewGroup container) {
    ...
}

public void bindSound(Sound sound) {
    mSound = sound;
    mButton.setText(mSound.getName());
}
}

```

Свяжите SoundAdapter со списком объектов Sound.

Листинг 18.15. Связывание со списком Sound (BeatBoxFragment.java)

```

private class SoundAdapter extends RecyclerView.Adapter<SoundHolder> {
    private List<Sound> mSounds;

    public SoundAdapter(List<Sound> sounds) {
        mSounds = sounds;
    }
    ...

    @Override
    public void onBindViewHolder(SoundHolder soundHolder, int position) {
        Sound sound = mSounds.get(position);
        soundHolder.bindSound(sound);
    }

    @Override
    public int getItemCount() {
        return 0;
        return mSounds.size();
    }
}

```

Передайте звуки из BeatBox в методе onCreateView(...).

Листинг 18.16. Передача звуков адаптеру (BeatBoxFragment.java)

```

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_beat_box, container, false);

    RecyclerView recyclerView = (RecyclerView)view
        .findViewById(R.id.fragment_beat_box_recycler_view);
    recyclerView.setLayoutManager(new GridLayoutManager(getActivity(), 3));
    recyclerView.setAdapter(new SoundAdapter());
    recyclerView.setAdapter(new SoundAdapter(mBeatBox.getSounds()));

    return view;
}

```

После добавления этого кода при запуске BeatBox на экране появляется сетка со звуковыми файлами (рис. 18.6).



Рис. 18.6. Готовый интерфейс BeatBox

Обращение к ресурсам

Вся работа этой главы успешно завершена. В следующей главе приложение BeatBox начнет непосредственную работу с содержимым активов.

Но перед этим давайте чуть подробнее поговорим о том, как работают активы.

С объектом `Sound` связан некоторый путь к файлу. Попытка открыть такой файл с использованием объекта `File` завершается неудачей; для работы с ним следует использовать `AssetManager`:

```
String assetPath = sound.getAssetPath();  
InputStream soundData = mAssets.open(assetPath);
```

В результате вы получаете стандартный объект `InputStream` для данных, с которым можно работать в коде точно так же, как с любым другим `InputStream`.

Некоторые API вместо объекта `InputStream` требуют объект `FileDescriptor`. (Именно он будет использоваться при работе с `SoundPool` в следующей главе.) В такой ситуации можно вызвать метод `AssetManager.openFd(String)`:

```
String assetPath = sound.getAssetPath();
```

```
// Объекты AssetFileDescriptor отличаются от FileDescriptor,  
AssetFileDescriptor assetFd = mAssets.openFd(assetPath);  
// но при необходимости вы можете легко получить  
// обычный объект FileDescriptor.  
FileDescriptor fd = assetFd.getFileDescriptor();
```

Для любознательных: «не-активы»?

В классе `AssetManager` определены методы с именем `openNonAssetFd(...)`. Возникает резонный вопрос: зачем классу, предназначенному для работы с активами, методы для работы с «не-активами» (`non-assets`)? В принципе, мы могли бы ответить: «Не обращайтесь внимания» и попытаться убедить вас в том, что вы никогда не слышали о существовании `openNonAssetFd(...)`.

Нам не известны никакие причины, по которым вам могли бы понадобиться эти методы, так что для их изучения тоже нет никаких реальных причин.

Однако вы купили нашу книгу, так что мы приведем ответ — просто для полноты картины.

Помните, ранее мы говорили о том, что в Android существуют две разные системы: активы и ресурсы? В системе ресурсов хорошо реализован механизм подбора и сопоставления. Однако некоторые большие ресурсы (как правило, изображения и необработанные звуковые файлы) слишком велики, поэтому фактически они хранятся в виде активов.

Во внутренней реализации Android открывает эти ресурсы для своих целей методами `openNonAsset`, часть из которых недоступна для внешнего пользователя.

Когда вам могут понадобиться эти методы? Насколько нам известно — никогда. По крайней мере, теперь вы знаете почему.

19

Воспроизведение аудио с использованием SoundPool

Теперь, когда вы освоили работу с активами, можно заняться непосредственным воспроизведением файлов .wav. Звуковые API Android в основном работают на достаточно низком уровне, но для приложения, которое мы создаем, существует практически идеальный кандидат: SoundPool.

Класс SoundPool позволяет загрузить в память большой набор звуков и управлять максимальным количеством звуков, воспроизводимых одновременно. Таким образом, если пользователь войдет в азарт и начнет жать все кнопки одновременно, это не приведет к сбою приложения или чрезмерному расходованию ресурсов на телефоне.

Готовы? Пора начинать.

Создание объекта SoundPool

Начнем с создания объекта SoundPool.

Листинг 19.1. Создание объекта SoundPool (Sound.java)

```
public class BeatBox {
    private static final String TAG = "BeatBox";

    private static final String SOUNDS_FOLDER = "sample_sounds";
    private static final int MAX_SOUNDS = 5;

    private AssetManager mAssets;
    private List<Sound> mSounds;
    private SoundPool mSoundPool;

    public BeatBox(Context context) {
        mAssets = context.getAssets();
        // Этот конструктор считается устаревшим,
        // но он нужен для обеспечения совместимости.
        mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
    }
}
```

```

        loadSounds();
    }
    ...
}

```

В Lollipop появился новый способ создания объектов SoundPool с использованием класса SoundPool.Builder. Но поскольку SoundPool.Builder не доступен в минимальной поддерживаемой версии API 16, мы вместо этого используем более старый конструктор SoundPool(int, int, int).

Первый параметр определяет, сколько звуков может воспроизводиться в любой момент времени. В данном случае передается значение 5. Если пять звуков воспроизводятся одновременно и вы попытаетесь воспроизвести шестой, SoundPool прекращает воспроизведение самого старого.

Второй параметр определяет тип аудиопотока, который может воспроизводиться объектом SoundPool. В Android поддерживаются разные аудиопотоки, каждый из которых обладает независимыми настройками громкости. Вот почему снижение громкости музыки не приводит к снижению громкости сигналов. За информацией о других вариантах обращайтесь к описанию констант AUDIO_* в документации AudioManager. STREAM_MUSIC устанавливает тот же уровень громкости, что у музыки и игр на устройстве.

А последний параметр? Он задает качество дискретизации. В документации сказано, что этот параметр игнорируется, поэтому в нем можно передать 0.

Загрузка звуков

Следующий шаг — загрузка звуков в SoundPool. Главное преимущество класса SoundPool перед другими механизмами воспроизведения звука — быстрая реакция: когда вы приказываете ему воспроизвести звук, воспроизведение начинается немедленно, без задержки.

За это приходится расплачиваться необходимостью загрузки звуков в SoundPool перед воспроизведением. Каждому загружаемому звуку назначается собственный целочисленный идентификатор. Добавьте в Sound поле mSoundId и сгенерированные get- и set-методы для работы с ним.

Листинг 19.2. Создание объекта SoundPool (Sound.java)

```

public class Sound {
    private String mAssetPath;
    private String mName;
    private Integer mSoundId;
    ...

    public String getName() {
        return mName;
    }

    public Integer getSoundId() {

```

```

        return mSoundId;
    }

    public void setSoundId(Integer soundId) {
        mSoundId = soundId;
    }
}

```

Объявление `mSoundId` с типом `Integer` вместо `int` позволяет представить неопределенное состояние `Sound` — для этого `mSoundId` присваивается `null`.

Теперь можно переходить к загрузке звуков. Добавьте в `BeatBox` метод `load(Sound)` для загрузки `Sound` в `SoundPool`.

Листинг 19.3. Загрузка звуков в `SoundPool` (`BeatBox.java`)

```

private void loadSounds() {
    ...
}

private void load(Sound sound) throws IOException {
    AssetFileDescriptor afd = mAssets.openFd(sound.getAssetPath());
    int soundId = mSoundPool.load(afd, 1);
    sound.setSoundId(soundId);
}
}

```

Вызов `mSoundPool.load(AssetFileDescriptor, int)` загружает файл в `SoundPool` для последующего воспроизведения. Для управления звуком, его повторного воспроизведения (или выгрузки) `mSoundPool.load(...)` возвращает идентификатор типа `int`, который сохраняется в только что определенном поле `mSoundId`. А так как вызов `openFd(String)` инициирует `IOException`, `load(Sound)` тоже инициирует `IOException`.

Загрузите все звуки, вызывая метод `load(Sound)` в `BeatBox.loadSounds()`.

Листинг 19.4. Загрузка всех звуков (`BeatBox.java`)

```

private void loadSounds() {
    ...

    mSounds = new ArrayList<Sound>();
    for (String filename : soundNames) {
        try {
            String assetPath = SOUNDS_FOLDER + "/" + filename;
            Sound sound = new Sound(assetPath);
            load(sound);
            mSounds.add(sound);
        } catch (IOException ioe) {
            Log.e(TAG, "Could not load sound " + filename, ioe);
        }
    }
}
}

```

Запустите приложение BeatBox и убедитесь в том, что все звуки были загружены правильно. Если при загрузке произошла ошибка, на панели LogCat появятся красные сообщения об исключениях.

Воспроизведение звуков

Остается последний шаг: воспроизвести загруженные звуки. Добавьте в BeatBox метод `play(Sound)`.

Листинг 19.5. Воспроизведение звуков (BeatBox.java)

```

    mSoundPool = new SoundPool(MAX_SOUNDS, AudioManager.STREAM_MUSIC, 0);
    loadSounds();
}

public void play(Sound sound) {
    Integer soundId = sound.getSoundId();
    if (soundId == null) {
        return;
    }
    mSoundPool.play(soundId, 1.0f, 1.0f, 1, 0, 1.0f);
}

public List<Sound> getSounds() {
    return mSounds;
}

```

Прежде чем воспроизводить звук с идентификатором `soundId`, необходимо сначала убедиться в том, что он отличен от `null`. Такое возможно, если объект `Sound` не удалось загрузить.

Если вы уверены, что значение отлично от `null`, воспроизведите звук вызовом `SoundPool.play(int, float, float, int, int, float)`. Параметры содержат соответственно: идентификатор звука, громкость слева, громкость справа, приоритет (игнорируется), признак циклического воспроизведения и скорость воспроизведения.

Для полной громкости и нормальной скорости воспроизведения передайте `1.0`. Передача `0` в признаке циклического воспроизведения означает «без заикливания». (Передайте `-1`, если хотите, чтобы воспроизведение длилось бесконечно долго. Мы считаем, что это только раздражает.)

После написания такого метода вы сможете организовать воспроизведение звука при нажатии одной из кнопок.

Листинг 19.6. Воспроизведение звука при нажатии кнопки (BeatBoxFragment.java)

```

private class SoundHolder extends RecyclerView.ViewHolder
    implements View.OnClickListener {
    private Button mButton;
    private Sound mSound;
}

```

```
public SoundHolder(LayoutInflater inflater, ViewGroup container) {
    super(inflater.inflate(R.layout.list_item_sound, parent, false));
    mButton = (Button)itemView.findViewById(R.id.button);
    mButton.setOnClickListener(this);
}

public void bindSound(Sound sound) {
    mSound = sound;
    mButton.setText(mSound.getName());
}

@Override
public void onClick(View v) {
    mBeatBox.play(mSound);
}
}
```

Нажмите кнопку (рис. 19.1) — вы должны услышать воспроизводимый звук.



Рис. 19.1. Действующая библиотека звуков

Выгрузка звуков

Приложение работает, но мы еще должны прибраться за собой. Сознательное приложение должно освободить ресурсы SoundPool вызовом SoundPool.release() после завершения работы. Добавьте соответствующий метод BeatBox.release().

Листинг 19.7. Освобождение SoundPool (BeatBox.java)

```
public class BeatBox {
    ---
    public void play(Sound sound) {
        ---
    }

    public void release() {
        mSoundPool.release();
    }
    ---
}
```

Освобождение ресурсов производится после завершения работы с SoundPool в BeatBoxFragment.

Листинг 19.8. Освобождение BeatBox (BeatBoxFragment.java)

```
public class BeatBoxFragment extends Fragment {
    ---
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ---
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        mBeatBox.release();
    }
    ---
}
```

Запустите приложение и убедитесь в том, что оно правильно работает с новым методом release().

Повороты и преемственность объектов

Ваше приложение ведет себя цивилизованно, и это хорошо. К сожалению, оно перестало правильно обрабатывать повороты. Попробуйте воспроизвести звук

69_ohm-локо и повернуть устройство: воспроизведение неожиданно прерывается. (Если этого не происходит, убедитесь в том, что приложение было построено и запущено с последней реализацией `onDestroy()`.)

Проблема заключается в следующем: при повороте `BeatBoxActivity` уничтожается. При этом `FragmentManager` также уничтожает и `BeatBoxFragment`. При этом вызываются методы прекращения жизненного цикла `BeatBoxFragment`: `onPause()`, `onStop()` и `onDestroy()`. В `BeatBoxFragment.onDestroy()` вызывается `BeatBox.release()`, что приводит к освобождению `SoundPool` и остановке воспроизведения.

Вы уже видели, как экземпляры `Activity` и `Fragment` «умирают» при поворотах; тогда проблема решалась при помощи `onSaveInstanceState(Bundle)`. Однако на этот раз такое решение не сработает, потому что оно основано на сохранении и восстановлении данных `Parcelable` в объекте `Bundle`.

`Parcelable`, как и `Serializable`, представляет собой API для сохранения объекта в потоке байтов. В Java объекты сохраняются либо посредством включения в `Bundle`, либо пометкой объекта `Serializable`, либо реализацией интерфейса `Parcelable`. Какой бы способ вы ни выбрали, всегда действует один принцип: ни один из этих инструментов не следует использовать, если объект не является *сохраняемым* (*stashable*).

Чтобы вы лучше поняли, что имеется в виду, представьте, что вы смотрите телевизионную передачу с другом. Вы можете записать канал, который вы смотрите, уровень громкости, настройку цветов и т. д. Даже если сработает пожарная сигнализация и выключится электричество, вы сможете позднее прочитать записанные данные и продолжить просмотр так, словно ничего не произошло.

Таким образом, конфигурация телевизора является *сохраняемой*. С другой стороны, время, проведенное за просмотром, для сохранения не пригодно: если выключится питание, сеанс завершается. Позднее вы можете вернуться и создать новый сеанс, но просмотр будет прерван, что бы вы ни делали. Следовательно, сеанс *не* является сохраняемым.

Некоторые части `BeatBox` являются сохраняемыми: например, все содержимое `SoundPool` сохраняется. Однако `SoundPool` больше напоминает сеанс просмотра. Да, вы можете создать новый объект `SoundPool`, в котором загружены те же звуки, что и в старом. Вы даже можете снова начать воспроизведение с того места, в котором оно было прервано. Однако при этом воспроизведение всегда будет ненадолго прерываться, что бы вы ни делали. Это означает, что класс `SoundPool` не является сохраняемым.

Несохраняемость имеет склонность к распространению. Если несохраняемый объект критичен для деятельности другого объекта, то и другой объект, скорее всего, не является сохраняемым. Класс `BeatBox` имеет ту же цель, что и `SoundPool`: он воспроизводит звуки. Из этого следует, что класс `BeatBox` не является сохраняемым. (Нам очень жаль.)

Обычный механизм `saveInstanceState` работает с сохраняемыми данными, но класс `BeatBox` таковым не является. Экземпляр `BeatBox` должен оставаться постоянно доступным при создании и уничтожении вашей активности.

Что же делать?

Удержание фрагмента

К счастью, у фрагментов существует свойство `retainInstance`, которое позволит сохранить экземпляр `BeatBox` между изменениями конфигурации. Переопределите `BeatBoxFragment.onCreate(...)` и задайте свойство фрагмента.

Листинг 19.9. Вызов `setRetainInstance(true)` (`BeatBoxFragment.java`)

```

...
public static BeatBoxFragment newInstance() {
    return new BeatBoxFragment();
}

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setRetainInstance(true);

    mBeatBox = new BeatBox(getActivity());
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {
    ...

```

По умолчанию свойство `retainInstance` у фрагментов равно `false`. Это означает, что фрагмент не удерживается, а уничтожается и создается заново при поворотах вместе с активностью-хостом. Вызов `setRetainInstance(true)` *удерживает* фрагмент. Такой фрагмент не уничтожается вместе с активностью, а сохраняется и передается новой активности.

При удержании фрагмента можно рассчитывать на то, что все его переменные экземпляров (такие, как `mBeatBox`) сохраняют прежние значения. Когда вы обратитесь к ним, они просто будут находиться на своих местах.

Снова запустите приложение `BeatBox`. Воспроизведите звук `69_ohm-loko`, поверните устройство и убедитесь в том, что воспроизведение продолжается беспрепятственно.

Повороты и удержание фрагментов

Давайте повнимательнее разберемся с тем, как работают удерживаемые фрагменты. В удержании используется тот факт, что представление фрагмента может быть уничтожено и создано заново без уничтожения самого фрагмента.

Во время изменения конфигурации `FragmentManager` сначала уничтожает представления фрагментов из своего списка. Представления фрагментов всегда уничтожаются и создаются заново при изменении конфигурации по тем же причинам, по которым уничтожаются и создаются заново представления активностей: в новой конфигурации могут быть задействованы новые ресурсы. На случай появления ресурсов, лучше соответствующих новой конфигурации, представление строится заново.

Затем `FragmentManager` проверяет свойство `retainInstance` каждого фрагмента. Если оно равно `false` (по умолчанию), то `FragmentManager` уничтожает экземпляр фрагмента. Фрагмент и его представление будут созданы заново новым экземпляром `FragmentManager` новой активности «по ту сторону» (рис. 19.2).

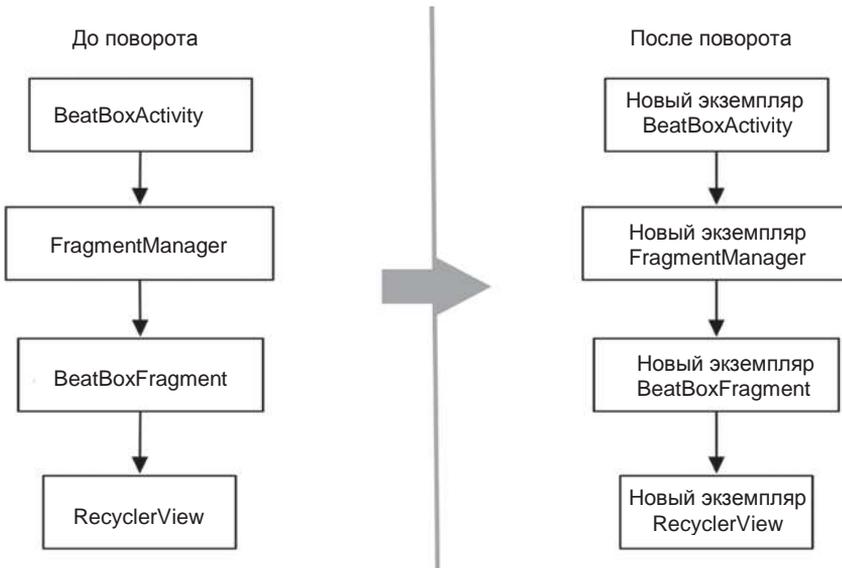


Рис. 19.2. Фрагмент пользовательского интерфейса при выполнении поворота (поведение по умолчанию)



Рис. 19.3. Фрагмент пользовательского интерфейса при выполнении поворота (с удержанием)

С другой стороны, если свойство `retainInstance` равно `true`, то уничтожается только представление фрагмента, но не сам фрагмент. При создании новой активности новый экземпляр `FragmentManager` находит удерживаемый фрагмент и создает заново его представление (рис. 19.3).

Удерживаемый фрагмент не уничтожается, а *отсоединяется* (`detached`) от гибнущей активности. При этом фрагмент переходит в удерживаемое состояние: фрагмент продолжает существовать, но не имеет активности-хоста (рис. 19.4).

Переход в состояние удержания происходит только при выполнении двух условий:

- ① Для фрагмента был вызван метод `setRetainInstance(true)`.
- ① Активность-хост уничтожается при изменении конфигурации (обычно повороте).

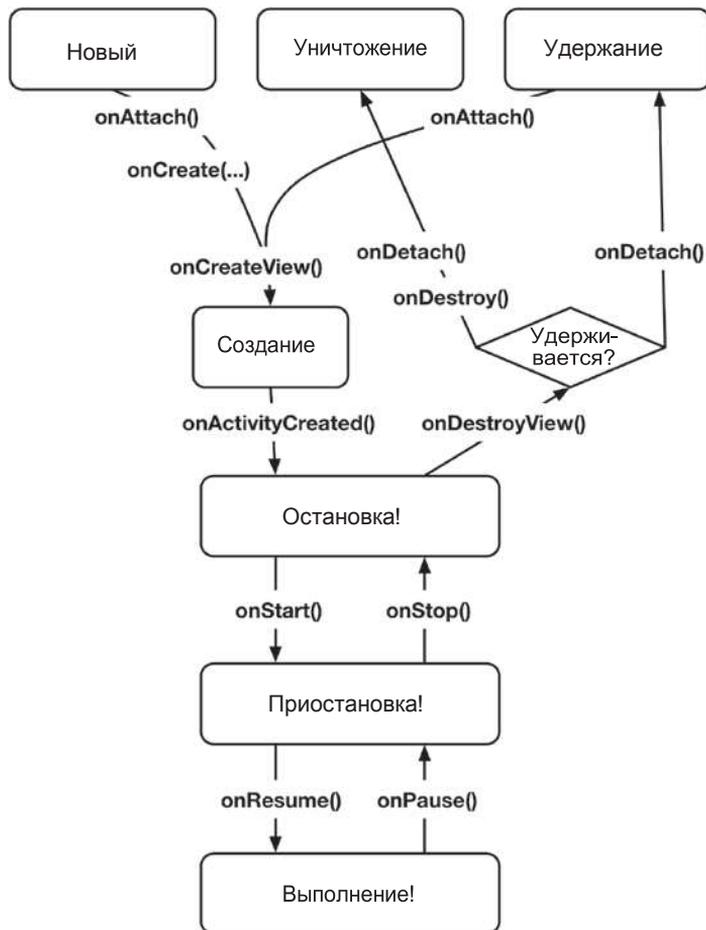


Рис. 19.4. Жизненный цикл фрагмента

Фрагмент находится в состоянии удержания в течение очень короткого времени — между отсоединением от старой активности и присоединением к новой, немедленно создаваемой активности.

Для любознательных: когда удерживать фрагменты

Удерживаемые фрагменты: удобно, верно? Да! Очень удобно. На первый взгляд они решают все проблемы, возникающие при уничтожении активностей и фрагментов при поворотах. При изменении конфигурации устройства создание нового представления обеспечивает подбор наиболее подходящих ресурсов, а у вас по-прежнему простой способ сохранения данных и объектов.

Возникает вопрос: почему бы не удерживать все фрагменты или почему фрагменты не удерживаются по умолчанию? Тем не менее мы рекомендуем использовать этот механизм только при крайней необходимости.

Во-первых, удерживаемые фрагменты попросту сложнее обычных. Когда с ними возникают проблемы, вам будет труднее разобраться в причинах происходящего. Программы всегда оказываются более сложными, чем нам хотелось бы, и когда можно обойтись без лишних сложностей — лучше так и поступить.

Во-вторых, фрагменты, обрабатывающие повороты с использованием сохранения состояния экземпляров, работают в любых ситуациях жизненного цикла, тогда как удерживаемые фрагменты работают только в том случае, когда активность уничтожается при изменении конфигурации. Если ваша активность уничтожается из-за того, что ОС понадобилось освободить память, то все удерживаемые фрагменты тоже будут уничтожены; это может привести к потере данных.

Для любознательных: подробнее о поворотах

`onSaveInstanceState(Bundle)` — еще один инструмент, который мы использовали для обработки поворотов. Собственно, если в вашем приложении не возникают проблемы с вращением, то это объясняется тем, что поведение `onSaveInstanceState(...)` по умолчанию работает успешно.

Приложение `CriminalIntent` служит хорошим примером. Фрагмент `CrimeFragment` не удерживается, но при внесении изменений в описание или переключении флажка новые состояния этих объектов `View` автоматически сохраняются и восстанавливаются после поворота. Метод `onSaveInstanceState(...)` был создан для решения именно этой задачи — сохранения и последующего восстановления пользовательского интерфейса приложения.

Главное различие между переопределением `Fragment.onSaveInstanceState(...)` и удержанием фрагмента связано с продолжительностью хранения данных. Если данные должны просто пережить изменение конфигурации, то удержание фрагмента потребует намного меньшей работы. Истинность этого утверждения осо-

бенно очевидна при сохранении объектов; вам не придется беспокоиться о том, реализует ли объект `Serializable`.

Но если данные должны «выжить» в течение большого времени, удержание фрагмента не поможет. Если активность уничтожается для освобождения памяти после того, как пользователь какое-то время бездействовал, удерживаемые фрагменты будут уничтожены точно так же, как и их обычные собратья.

Чтобы это различие стало более очевидным, вспомните приложение `GeoQuiz`. Проблема с поворотом, с которой мы столкнулись, заключалась в том, что индекс вопроса при повороте обнулялся. До какого бы вопроса ни добрался пользователь, при повороте устройства он снова возвращался к первому вопросу. Тогда мы сохранили индекс и снова загрузили его, чтобы пользователь увидел правильный вопрос.

В приложении `GeoQuiz` фрагменты не использовались, но представьте себе переработанную версию `GeoQuiz` с фрагментом `QuizFragment`, хостом которого является `QuizActivity`. Как поступить: переопределить `Fragment.onSaveInstanceState(...)` для сохранения индекса или удержать `QuizFragment`, чтобы переменная продолжала существовать?

Нарис. 19.5 изображены три разных срока жизни, с которыми вам придется иметь дело: срок жизни объекта активности (и его неудерживаемых фрагментов), срок жизни удерживаемого фрагмента и срок жизни записи активности.

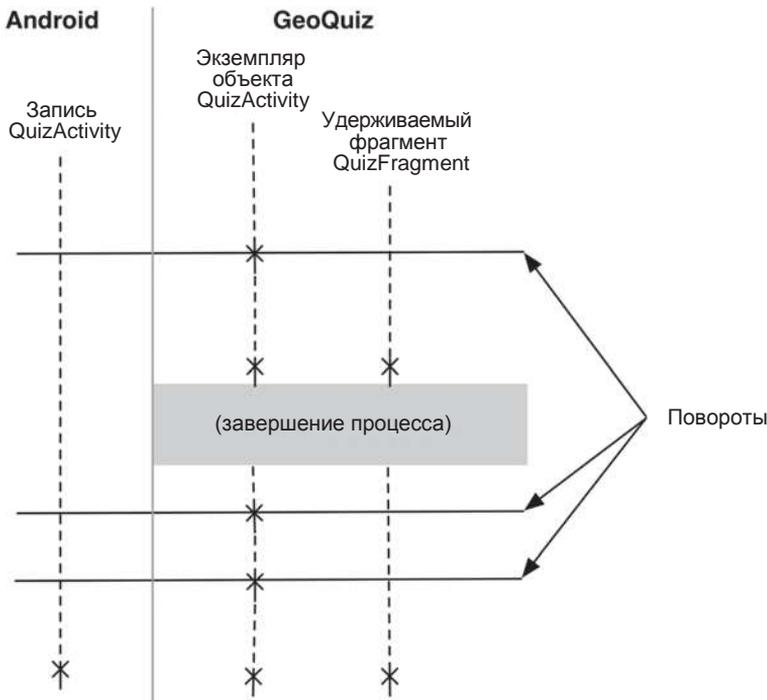


Рис. 19.5. Три срока жизни

Срок жизни объекта активности слишком короток. Это обстоятельство становится источником проблемы с поворотом: индекс определенно должен пережить объект активности.

Если удерживать QuizFragment, то срок жизни индекса будет в точности равен сроку жизни удерживаемого фрагмента. Когда GeoQuiz содержит всего пять вопросов, удержание QuizFragment оказывается самым простым решением и требует минимума кода. Вы инициализируете индекс как переменную класса и вызываете `setRetainInstance(true)` в `QuizFragment.onCreate(...)`.

Листинг 19.10. Удержание гипотетического фрагмента QuizFragment

```
public class QuizFragment extends Fragment {
    ---
    private int mCurrentIndex = 0;
    ---

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setRetainInstance(true);
    }

    ---
}
```

Привязка индекса к сроку жизни удерживаемого фрагмента позволяет ему пережить уничтожение объекта активности и решает проблему обнуления индекса при повороте. Но как видно из рис. 19.5, удержание QuizFragment не спасает значение индекса при завершении процесса, которое может произойти, если пользователь оставит приложение на некоторое время, а активность и удерживаемый фрагмент будут уничтожены для освобождения памяти.

Если вопросов всего пять, пользователь может без особых проблем начать заново. Но что, если GeoQuiz содержит 100 вопросов? Необходимость возвращаться к началу и отвечать на все вопросы заново вызовет у пользователя обоснованное раздражение. Состояние индекса должно пережить время жизни записи активности. Для этого индекс должен сохраняться в `onSaveInstanceState(...)`. Если пользователь оставит приложение на некоторое время, он всегда сможет продолжить с того места, на котором остановился.

Итак, если какие-либо данные активности или фрагмента должны существовать в течение длительного времени, свяжите их со сроком жизни записи активности — переопределите метод `onSaveInstanceState(Bundle)` и сохраните состояние для того, чтобы восстановить его в будущем.

20

Стили и темы

Приложение BeatBox эффектно звучит; пора придать ему столь же эффектный внешний вид.

До настоящего момента в BeatBox использовалось стандартное стилевое оформление пользовательского интерфейса. Кнопки — стандартные. Цвета — стандартные. Приложение ничем не выделяется на общем фоне. У него нет своего «лица».

К счастью, ситуацию можно изменить. У нас для этого имеются подходящие технологии. На рис. 20.1 изображено заметно улучшенное — или по крайней мере более стильное — приложение BeatBox.

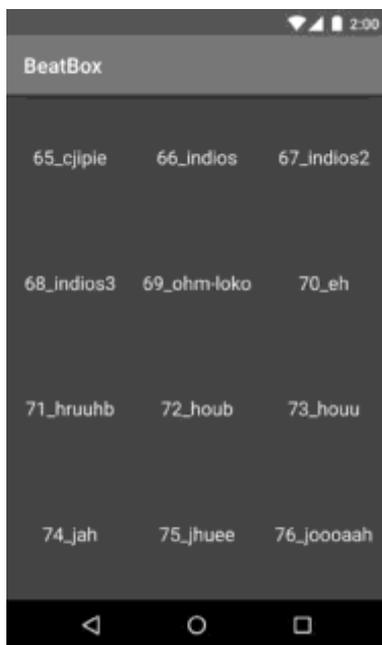


Рис. 20.1. Приложение BeatBox с новым оформлением

Цветовые ресурсы

Начнем с определения нескольких цветов, которые будут использоваться в этой главе. Создайте файл `colors.xml` в папке `res/values`.

Листинг 20.1. Определение цветов (`res/values/colors.xml`)

```
<resources>
  <color name="red">#F44336</color>
  <color name="dark_red">#C3352B</color>
  <color name="gray">#607D8B</color>
  <color name="soothing_blue">#0083BF</color>
  <color name="dark_blue">#005A8A</color>
</resources>
```

Цветовые ресурсы удобны тем, что вы можете в одном месте определить цветовые значения, которые затем будут использоваться в разных местах приложения.

Стили

А теперь займемся изменением внешнего вида кнопок `BeatBox`. *Стиль* (`style`) представляет собой набор атрибутов, которые могут применяться к виджетам.

Откройте файл `res/values/styles.xml` и добавьте стиль с именем `BeatBoxButton`. (При создании приложения `BeatBox` в новый проект должен быть включен встроенный файл `styles.xml`. Если в вашем проекте его нет, создайте этот файл.)

Листинг 20.2. Добавление стиля (`res/values/styles.xml`)

```
<resources>
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
  </style>

  <style name="BeatBoxButton">
    <item name="android:background">@color/dark_blue</item>
  </style>
</resources>
```

Мы создали стиль с именем `BeatBoxButton`. Этот стиль определяет всего один атрибут `android:background`, которому назначается темно-синий цвет. Вы можете применить этот стиль к любому количеству виджетов, а потом обновить атрибуты всех этих виджетов в одном месте.

Примените стиль `BeatBoxStyle` к кнопкам приложения `BeatBox`.

Листинг 20.3. Использование стиля (`res/layout/list_item_sound.xml`)

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  style="@style/BeatBoxButton"
  android:id="@+id/button"
```

```
android:layout_width="match_parent"  
android:layout_height="120dp"  
tools:text="Sound name"/>
```

Запустив BeatBox, вы увидите, что все кнопки окрасились в темно-синий цвет фона (рис. 20.2).



Рис. 20.2. Приложение BeatBox со стилями кнопок

Стиль можно создать для любого набора атрибутов, которые должны многократно использоваться в приложении. Что и говорить, удобно.

Наследование стилей

Стили также поддерживают наследование. Стиль может наследовать и переопределять атрибуты из других стилей.

Создайте новый стиль с именем `BeatBoxButton.Strong`, который наследует от `BeatBoxButton`, но дополнительно выделяет текст полужирным шрифтом.

Листинг 20.4. Наследование от `BeatBoxButton` (`res/layout/styles.xml`)

```
...  
<style name="BeatBoxButton">  
  <item name="android:background">@color/dark_blue</item>  
</style>
```

```
<style name="BeatBoxButton.Strong">
  <item name="android:textStyle">bold</item>
</style>
...

```

(Хотя атрибут `android:textStyle` также можно было добавить в стиль `BeatBoxButton` напрямую, мы создали `BeatBoxButton.Strong` для демонстрации механизма наследования стилей.)

Схема формирования имен в данном случае выглядит немного странно. Когда вы присваиваете стилю имя `BeatBoxButton.Strong`, вы тем самым указываете, что он наследует атрибуты от `BeatBoxButton`.

Также существует альтернативный механизм формирования имен при наследовании — с явным указанием родителя при объявлении стиля:

```
<style name="BeatBoxButton">
  <item name="android:background">@color/dark_blue</item>
</style>

<style name="StrongBeatBoxButton" parent="@style/BeatBoxButton">
  <item name="android:textStyle">bold</item>
</style>

```

В приложении `BeatBox` будет использоваться схема `BeatBoxButton.Strong`.

Обновите файл `list_item_sound.xml`, чтобы использовать новый стиль с жирным шрифтом.

Листинг 20.5. Применение жирного шрифта (`res/layout/list_item_sound.xml`)

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  style="@style/BeatBoxButton.Strong"
  android:id="@+id/button"
  android:layout_width="match_parent"
  android:layout_height="120dp"
  tools:text="Sound name"/>

```

Запустите приложение `BeatBox` и убедитесь в том, что текст на кнопках действительно выводится жирным шрифтом (рис. 20.3).

Темы

Стили — классная штука. Они позволяют определить набор атрибутов в одном месте, а затем применить их к любому количеству виджетов на ваше усмотрение. Впрочем, у них есть и недостаток: вам придется последовательно применять их к каждому виджету. А если вы пишете сложное приложение с множеством кнопок в многочисленных макетах? Добавление стиля `BeatBoxButton` ко всем кнопкам станет весьма масштабной задачей.

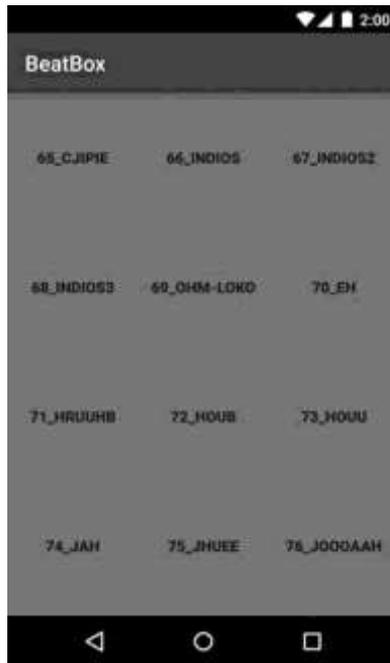


Рис. 20.3. Приложение BeatBox с жирным шрифтом

На помощь приходят *темы* (themes). Темы идут еще дальше по сравнению со стилями: они, как и стили, позволяют определить набор атрибутов в одном месте, но затем эти атрибуты автоматически применяются во всем приложении. В атрибутах темы могут содержаться ссылки на конкретные ресурсы (например, цвета), а также ссылки на стили. Например, в определении темы можно сказать: «Я хочу, чтобы все кнопки использовали *этот* стиль». И вам не придется отыскивать каждый виджет кнопки и приказывать ему использовать указанную тему.

Изменение темы

При создании приложения BeatBox ему была назначена тема по умолчанию. Откройте файл `AndroidManifest.xml` и найдите атрибут `theme` в теге `application`.

Листинг 20.6. Тема BeatBox (`AndroidManifest.xml`)

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.bignerdranch.android.beatbox" >

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

```
    ...  
</application>  
  
</manifest>
```

Атрибут `theme` ссылается на тему с именем `AppTheme`. Тема `AppTheme` была объявлена в файле `styles.xml`, который мы изменили ранее.

Как видите, тема также является стилем. Однако темы определяют другие атрибуты (вскоре вы убедитесь в этом). Кроме того, определение тем в манифесте наделяет их суперспособностями: именно этот факт позволяет автоматически применить тему в границах целого приложения.

Перейдите к определению темы `AppTheme`, щелкнув с нажатой клавишей `Command` (или `Ctrl` в `Windows`) на `@style/AppTheme`. `Android Studio` открывает `res/values/styles.xml`.

Листинг 20.7. Тема `AppTheme` в `BeatBox` (`res/values/styles.xml`)

```
<resources>  
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
    </style>  
  
    <style name="BeatBoxButton">  
        <item name="android:background">@color/dark_blue</item>  
    </style>  
  
    ...  
</resources>
```

(На момент написания книги новым проектам, создаваемым в `Android Studio`, назначалась тема `AppCompat`. Если тема `AppCompat` отсутствует в вашем решении, выполните инструкции из главы 13 и переведите `BeatBox` на использование библиотеки `AppCompat`.)

`AppTheme` наследует атрибуты от `Theme.AppCompat.Light.DarkActionBar`. В `AppTheme` можно добавлять или переопределять отдельные значения родительской темы.

Библиотека `AppCompat` включает три основные темы:

- ① `Theme.AppCompat` — темная тема;
- ① `Theme.AppCompat.Light` — светлая тема;
- ① `Theme.AppCompat.Light.DarkActionBar` — светлая тема с темной панелью инструментов.

Замените родительскую тему на `Theme.AppCompat`, чтобы в `BeatBox` использовалась базовая темная тема.

Листинг 20.8. Переключение на темную тему (`res/values/styles.xml`)

```
<resources>  
  
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
```

```
</style>  
...  
</resources>
```

Запустите BeatBox и посмотрите, как выглядит темная тема (рис. 20.4).



Рис. 20.4. BeatBox с темной темой

Добавление цветов в тему

Разобравшись с выбором базовой темы, мы переходим к настройке атрибутов темы AppTheme приложения BeatBox.

В файле styles.xml определите три атрибута вашей темы.

Листинг 20.9. Изменение атрибутов темы (res/values/styles.xml)

```
<resources>  
  <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  
    <item name="colorPrimary">@color/red</item>  
    <item name="colorPrimaryDark">@color/dark_red</item>  
    <item name="colorAccent">@color/gray</item>  
  </style>  
  ...  
</resources>
```

Здесь определяются три атрибута темы. Эти атрибуты похожи на атрибуты стилей, которыми мы занимались ранее, но они задают другие свойства. Атрибуты стиля задают свойства индивидуального виджета — как, например, `textStyle` при выделении текста кнопки жирным шрифтом. Атрибуты темы имеют более широкую область действия: это свойства, которые задаются на уровне темы и становятся доступными для любых виджетов. Например, панель инструментов обращается к атрибуту темы `colorPrimary` для назначения своего цвета фона.

Назначение этих трех атрибутов приводит к масштабным последствиям. Атрибут `colorPrimary` определяет первичный цвет фирменного стиля вашего приложения. Этот цвет будет использоваться для фона панели инструментов, а также в нескольких других местах. Атрибут `colorPrimaryDark` используется для окраски строки состояния, отображаемой у верхнего края экрана. Обычно цвет `colorPrimaryDark` представляет собой чуть более темную версию цвета `colorPrimary`. Тематическое оформление строки состояния относится к числу возможностей, добавленных в Android в Lollipop. Помните, что строка состояния будет окрашена в черный цвет на старых устройствах (независимо от настроек темы). На рис. 20.5 показан эффект от назначения этих двух атрибутов темы в приложении BeatBox.



Рис. 20.5. BeatBox с цветовыми атрибутами AppCompatActivity

Наконец, `colorAccent` назначается серый цвет. Цвет `colorAccent` должен контрастировать с атрибутом `colorPrimary`; он используется для формирования оттенка в некоторых виджетах (например, `EditText`).

Атрибут `colorAccent` на внешнем виде приложения BeatBox не отражается, потому что кнопки не поддерживают оттенки. Тем не менее мы все равно задаем `colorAccent`, потому что эти три цветовых атрибута удобнее рассматривать вместе.

Запустите приложение BeatBox и понаблюдайте за новыми цветами в действии. Ваше приложение должно выглядеть так, как показано на рис. 20.5.

Переопределение атрибутов темы

Теперь пора поглубже изучить вопрос и поближе познакомиться с атрибутами тем, которые вы можете переопределять. Учтите, что исследование тем — непростое дело. Почти не существует документации, в которой было бы указано, какие существуют атрибуты, какие из них вы можете переопределять и что делает тот или иной атрибут. Вам придется самостоятельно прокладывать путь в этих джунглях. Хорошо, что у вас есть надежный проводник (эта книга).

Начнем с изменения цвета фона BeatBox посредством модификации темы. Хотя теоретически вы можете открыть файл `res/layout/fragment_beat_box.xml` и вручную задать атрибут `android:background` для виджета `RecyclerView`, а затем повторить процесс со всеми остальными файлами макетов фрагментов и активностей. Конечно, это будет весьма неэффективно.

Тема всегда назначает цвет фона. Назначая другой цвет поверх этого, вы выполняете лишнюю работу. Кроме того, дублирование атрибута в приложении усложняет сопровождение кода.

Исследование тем

Вместо этого было бы правильнее переопределить атрибут цвета фона в теме. Чтобы узнать имя атрибута, взгляните, как атрибут задается в родительской теме: `Theme.AppCompat`.

Возникает резонный вопрос: как узнать, какой атрибут нужно переопределить, если я не знаю его имени? Никак. Вы будете читать имена атрибутов, и в какой-то момент у вас блеснет догадка: «А вот это похоже». Вы переопределите атрибут, запустите приложение и будете надеяться, что сделали разумный выбор.

В конечном итоге требуется найти самого первого предка вашей темы: ее пра-пра-пра... в общем, предка неизвестно какого уровня. Для этого вы будете переходить к родителю более высокого уровня, пока не найдете тему, не входящую в библиотеку `AppCompat`, — возможно, вообще не имеющую родителя.

Откройте файл `styles.xml` и щелкните на `Theme.AppCompat` с нажатой клавишей `Command` (или `Ctrl` в `Windows`). Посмотрим, как далеко уходит этот лабиринт.

(Если вам не удастся переходить по атрибутам тем прямо в `Android Studio` или вы предпочитаете делать это вне `Android Studio`, исходные файлы тем `Android` находятся в каталоге `ваш-каталог-SDK/platforms/android-21/data/res/values`.)

На момент написания книги при этом открывался очень большой файл, в котором выделялась следующая строка:

```
<style name="Theme.AppCompat" parent="Base.Theme.AppCompat" />
```

Тема `Theme.AppCompat` наследует атрибуты от `Base.Theme.AppCompat`. Интересно, что `Theme.AppCompat` не переопределяет никакие атрибуты, а только содержит ссылку на своего родителя.

Щелкните на `Base.Theme.AppCompat` с нажатой клавишей `Command`. Android Studio сообщит, что тема уточняется по ресурсам. Существует несколько разных версий этой темы в зависимости от используемой версии Android.

Выберите версию `values-v14/values.xml`; открывается определение `Base.Theme.AppCompat` (рис. 20.6).

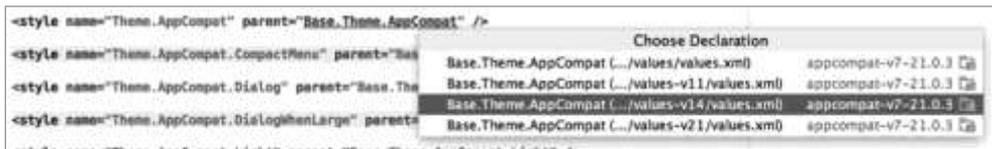


Рис. 20.6. Выбор родителя v14

(Мы выбрали версию `v14`, потому что `BeatBox` поддерживает API уровня 16 и выше. Если выбрать версию `v21`, вы могли бы столкнуться с новыми возможностями, добавленными в API уровня 21. За дополнительной информацией обращайтесь к упражнению в конце главы.)

```
<style name="Base.Theme.AppCompat" parent="Base.V14.Theme.AppCompat">
  <item name="android:actionModeCutDrawable">?actionModeCutDrawable</item>
  <item name="android:actionModeCopyDrawable">?actionModeCopyDrawable</item>
  <item name="android:actionModePasteDrawable">?actionModePasteDrawable</item>
  <item name="android:actionModeSelectAllDrawable">
    ?actionModeSelectAllDrawable</item>
  <item name="android:actionModeShareDrawable">?actionModeShareDrawable</item>
</style>
```

Тема содержит несколько атрибутов, но на первый взгляд нет ничего, что соответствовало бы цели: изменению цвета фона. Перейдите к `Base.V14.Theme.AppCompat`.

```
<style name="Base.V14.Theme.AppCompat" parent="Base.V11.Theme.AppCompat" />
```

`Base.V14.Theme.AppCompat` — еще одна тема, которая существует только ради имени и не переопределяет никакие атрибуты. Вернитесь к родительской теме: `Base.V11.Theme.AppCompat`.

```
<style name="Base.V11.Theme.AppCompat" parent="Base.V7.Theme.AppCompat" />
```

Еще одна пустая тема. Переходим к родителю.

```
<style name="Base.V7.Theme.AppCompat" parent="Platform.AppCompat">
  <item name="windowActionBar">true</item>
  <item name="windowActionBarOverlay">>false</item>
  ...
</style>
```

Постепенно приближаемся. `Base.V7.Theme.AppCompat` содержит много атрибутов, но и это не совсем то, что нужно. При выходе за пределы `AppCompat` вы найдете еще много разных атрибутов. Перейдите к `Platform.AppCompat`. Вы увидите, что и эта тема уточняется по ресурсам. Выберите версию `values-v11/values.xml`.

```
<style name="Platform.AppCompat" parent="android:Theme.Holo">
  <item name="android:windowNoTitle">true</item>
  <item name="android:windowActionBar">false</item>
  <item name="buttonBarStyle">?android:attr/buttonBarStyle</item>
  <item name="buttonBarButtonStyle">?android:attr/buttonBarButtonStyle</item>
  <item name="selectableItemBackground">?android:attr/
selectableItemBackground</item>
  ...
</style>
```

Наконец-то мы видим, что родителем темы `Platform.AppCompat` является `android:Theme.Holo`. Обратите внимание: ссылка на тему `Holo` не записывается в виде `Theme.Holo`. К ней также добавляется префикс пространства имен `android`.

Считайте, что библиотека `AppCompat` живет внутри вашего приложения. При построении проекта вы включаете библиотеку `AppCompat`, которая приносит с собой набор файлов с кодом Java и разметкой XML. Эти файлы практически ничем не отличаются от файлов, которые вы пишете самостоятельно. Если вы захотите сослаться на какой-либо компонент из библиотеки `AppCompat`, вы делаете это напрямую, используя запись `Theme.AppCompat`, потому что эти файлы существуют в вашем приложении.

Темы, существующие в ОС Android, такие как `Theme.Holo`, должны объявляться с пространством имен, указывающим на их местоположение. Библиотека `AppCompat` использует запись `android:Theme.Holo`, потому что тема `Holo` существует в ОС Android.

Перейдите к `android:Theme.Holo`.

```
<style name="Theme.Holo">
  <item name="colorForeground">@color/bright_foreground_holo_dark</item>
  <item name="colorForegroundInverse">...</item>
  <item name="colorBackground">@color/background_holo_dark</item>
  <item name="colorBackgroundCacheHint">...</item>
  <item name="disabledAlpha">0.5</item>
  <item name="backgroundDimAmount">0.6</item>
  ...
</style>
```

Наконец, мы пришли к месту назначения. Здесь представлены все атрибуты, которые можно переопределять в теме. Конечно, вы можете перейти к родителю `Theme.Holo: Theme`, но это не обязательно. Тема `Holo` переопределяет все атрибуты, которые вам понадобятся.

В самом начале объявляется `colorBackground`. По имени атрибута можно предположить, что он определяет цвет фона темы.

```
<style name="Theme.Holo">
  <item name="colorForeground">@color/bright_foreground_holo_dark</item>
  <item name="colorForegroundInverse">...</item>
  <item name="colorBackground">@color/background_holo_dark</item>
  ...
</style>
```

Этот атрибут должен переопределяться в приложении BeatBox. Вернитесь к файлу styles.xml и переопределите атрибут colorBackground.

Листинг 20.10. Настройка фона окна (res/values/styles.xml)

```
<style name="AppTheme" parent="Theme.AppCompat">
  <item name="colorPrimary">@color/red</item>
  <item name="colorPrimaryDark">@color/dark_red</item>
  <item name="colorAccent">@color/gray</item>

  <item name="android:colorBackground">@color/soothing_blue</item>
</style>
```

Обратите внимание на необходимость использования пространства имен android при переопределении атрибута, потому что colorBackground объявляется в ОС Android.

Запустите BeatBox, прокрутите список до конца RecyclerView и убедитесь в том, что в том месте, где фон не закрыт кнопками, он окрашен в синий цвет (рис. 20.7).



Рис. 20.7. BeatBox с окраской фона в теме

Действия, которые мы только что проделали для нахождения атрибута `color-Background`, приходится выполнять каждому разработчику Android при модификации темы приложения. Документация по этим атрибутам практически отсутствует. Чтобы узнать о доступных возможностях, обычно приходится обращаться к источникам.

Итак, мы переходили по следующим темам:

- ① Theme.AppCompat
- ① Base.Theme.AppCompat
- ① Base.V14.Theme.AppCompat
- ① Base.V11.Theme.AppCompat
- ① Base.V7.Theme.AppCompat
- ① Platform.AppCompat
- ① android:Theme.Holo

Мы переходили по иерархии тем до тех пор, пока не добрались до одной из тем ОС Android (за пределами библиотеки AppCompat). Когда вы лучше освоитесь с возможностями работы с темами, вероятно, вы сразу начнете переходить к соответствующей теме Android. Тем не менее полезно пройти по иерархии, чтобы понять логику происхождения темы.

Иерархия тем может измениться со временем, но задача перемещения по иерархии останется неизменной. Вы переходите по иерархии тем до тех пор, пока не будет найден атрибут, который требуется переопределить.

Изменение атрибутов кнопки

Ранее мы настраивали кнопки приложения BeatBox вручную, задавая атрибут `style` в файле `res/layout/list_item_sound.xml`. В более сложных приложениях, в которых кнопки распределены по многим фрагментам, решение с назначением атрибута `style` для каждой кнопки плохо масштабируется. В такой ситуации можно пойти дальше и определить в теме стиль для всех кнопок приложения.

Прежде чем добавлять в тему стиль кнопок, удалите атрибут `style` из файла `res/layout/list_item_sound.xml`.

Листинг 20.11. Долгой! Есть способ получше (`res/layout/list_item_sound.xml`)

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    style="@style/BeatBoxButton.Strong"
    android:id="@+id/button"
    android:layout_width="match_parent"
    android:layout_height="120dp"
    tools:text="Sound name"/>
```

Запустите приложение BeatBox и убедитесь в том, что кнопки вернулись к старому невыразительному виду.

Вернитесь к определению Theme.Holo и найдите группу атрибутов кнопок.

```
<style name="Theme.Holo">
  ...
  <!-- Стили кнопок -->
  <item name="buttonStyle">@style/Widget.Holo.Button</item>

  <item name="buttonStyleSmall">@style/Widget.Holo.Button.Small</item>
  <item name="buttonStyleInset">@style/Widget.Holo.Button.Inset</item>
  ...
</style>
```

Обратите внимание на атрибут с именем buttonStyle: он определяет стиль любой нормальной кнопки в приложении. Атрибуту buttonStyle вместо значения назначается ресурс стиля. При обновлении атрибута colorBackground передается значение: цвет. В нашем случае атрибут buttonStyle должен ссылаться на стиль. Перейдите к Widget.Holo.Button, чтобы просмотреть стиль кнопки.

```
<style name="Widget.Holo.Button" parent="Widget.Button">
  <item name="background">@drawable/btn_default_holo_dark</item>
  <item name="textAppearance">?attr/textAppearanceMedium</item>
  <item name="textColor">@color/primary_text_holo_dark</item>
  <item name="minHeight">48dip</item>
  <item name="minWidth">64dip</item>
</style>
```

Каждой кнопке Button, используемой в BeatBox, назначаются эти атрибуты.

Воспроизведите в BeatBox то, что происходит в теме Android. Измените родителя BeatBoxButton, чтобы атрибуты наследовались от существующего стиля кнопки. Также удалите стиль BeatBoxButton.Strong, который использовался ранее.

Листинг 20.12. Создание стиля кнопки (res/values/styles.xml)

```
<resources>

  <style name="AppTheme" parent="Theme.AppCompat">dark_blue
    <item name="colorPrimary">@color/red</item>
    <item name="colorPrimaryDark">@color/dark_red</item>
    <item name="colorAccent">@color/gray</item>

    <item name="android:colorBackground">@color/soothing_blue</item>
  </style>

  <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
    <item name="android:background">@color/dark_blue</item>
  </style>

  <del style name="BeatBoxButton.Strong">
    <del item name="android:textStyle">bold</del>
  </del>

</resources>
```

Мы указали родителя `android:style/Widget.Holo.Button`. Наша кнопка должна наследовать все свойства обычной кнопки, а затем избирательно изменять некоторые атрибуты.

Если родительская тема для `BeatBoxButton` не указана, то внешний вид кнопок резко ухудшается и кнопка вообще перестает напоминать кнопку. Многие привычные свойства, например выравнивание текста по центру кнопки, пропадают.

Итак, стиль `BeatBoxButton` полностью определен, и мы можем использовать его в приложении. Вернитесь к атрибуту `buttonStyle`, который мы обнаружили ранее в ходе исследования тем Android. Продублируйте этот атрибут в своей теме.

Листинг 20.13. Использование стиля `BeatBoxButton` (`res/values/styles.xml`)

```
<resources>

  <style name="AppTheme" parent="Theme.AppCompat">
    <item name="colorPrimary">@color/red</item>
    <item name="colorPrimaryDark">@color/dark_red</item>
    <item name="colorAccent">@color/gray</item>

    <item name="android:colorBackground">@color/soothing_blue</item>
    <item name="android:buttonStyle">@style/BeatBoxButton</item>
  </style>

  <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
    <item name="android:background">@color/dark_blue</item>
  </style>

</resources>
```

Мы переопределяем атрибут `buttonStyle` из тем Android и подставляем свой собственный стиль: `BeatBoxButton`.

Запустите приложение `BeatBox` и обратите внимание на то, что все кнопки окрасились в темно-синий цвет (рис. 20.8). Таким образом, мы изменили внешний вид всех обычных кнопок в `BeatBox` без прямой модификации файлов макетов. Атрибуты темы в Android — сила!

Возможно, вы заметили, что при нажатии внешний вид кнопок не изменяется. Дело в том, что мы не определили изменение стиля для нажатого состояния. В следующей главе проблема будет решена, и тогда кнопки проявят себя в полной мере.

Для любознательных: подробнее о наследовании стилей

Описание наследований стилей, приведенное ранее, не содержит полной информации. Возможно, вы заметили изменение в стиле наследования во время изучения иерархии тем. В темах `AppCompat` имя темы используется для обозначения наследования — до того момента, когда мы приходим к теме `Platform.AppCompat`.



Рис. 20.8. Приложение BeatBox с полностью определенными темами

```
<style name="Platform.AppCompat" parent="android:Theme.Holo">  
...  
</style>
```

Здесь вместо «наследного» стиля формирования имен происходит переключение на прямое определение родителя в атрибуте `parent`. Почему?

Указание родительской темы в имени темы работает только для тем, существующих в одном пакете. Таким образом, в темах ОС Android в большинстве случаев используется «наследный» стиль имен, и библиотека AppCompat действует так же. Но как только наследование пересекает границу AppCompat, используется атрибут `parent`.

Это правило желательно соблюдать и в ваших приложениях. Укажите родителя темы в имени темы, если вы наследуете от одной из своих собственных тем. Если же наследование осуществляется от стиля или темы ОС Android, используйте явное задание атрибута `parent`.

Для любознательных: обращение к атрибутам тем

После того как атрибуты будут объявлены в теме, вы сможете обращаться к ним из XML или из кода.

Для обращения к атрибуту темы из разметки XML используется запись, продемонстрированная для атрибута `divider` в главе 17. Для ссылки на конкретное значение в XML (например, цвет) используется синтаксис `@`. Запись `@color/gray` указывает на конкретный ресурс.

Для ссылок на ресурс в теме используется знак `?`.

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/button"
  android:layout_width="match_parent"
  android:layout_height="120dp"
  android:background="?attr/colorAccent"
  tools:text="Sound name"/>
```

Знак `?` указывает на использование ресурса, на который указывает атрибут `colorAccent` вашей темы. В данном случае это серый цвет, определенный в файле `colors.xml`.

Также можно использовать атрибуты темы в коде, хотя на этот раз запись получается не столь компактной.

```
Resources.Theme theme = getActivity().getTheme();
int[] attrsToFetch = { R.attr.colorAccent };
TypedArray a = theme.obtainStyledAttributes(R.style.AppTheme, attrsToFetch);
int accentColor = a.getInt(0, 0);
a.recycle();
```

В объекте `Theme` мы приказываем разрешить атрибут `R.attr.colorAccent`, определенный в `AppTheme: R.style.AppTheme`. Вызов возвращает массив `TypedArray`, содержащий данные. Из массива `TypedArray` извлекается значение типа `int`, которое в дальнейшем может использоваться, например, для изменения фона кнопки.

Панель инструментов и кнопки в `BeatBox` именно так поступают для определения своего стиля на основании атрибутов темы.

Упражнение. Базовая тема

При создании `BeatBoxButton` атрибуты наследовались от `android:style/Widget.Holo.Button`. Хотя наследование от темы `Holo` работает, оно не позволяет воспользоваться новейшей из доступных тем.

В `Android 5.0 (Lollipop)` появилась материальная тема, изменяющая различные свойства кнопок, включая размер шрифта. Попробуйте воспользоваться преимуществами нового оформления на устройстве с поддержкой материальной темы.

Создайте версию файла `styles.xml` с уточнением по ресурсам: `values-v21/styles.xml`. Затем создайте две версии стиля `BeatBoxButton`. Одна должна наследовать атрибуты от `Widget.Holo.Button`, а другая — от `Widget.Material.Button`.

21

Графические объекты

После назначения тем BeatBox пришло время сделать что-то с кнопками.

В текущей версии кнопки никак не реагируют на нажатия; это просто синие прямоугольники. В этой главе мы при помощи *графических объектов* (drawable) XML поднимем приложение BeatBox на новый уровень (рис. 21.1).



Рис. 21.1. Новая версия BeatBox

В Android графическим объектом называется все, что предназначено для рисовки на экране, будь то абстрактная фигура, класс, производный от `Drawable`, или растровое изображение. Нам уже знаком один вид графических объектов: `BitmapDrawable`, представляющий растровое изображение. В этой главе мы рассмотрим несколько других видов графических объектов: *списки состояний*, *геометрические фигуры* и *списки слоев*. Все три вида обычно определяются в файлах XML, поэтому мы объединим их в более широкую категорию *графических объектов XML*.

Унификация кнопок

Прежде чем браться за создание графических объектов XML, внесите изменения в файл `list_item_sound.xml`.

Листинг 21.1. Определение размеров (`res/layout/list_item_sound.xml`)

```
<Button xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/list_item_sound_button"
  android:layout_width="match_parent"
  android:layout_height="120dp"
  tools:text="Sound name"/>

<FrameLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_margin="8dp"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">

  <Button
    android:id="@+id/list_item_sound_button"
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_gravity="center"
    tools:text="Sound name"/>
</FrameLayout>
```

Каждой кнопке назначается ширина и высота 100 dp, чтобы последующее преобразование кнопок в круги не приводило к искажениям.

В `RecyclerView` независимо от размера экрана всегда отображаются три столбца. При наличии свободного места `RecyclerView` растянет эти столбцы по размерам экрана. В нашем приложении кнопки растягиваться не должны, поэтому они были заключены в виджет `FrameLayout`: он будет растягиваться, а кнопки — нет.

Запустите приложение `BeatBox`. Вы увидите, что все кнопки имеют одинаковые размеры и разделяются небольшими интервалами (рис. 21.2).



Рис. 21.2. Равномерное распределение кнопок

Геометрические фигуры

Придадим кнопкам круглую форму с помощью `ShapeDrawable`. Поскольку графические объекты XML не привязаны к конкретной плотности пикселей, обычно они размещаются в папке `drawable` по умолчанию (а не в одной из специализированных папок).

В окне инструментов Project создайте в папке `res/drawable` новый файл с именем `button_beat_box_normal.xml`. (Почему кнопка названа «normal», то есть «нормальной»? Потому что скоро появится другая, менее нормальная.)

Листинг 21.2. Создание графического объекта
(`res/drawable/button_beat_box_normal.xml`)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/dark_blue"/>

</shape>
```

Этот файл создает эллиптический графический объект, заполненный темно-синим цветом. Вы можете использовать элемент `shape` для создания других фигур: прямоугольников, линий, градиентов и т. д. За подробностями обращайтесь к документации по адресу <http://developer.android.com/guide/topics/resources/drawable-resource.html>.

Назначьте `button_beat_box_normal` в качестве фона кнопок.

Листинг 21.3. Изменение фона кнопок (`res/drawable/button_beat_box_normal.xml`)

```
<resources>
```

```
    <style name="AppTheme" parent="Theme.AppCompat">
```

```
        ...
```

```
    </style>
```

```
    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
```

```
        <item name="android:background">@color/dark_blue</item>
```

```
        <item name="android:background">@drawable/button_beat_box_normal</item>
```

```
    </style>
```

```
</resources>
```

Запустите приложение `BeatBox`. Все кнопки стали круглыми (рис. 21.3).



Рис. 21.3. Круглые кнопки

Нажмите кнопку. Вы услышите звук, но внешний вид кнопки не изменится. Было бы лучше, если бы нажатая кнопка отличалась внешним видом от ненажатой.

Списки состояний

Чтобы решить эту проблему, сначала определите новую геометрическую фигуру для кнопки в нажатом состоянии.

Создайте в папке `res/drawable` файл `button_beat_box_pressed.xml`. Нажатая кнопка будет выглядеть так же, как обычная, но ей будет назначен красный цвет фона.

Листинг 21.4. Определение графического объекта для нажатой кнопки (`res/drawable/button_beat_box_pressed.xml`)

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="oval">

    <solid
        android:color="@color/red"/>

</shape>
```

Новая версия должна использоваться тогда, когда пользователь нажимает кнопку. Для решения этой задачи используется список состояний.

Список состояний представляет собой графический объект, который указывает на другие графические объекты в зависимости от состояния некоторого селектора. Кнопка может находиться в нажатом и ненажатом состоянии. Список состояний будет определять один графический объект как фон для кнопки в нажатом состоянии и другой графический объект для ненажатых кнопок.

Определите список состояний в папке `drawable`.

Листинг 21.5. Создание списка состояний (`res/drawable/button_beat_box.xml`)

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/button_beat_box_pressed"
        android:state_pressed="true"/>
    <item android:drawable="@drawable/button_beat_box_normal" />
</selector>
```

Измените стиль кнопок, чтобы список состояний использовался в качестве фона.

Листинг 21.6. Назначение списка состояний (`res/values/styles.xml`)

```
<resources>
    <style name="AppTheme" parent="Theme.AppCompat">
        ...
    </style>

    <style name="BeatBoxButton" parent="android:style/Widget.Holo.Button">
        <item name="android:background">@drawable/button_beat_box_
            normal</item>
```

```
<item name="android:background">@drawable/button_beat_box</item>  
</style>
```

```
</resources>
```

Когда кнопка находится в нажатом состоянии, в качестве фона используется графический объект `button_beat_box_pressed`. В противном случае фон кнопки определяется `button_beat_box_normal`.

Запустите приложение `BeatBox` и нажмите кнопку. Фон кнопки изменяется (рис. 21.4). Впечатляет, не правда ли?



Рис. 21.4. BeatBox с кнопкой в нажатом состоянии

Списки состояний — удобный инструмент настройки элементов интерфейса. Также поддерживаются и другие состояния, включая состояние блокировки, наличия фокуса и активации. За подробностями обращайтесь к документации по адресу <http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList>.

Списки слоев

Приложение `BeatBox` хорошо смотрится: в нем используются круглые кнопки, которые визуально реагируют на нажатия. Но пришло время сделать что-то более интересное.

Списки слоев (layer list) позволяют объединить два графических объекта XML в один объект. Вооружившись этим инструментом, мы добавим темное кольцо вокруг кнопки в нажатом состоянии.

Листинг 21.7. Использование списка слоев
(res/drawable/button_beat_box_pressed.xml)

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
  <item>
    <shape
      android:shape="oval">
      <solid
        android:color="@color/red"/>
    </shape>
  </item>
  <item>
    <shape
      android:shape="oval">
      <stroke
        android:width="4dp"
        android:color="@color/dark_red"/>
    </shape>
  </item>
</layer-list>
```

Список слоев состоит из двух графических объектов. Первый объект — красный круг, как и перед изменением. Второй графический объект будет нарисован поверх первого. Второй графический объект представляет собой другой овал с толщиной линии 4dp; в результате создается кольцо темно-красного цвета. В список слоев можно включить более двух графических объектов, чтобы добиться еще более сложного результата.

Запустите приложение BeatBox и нажмите кнопку. Вы увидите, что в нажатом состоянии она выделяется кольцом (рис. 21.5). Еще лучше, чем прежде.

Добавление списка слоев завершает приложение BeatBox. Помните, как банально выглядел исходный интерфейс? Чтобы создать нечто куда более интересное, нам потребовалось совсем немного времени. Сделайте ваше приложение приятным для глаз — и ваши усилия окупятся его популярностью.

Для любознательных: для чего нужны графические объекты XML?

Кнопки всегда должны быть в нажатом состоянии, поэтому список состояний является критическим компонентом любого приложения Android. Но как насчет геометрических фигур и списков слоев? Стоит ли использовать их?



Рис. 21.5. Готовое приложение BeatBox

Графические объекты XML отличаются гибкостью. Их можно использовать для многих целей и легко обновлять в будущем. Комбинации списков слоев и геометрических фигур позволяют создавать сложные фоны без графического редактора. Если вы решите изменить цветовую схему в BeatBox, обновить цвета в графическом объекте XML будет несложно.

В этой главе графические объекты XML определялись в каталоге `drawable` без квалификаторов, уточняющих плотность пикселей. Это объясняется тем, что графические объекты XML не зависят от плотности. Стандартные растровые изображения обычно приходится создавать для нескольких вариантов плотности, чтобы изображение выглядело четко на большинстве устройств. Графические объекты XML достаточно определить только один раз, и они будут четко выглядеть при любой плотности.

Для любознательных: 9-зонные изображения

Иногда (а может быть, очень часто) вы будете использовать обычные графические изображения в качестве фона кнопок. Но что произойдет с этими изображениями, если кнопка может отображаться в разных вариантах размеров?

Если ширина кнопки превышает ширину фонового изображения, то изображение просто растянется, верно? Но всегда ли результат будет хорошо выглядеть?

Равномерное растяжение фонового изображения не всегда приводит к хорошему результату. Иногда требуется лучше управлять тем, как будет растягиваться изображение.

В этом разделе приложение BeatBox будет переведено на использование *9-зонного изображения* в качестве фона кнопок. Для начала изменим файл `list_item_sound.xml` для того, чтобы размеры кнопки могли изменяться по размерам свободного пространства.

Листинг 21.8. Включение растяжения кнопок (`res/layout/list_item_sound.xml`)

```
<FrameLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_margin="8dp"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">

  <Button
    android:id="@+id/list_item_sound_button"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center"
    tools:text="Sound name"/>
</FrameLayout>
```

Теперь кнопки занимают все доступное место, оставляя поля шириной 8dp. Изображение на рис. 21.6 будет использоваться в качестве нового фона кнопок BeatBox.



Рис. 21.6. Новое фоновое изображение кнопки (`res/drawable-xxhdpi/ic_button_beat_box_default.png`)

В решениях этой главы (см. раздел «Добавление значка» главы 2) вы найдете это изображение вместе с версией для нажатого состояния в папке `xxhdpi`. Скопируйте оба изображения в папку `drawable-xxhdpi` вашего проекта и внесите изменения в файл `button_beat_box.xml`, чтобы назначить их фонами кнопок.

Листинг 21.9. Назначение новых фоновых изображений кнопок (res/drawable/button_beat_box.xml)

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/ic_button_beat_box_pressed"
        android:state_pressed="true"/>
  <item android:drawable="@drawable/ic_button_beat_box_default" />
</selector>
```

Запустите BeatBox; вы увидите, что кнопки выводятся с новым фоном (рис. 21.7).



Рис. 21.7. BeatBox

И что получилось?.. Да ничего хорошего.

Почему результат плохо выглядит? Android равномерно растягивает изображение `ic_beat_box_button.png`, включая загнутый уголок и закругления. Было бы лучше, если бы вы могли явно указать, какие части изображения можно растягивать, а какие должны остаться в исходном виде. На помощь приходят *9-зонные* изображения.

Файл *9-зонного* изображения специально форматируется так, чтобы система Android знала, какие части изображения можно или нельзя форматировать. При правильной реализации это гарантирует, что края и углы фона будут соответствовать изображению в том виде, в каком оно было создано.

Почему изображения называются «9-зонными»? Такие изображения разбиваются сеткой 3×3 — такая сетка состоит из 9 элементов, или *зон* (patches). Углы сетки не масштабируются, стороны масштабируются только в одном направлении, а центральная зона масштабируется в обоих направлениях (рис. 21.8).

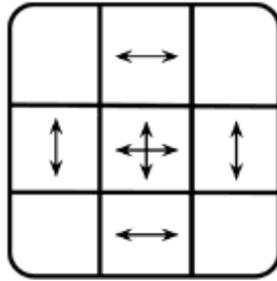


Рис. 21.8. Как работает 9-зонное изображение

9-зонное изображение во всех отношениях напоминает обычный файл png, за исключением двух аспектов: имя файла завершается суффиксом `.9.png` и изображение имеет дополнительную рамку толщиной в один пиксел. Рамка задает местонахождение центрального квадрата. Черными пикселями рамки обозначается центр, а прозрачными — края.

9-зонное изображение можно создать в любом графическом редакторе, но проще воспользоваться программой `draw9patch`, включенной в поставку Android SDK, или средствами Android Studio. Впрочем, на момент написания книги редактор 9-зонных изображений Android Studio был довольно капризным. Если вам понадобится программа `draw9patch`, она находится в каталоге `tools` установки SDK.

Сначала преобразуйте два новых фоновых изображения в 9-зонный формат: щелкните правой кнопкой мыши на файле `ic_button_beat_box_default.png` в окне инструментов Project и выберите команду Refactor ▶ Rename.... Переименуйте файл в `ic_button_beat_box_default.9.png`. Затем повторите процесс и переименуйте файл с «нажатой» версией изображения в `ic_button_beat_box_pressed.9.png`.

Сделайте двойной щелчок на стандартном изображении в окне инструментов Project, чтобы открыть его во встроенном 9-зонном редакторе Android Studio (рис. 21.9). (Если Android Studio не открывает редактор, попробуйте закрыть файл и свернуть папку `drawable` в окне инструментов Project, после чего откройте изображение заново.)

В 9-зонном редакторе заполните черные пикселы на верхней и левой границе, чтобы обозначить растягиваемые области изображения, как показано на рис. 21.9.

Этими двумя линиями вы сообщаете Android, что при изменении размеров правая верхняя часть изображения и углы не должны растягиваться. Повторите процедуру с «нажатой» версией изображения.

Левый и верхний участки границы отмечают растягиваемую область изображения. А как насчет правого и нижнего участка? Они отмечают необязательную

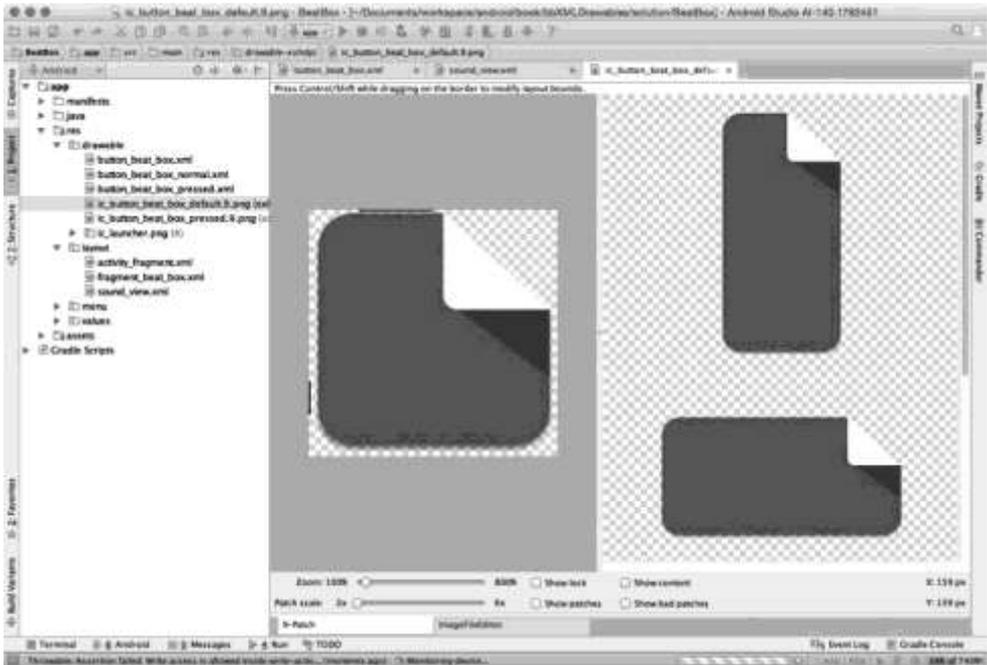


Рис. 21.9. Создание 9-зонного изображения

область прорисовки — область, в которой должно выводиться некое содержимое (обычно текст). Если область прорисовки не задана, по умолчанию считается, что она совпадает с растягиваемой областью. В нашем случае это именно так (текст кнопки должен выводиться в растягиваемой области), поэтому область прорисовки не указывается.

Запустите приложение BeatBox и посмотрите, как работает новое 9-зонное изображение (рис. 21.10).

Попробуйте повернуть устройство в альбомную ориентацию. Изображения растягиваются еще сильнее, но фон кнопки все равно выглядит хорошо.

Для любознательных: Miramar

Квалификаторы ресурсов и графические объекты удобны. Когда вам потребуется добавить графику в приложение, вы генерируете изображение с несколькими разными размерами и раскладываете их по папкам с квалификаторами: `drawable-mdpi`, `drawable-hdpi` и т. д. Затем вы обращаетесь к изображению по имени, а Android выбирает версию с нужной плотностью в зависимости от текущего устройства.

Однако у этой системы есть свой недостаток. Файл APK, публикуемый в Google Play Store, содержит все изображения из каталогов `drawable` для всех вариантов плотности, добавленные в проект, — при том что многие из них использоваться



Рис. 21.10. Новая улучшенная версия

не будут. Конечно, это приводит к неэффективному увеличению размера приложения.

Для решения этой проблемы можно сгенерировать разные APK для всех вариантов плотности пикселей: APK с mdpi-версией приложения, APK с hdpi-версией приложения и т. д. (За дополнительной информацией о построении разных версий APK обращайтесь к документации: <http://tools.android.com/tech-docs/new-build-system/user-guide/apk-splits>.)

Впрочем, у этого правила есть одно исключение: вы должны поддерживать значки приложения для лаунчера во всех вариантах плотности.

В Android *лаунчер* (launcher) представляет собой домашний экран со значками приложений (эта тема подробнее рассматривается в главе 22); он открывается при нажатии кнопки Home.

Некоторые новые лаунчеры отображают значки приложений в большем размере, чем принято. Чтобы увеличенный значок нормально смотрелся, такие лаунчеры должны взять значок из версии со следующей плотностью. Например, на hdpi-устройствах для представления приложения в лаунчере будет использоваться значок xhdpi. Но если xhdpi-версия исключена из APK, лаунчеру придется вернуться к версии с более низким разрешением. Масштабированные значки с низким разрешением кажутся размытыми, а мы хотим, чтобы значок приложения выглядел четко и аккуратно.

В Android для решения этой проблемы используется каталог `mipmap`. На момент написания книги новые проекты в Android Studio настроены на использование `mipmap`-ресурсов для значка лаунчера (рис. 21.11).



Рис. 21.11. Значки `mipmap`

При включении разбивки APK ресурсы `mipmap` не удаляются из APK. В остальном такие ресурсы не отличаются от графических объектов.

Итак, мы рекомендуем просто разместить значок лаунчера в разных каталогах `mipmap`. Всем остальным изображениям место в каталогах `drawable`.