

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

ANDROID



Урок №5

Ресурсы приложения Android

Содержание

1.	Ресурсы приложения. Типы ресурсов Android-приложения	4
1.1.	Color Resources. Ресурсы констант для цветовых значений	7
1.2.	Color State List Resources. Ресурсы наборов цветовых значений для разных состояний виджета	9
1.3.	Drawable Resources. Ресурсы графических изображений	24
1.4.	String Resources. Строковые ресурсы приложения. Локализация приложений.....	66
1.5.	Style Resources. Ресурсы стилей для внешнего вида виджетов и приложения	84
1.6.	MipMap ресурсы.....	109
1.7.	Dimesion Resources. Ресурсы для хранения величин размеров	118

1.8. Каталоги ресурсов raw и assets	120
2. Применение виджета Toolbar вместо ActionBar	124
3. Анимация.	
Создание анимации в ресурсах приложения.	
Применение анимации.	141
3.1. Property Animation. Анимация путем изменения свойств виджета. Примеры	142
3.2. Value Animation. Анимация путем изменения значения. Примеры	164
3.3. Интерполяторы.	184
3.4. View Animation. Примеры	188
3.5. Frame Animation. Анимация путем последовательной смены изображений через указанные промежутки времени. Примеры.	213
4. Практический пример приложения с использованием анимации.	226
4.1. Вводное описание общих частей практического примера	230
4.2. Использование Value Animation	246
4.3. Использование View Animation	260
4.4. Использование Property Animation	274
4.5. Завершающее описание общих частей рассматриваемого примера.	284
5. Домашнее задание	291

1. Ресурсы приложения. Типы ресурсов Android-приложения

Android приложение содержит в себе не только исполняемый код — оно имеет достаточно сложную структуру, в которой кроме исполняемого кода содержатся данные, представленные в двоичном виде.

Эти данные предназначены для использования в приложении. Это могут быть изображения, которые отображаются в приложении в виде пиктограмм или иконок на виджетах. Это могут быть строковые сообщения на разных языках, которые выводятся в приложении. Это могут быть файлы, например в формате pdf для отображения справочной информации, или файлы в формате mp3 для звукового сопровождения работы приложения. Можно и дальше продолжить приводить примеры таких данных.

Суть в том, что любое сложное приложение не может обойтись без этих данных, так как эти данные делают приложение более красивым, функциональным, запоминающимся, привлекательным для пользователя.

В большинстве случаев ресурсы Android приложения представлены в виде xml файлов. Ресурсы делятся на типы и для каждого типа предназначен свой каталог в проекте приложения.

Типы ресурсов Android-приложения:

- **Animation Resources** (*Ресурсы анимации*). Существует две разновидности ресурсов анимации: Tween анимация (ресурсы располагаются в каталоге /res/anim и доступны через класс **R.anim**) которая осуществляет изменение свойств виджетов или других значений в течении заданного времени по заданному алгоритму интерполяции, и Frame анимация (ресурсы располагаются в каталоге /res/drawable и доступны через класс **R.drawable**) которая осуществляет анимацию посредством последовательного изменения множества изображений («кадров») с равным промежутком времени.
- **Color Resources** (*Ресурсы констант цветов*). Располагаются в каталоге /res/values в файле colors.xml и доступны через класс **R.color**. Этот тип ресурсов содержит константы значений для цветов, которые используются в приложении. Использование этого типа ресурсов делает удобным процесс изменения значений цвета для случаев, когда одно и то же значение цвета используется многократно в разных частях программы.
- **Color State List Resources** (*Ресурсы списка цветов для разных состояний виджетов*). Располагаются в каталоге /res/color и доступны через класс **R.color**. Вы знаете, что многие виджеты имеют разный вид в зависимости от своего состояния. Например, кнопка **android.widget.Button** может выглядеть по разному в зависимости от того, находится ли она в фокусе или на нее сейчас

нажимает пользователь или она неактивна. Вот для таких разных состояний виджетов и предназначен данный тип ресурсов — с его помощью можно изменять внешний вид виджетов.

- **Drawable Resources** (*Графические, «отрисовываемые» ресурсы*). Располагаются в каталоге /res/drawable и доступны через класс **R.drawable**. Этот тип ресурсов предназначен для графических файлов растровых изображений (bmp, png, jpg, gif и т. д.) и для xml файлов, содержащих параметры для рисуемых фигур в виде элементов xml.
- **Layout Resources** (*Ресурсы макетов раскладок виджетов*). Располагаются в каталоге /res/layout и доступны через класс **R.layout**. Этот тип ресурсов содержит xml файлы с макетами раскладок внешнего вида Активностей, Диалоговых окон, элементов списков и так далее. Вам уже знаком этот тип ресурсов по предыдущим урокам, поэтому в данном уроке он рассматриваться не будет.
- **Menu Resources** (*Ресурсы меню приложения*). Располагаются в каталоге /res/menu и доступны через класс **R.menu**. Этот тип ресурсов содержит xml файлы содержащие информацию о пунктах меню и контекстных меню приложения. Вам уже знаком этот тип ресурсов по предыдущим урокам. В данном уроке этот тип ресурсов рассматриваться не будет, но будет использоваться в разделе, посвященном Панели инструментов **ToolBar**.
- **String Resources** (*Строковые ресурсы*). Располагаются в каталоге /res/values и доступны через классы **R.string** и **R.array**. Ресурсы предназначены для хранения значений

строковых констант для строк, которые используются в приложении. Использование этого типа ресурсов делает удобным процесс локализации приложения.

- **Style** (*Ресурсы стилей*). Располагаются в каталоге /res/values и доступны через класс **R.style**. С помощью этого типа ресурсов можно создавать наборы визуальных свойств для виджетов (таких как **android:textColor**, **android:background**, **android:textStyle** и так далее) чтобы применять одни и те же наборы для разных виджетов приложения.
- **More Resource Types** (Другие типы ресурсов). Располагаются в каталоге /res/values и доступны через классы **R.bool**, **R.integer**, **R.dimen** и так далее. Этот тип ресурсов предназначен для хранения всевозможных значений, таких как числовые константы, значения отступов (margin, padding) и так далее.

1.1. Color Resources.

Ресурсы констант для цветовых значений

Color Resources представляют собой ресурсы, содержащие константы для цветовых значений. Эти константы удобно применять как в xml файлах, так и в java файлах проекта.

Главная информация о ресурсах данного типа:

- Расположение файла с ресурсами *Color Resources*: /res/values/colors.xml.
- В файле /res/values/colors.xml объявляются множество констант, то есть один файл предназначен для хранения всех цветовых констант приложения.

- Для ссылки из программного кода Java к ресурсам данного типа используется обращение: **R.color.цветовая_константа**.
- Для ссылки из xml к ресурсам данного типа используется обращение: **@color/цветовая_константа**.

В Листинге 1.1 приведен пример файла `/res/values/colors.xml`.

Листинг 1.1. Пример файла ресурсов `/res/values/colors.xml`, содержащего константы для значений цветов

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#000000</color>
    <color name="red">#FF0000</color>
    <color name="green">#00FF00</color>
    <color name="colorPrimary">#9C27B0</color>
    <color name="colorPrimaryDark">#6A1B9A</color>
    <color name="colorAccent">#FF4081</color>
</resources>
```

Описание элементов xml из Листинга 1.2:

- **<resources>** — обязательный элемент. Является корневым элементом для xml документа. Содержит один или более элементов **<color>**.
- **<color>** — описывает цветовую константу. При помощи атрибута **name** задается имя цветовой константы, а значением элемента **<color>** является значение цвета в шестнадцатеричном формате RGB или ARGB которое начинается с символа `#`. Возможные форматы указания значения цвета: `#RGB`, `#ARGB`, `#RRGGBB`, `#AARRGGBB`.

Применение цветовых констант, объявленных в ресурсах /res/values/color.xml приведено в Листинге 1.2 для файлов xml и в Листинге 1.3 для файлов Java. Применение цветовых констант в этих Листингов выделено жирным текстом.

Листинг 1.2. Применение цветовых констант из файла ресурсов /res/values/colors.xml в xml файлах проекта

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/edt1"  
    android:textColor="@color/green"  
/>
```

Листинг 1.3. Применение цветовых констант из файла ресурсов /res/values/colors.xml в Java файлах проекта

```
Button btn = (Button) this.findViewById(R.id.btn1);  
btn.setTextColor(this.getResources().getColor(  
    R.color.green, this.getTheme()));
```

Примечание: пример из Листинга 1.3 требует уровня API 23 и выше.

1.2. Color State List Resources. Ресурсы наборов цветовых значений для разных состояний виджета

Color State List Resources представляют собой ресурсы, определяющие набор цветов для разных состояний виджетов. Наборы цветов представляются в виде списков.

Главная информация о ресурсах данного типа:

- Расположение файлов с ресурсами *Color State List Resources*: /res/color/имя_файла.xml.
- Имя файла ресурсов впоследствии используется как идентификатор (resource ID) ресурса.
- Данный тип ресурсов компилируется в объекты класса [android.content.res.ColorStateList](#).
- Для ссылки из программного кода Java к ресурсам данного типа используется обращение: R.color.filename.
- Для ссылки из xml к ресурсам данного типа используется обращение: @[package:]color/filename.

Синтаксис содержимого файлов для ресурсов данного типа приведен в Листинге 1.4.

Листинг 1.4. Синтаксис файлов xml для ресурсов Color State List Resources

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
    "http://schemas.android.com/apk/res/android" >
    <item
        android:color="hex_color"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_window_focused=
            ["true" | "false"] />
    </item>
</selector>
```

Описание элементов xml из Листинга 1.4:

- **<selector>** — обязательный элемент. Является корневым элементом для xml документа. Содержит один или более элементов **<item>**.

Атрибуты элемента **<selector>**:

- **xmlns:android** — обязательный атрибут. Определяет пространство имен xml документа. Атрибут должен содержать значение: <http://schemas.android.com/apk/res/android>.
- **<item>** — должен быть дочерним элементом для элемента **<selector>**. Определяет цвет для использования в определенных состояниях виджетов. Атрибуты элемента **<item>**:
 - **android:color** — обязательный атрибут. Определяет цвет. Цвет задается в шестнадцатеричном представлении в формате RGB и, при необходимости, можно указать еще и альфа-канал. Величина цвета всегда начинается с символа # и затем следуют значения для alpha-red-green-blue составляющих в таких возможных форматах: #RGB, #ARGB, #RRGGBB, #AARRGGBB.
 - **android:state_pressed** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован, когда пользователь нажал на виджет (например, пользователь нажал на кнопку **android.widget.Button**). Содержит *false*, если этот элемент (**<item>**) должен быть использован по умолчанию (для не нажатого состояния).

- **android:state_focused** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован, когда виджет получил фокус ввода (например, кнопка **android.widget.Button** выделяется при помощи трекбола/trackball). Содержит *false*, если этот элемент должен быть использован для состояния по умолчанию (т. е. при отсутствии фокуса ввода).
- **android:state_selected** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован, когда виджет выбран (например, вкладка открыта). Содержит *false*, если этот элемент должен быть использован, когда виджет не выбран.
- **android:state_checkable** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован, когда виджет может быть отмечен галочкой. Содержит *false*, если этот элемент должен быть использован, когда виджет не может быть отмечен галочкой. Этот атрибут используется только тогда, когда виджет может переходить из состояния возможности выбора галочкой в состояние невозможности выбора галочкой и наоборот.
- **android:state_checked** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован, когда виджет отмечен галочкой. Содержит *false*, если этот элемент должен быть использован, когда виджет не отмечен галочкой.
- **android:state_enabled** — тип Boolean. Содержит *true*, если этот элемент (**<item>**) должен быть использован,

когда виджет «разрешен» (*enabled*), т. е. может быть использован (например, виджет способен принимать события прикосновения или клика). Содержит *false*, если этот элемент должен быть использован, когда виджет «не разрешен», т. е. не может быть использован.

- **android:state_window_focused** — тип Boolean. Содержит *true*, если этот элемент (*<item>*) должен быть использован, когда окно приложения имеет фокус ввода, то есть находится на переднем плане. Содержит *false*, если этот элемент должен быть использован, когда окно приложения не имеет фокуса. Например, когда над окном приложения находится Диалоговое окно.

Теперь рассмотрим применение *Color State List Resources* на примере. Для примера создан модуль *app* в проекте с файлами исходного кода, которые прилагаются к данному уроку.

В примере создадим несколько файлов *Color State List Resources* для разных виджетов: кнопок *android.widget.Button*, текстовых полей *android.widget.EditText* и флажков *android.widget.CheckBox*.

Перед тем как мы приступим к созданию файлов ресурсов для *Color State List Resources*, нам необходимо создать каталог */res/color* в модуле нашего проекта. Для этого необходимо кликнуть правой кнопкой мыши название модуля и в появившемся контекстном меню выбрать *New/File Resource Directory* (см. Рис. 1.1).

Урок №5

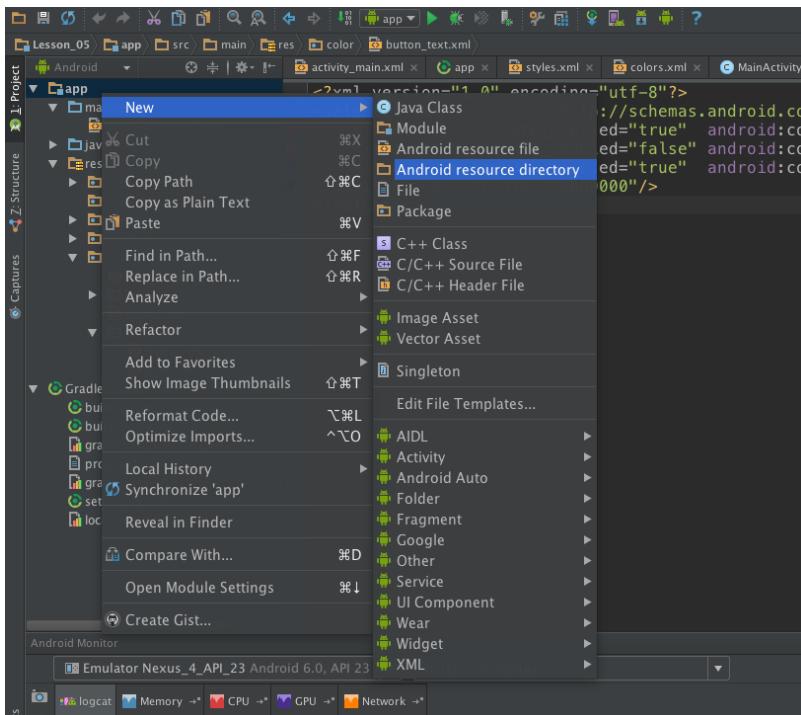


Рис. 1.1. Добавление каталога ресурсов в модуль проекта

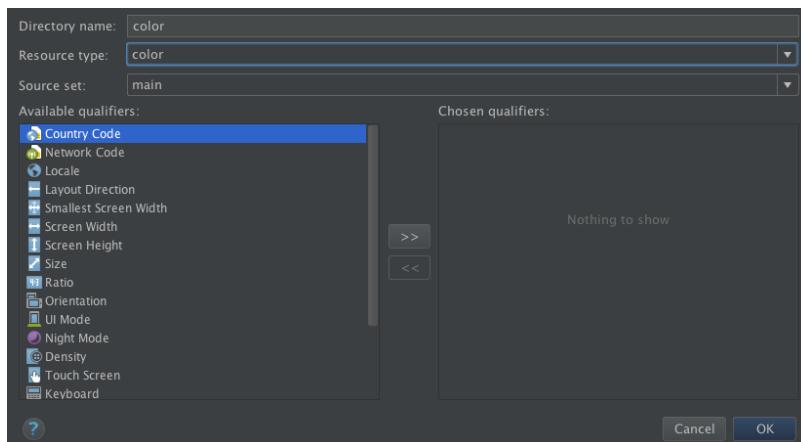


Рис. 1.2. Диалоговое окно выбора типа каталога ресурсов

В появившемся диалоговом окне необходимо выбрать в поле *Resource Type* значение *color* и указать название создаваемого каталога *color* (см. Рис. 1.2). Готово. Каталог создан.

Теперь перейдем к добавлению файлов в этот каталог. Создадим файл *Color State List Resources*, в котором будут определены цвета для текста кнопки **android.widget.Button**, в зависимости от состояния кнопки. Для добавления файла необходимо кликнуть правой кнопкой мыши по названию папки *color* и в появившемся контекстном меню выбрать *New/Color resource file* (см. Рис. 1.3).

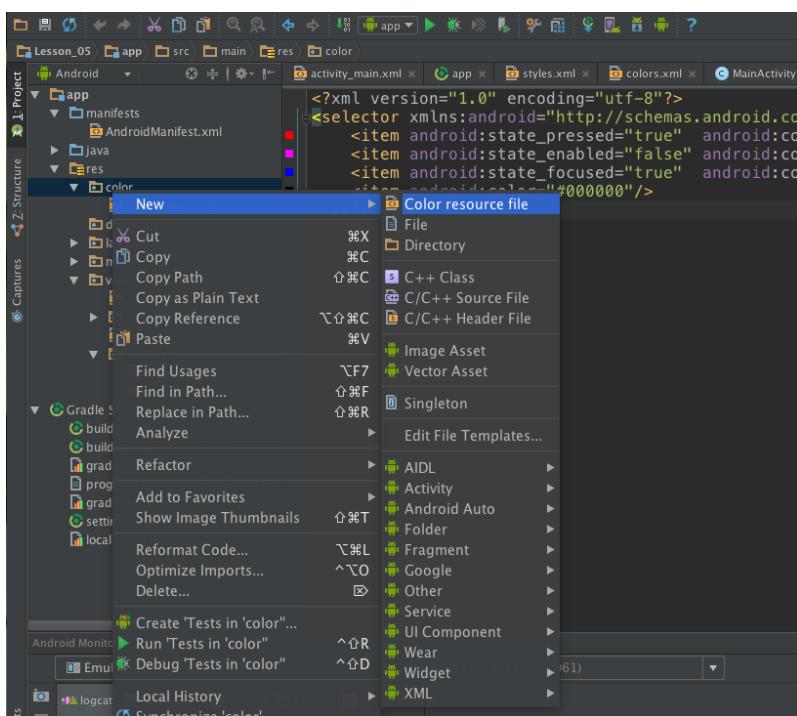


Рис. 1.3. Добавление файла ресурсов Color State List Resource в каталог /res/color модуля проекта

В появившемся диалоговом окне необходимо ввести имя файла (см. Рис. 1.4).

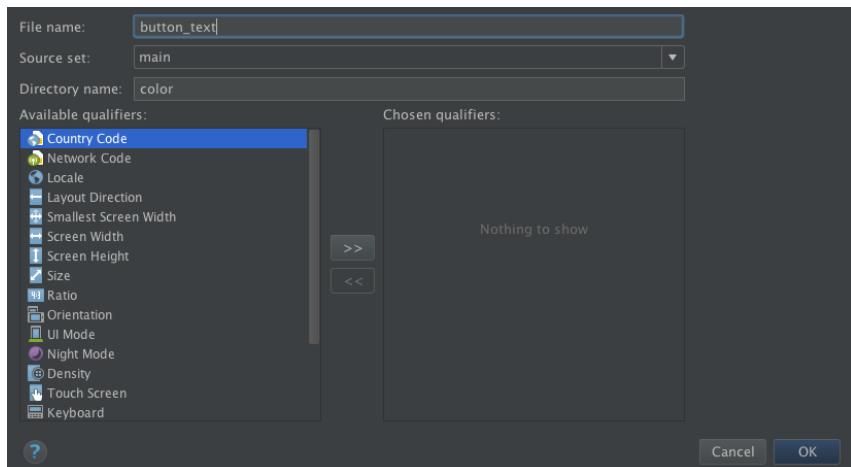


Рис. 1.4. Назначение файлу ресурсов названия

Назовем создаваемый файл button_text.xml. Содержимое файла /res/color/button_text.xml приведено в Листинге 1.5.

Листинг 1.5. Файл ресурсов Color State List Resources для текста кнопки android.widget.Button

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
    "http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:color="#FF0000" /><!-- pressed -->
    <item android:state_enabled="false"
        android:color="#FF00FF" /><!-- disabled -->
    <item android:color="#000000"/>
        <!-- default -->
</selector>
```

Как видно из Листинга 1.5, в списке состояний цвета для кнопки определены цвета для следующих состояний: красный цвет текста, когда кнопка нажата (`android:state_pressed="true"`), фиолетовый цвет текста, когда кнопка запрещена (`android:state_enabled="false"`) и черный цвет текста — для состояния по умолчанию.

Добавим в модуль проекта еще два файла с ресурсами *Color State List Resources*: один файл — для состояний виджета `android.widget.EditText` и один файл — для состояний виджета `android.widget.CheckBox`. Файлы назовем соответственно `/res/color/edittext_text.xml` (Листинг 1.6) и `/res/color/checkbox_text.xml` (Листинг 1.7).

Листинг 1.6. Файл ресурсов Color State List Resource для текста виджета `android.widget.EditText`

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
    "http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:color="#FF0000" /> <!-- pressed -->
    <item android:state_enabled="false"
        android:color="#FF00FF" /> <!-- disabled -->
    <item android:state_focused="true"
        android:color="#00695C" /> <!-- focused -->
    <item android:state_focused="false"
        android:color="#EF6C00" /> <!-- NOT focused -->
</selector>
```

Как видно из Листинга 1.6, в списке состояний цвета для виджета `android.widget.EditText` (Текстовое поле) определены цвета для следующих состояний: красный цвет текста — для состояния, когда пользователь

нажимает (прикасается) на текстовое поле (`android:state_pressed="true"`), фиолетовый цвет текста, когда текстовое поле запрещено (`android:state_enabled="false"`), зеленый, когда текстовое поле `android.widget.EditText` имеет фокус ввода (`android:state_focused="true"`) и оранжевый цвет, когда текстовое поле `android.widget.EditText` не имеет фокуса ввода (`android:state_focused="false"`).

Листинг 1.7. Файл ресурсов Color State List Resources для текста виджета `android.widget.CheckBox`

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android=
    "http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:color="#FF0000" /><!-- pressed -->
    <item android:state_enabled="false"
        android:color="#FF00FF" /><!-- disabled -->
    <item android:state_checked="true"
        android:color="#0000FF" /><!-- checked -->
    <item android:color="#000000"/>
    <!-- default -->
</selector>
```

Содержимое файла `/res/color/checkbox_text.xml` (Листинг 1.7) для состояний виджета похоже на предыдущие два файла ресурсов *Color State List Resource* только для виджета `android.widget.CheckBox` добавлено состояние `android:state_checked`. Как видно из Листинга 1.7, в списке состояний цвета для виджета `android.widget.CheckBox` (Флажок, чекбокс) определены цвета для следующих состояний: красный цвет текста — для состояния, когда пользователь нажимает (прикасается) на чекбокс

(`android:state_pressed="true"`), фиолетовый цвет текста, когда чекбокс запрещен (`android:state_enabled="false"`), синий цвет текста — для состояния, когда чекбокс отмечен галочкой (`android:state_checked="true"`).

Внешний вид работы примера изображен на Рис. 1.5 и Рис. 1.6. Для демонстрации запрещенного (`disabled`) состояния виджетов предназначена кнопка с идентификатором `R.id.btn2` и надписью *Disable all widgets* («Запретить все виджеты»). При нажатии на эту кнопку, рассматриваемые в примере виджеты (Кнопка `R.id.btn1`, Текстовые поля `R.id.edt1` и `R.id.edt2`, Флажки `R.id.cb1` и `R.id.cb2`) переводятся в состояние *disabled* (с помощью вызова метода `виджет.setEnabled(true);`) и надпись на кнопке `R.id.btn2` меняется на *Enable all widgets* («Разрешить все виджеты»). Следующее нажатие на кнопку `R.id.btn2` приведет к переводу рассматриваемых виджетов в состояние *enabled* (с помощью вызова метода `виджет.setEnabled(true);`). И так далее. Для демонстрации остальных состояний («нажат», «отмечен», «в фокусе») виджетов дополнительной функциональности не требуется. В Листинге 1.8 приведен xml код верстки макета Активности для рассматриваемого примера.

Листинг 1.8. Xml верстка макета Активности для рассматриваемого примера использования ресурсов Color State List Resources

```
<?xml version="1.0" encoding="utf-8"?>
<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:id="@+id/mainLL"
        android:orientation="vertical"

    tools:context="itstep.com.myapp.MainActivity">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:text="Button"
        android:textAllCaps="false"
        android:id="@+id	btn1"
        android:textColor="@color/button_text"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/edit1"
        android:text="Text Field №1"
        android:textColor="@color/edittext_text"
    />

    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/edit2"
        android:text="Text Field №2"
        android:textColor="@color/edittext_text"
    />

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Checkbox №1"
```

```
    android:id="@+id/cb1"
    android:textColor="@color/checkbox_text"
  />

<CheckBox
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Checkbox №2"
  android:id="@+id/cb2"
  android:textColor="@color/checkbox_text"
/>

<Button
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:layout_gravity="center_horizontal"
  android:text="Disable all widgets"
  android:textAllCaps="false"
  android:onClick="btnClick"
  android:id="@+id(btn2"
/>

</LinearLayout>
```

В Листинге 1.8 жирным шрифтом выделен код, назначающий файлы ресурсов *Color State List Resource* в качестве цвета текста для виджетов: файл /res/color/button_text.xml для виджета **R.id.btn1**, файл /res/color/edittext_text.xml для виджетов **R.id.edt1** и **R.id.edt2**, файл /res/color/checkbox_text.xml для виджетов **R.id.cb1** и **R.id.cb2**.

На Рис. 1.5 изображена поэтапная работа приложения из рассматриваемого в данном разделе примера, в виде снимков экрана устройства.

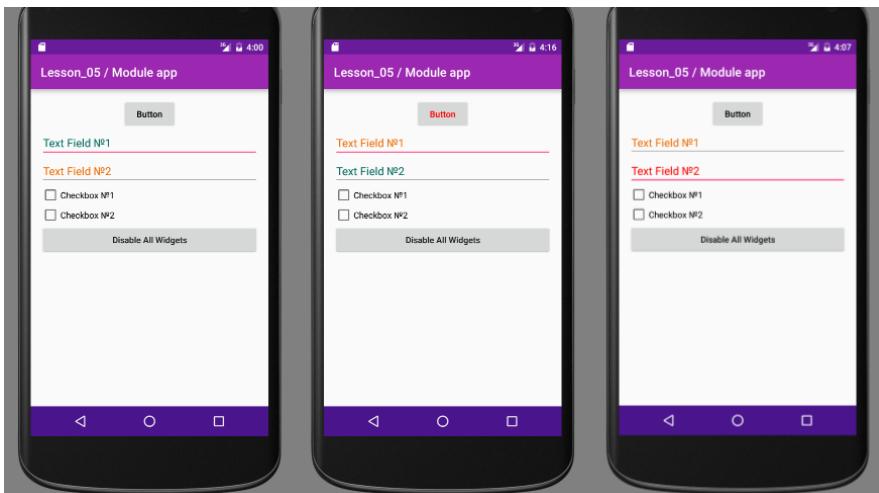


Рис. 1.5. Внешний вид работы примера из Листингов 1.5–1.8

На левом снимке Рис. 1.5 изображен внешний вид приложения со следующими состояниями виджетов: кнопка **R.id.btn1** и флагки **R.id.cb1**, **R.id.cb2** находятся в состоянии «по умолчанию» — цвет текста черный, текстовое поле **R.id.edt1** (с надписью *Text Field №1*) имеет фокус ввода и поэтому цвет текста зеленый, текстовое поле **R.id.edt2** (с надписью *Text Field №2*) не имеет фокуса ввода и поэтому цвет текста оранжевый (см. Листинг 1.6).

На центральном снимке рис. 1.5 кнопка **R.id.btn1** находится в нажатом состоянии и фокус ввода имеет текстовое поле **R.id.edt2**. Эти состояния обозначены разными цветами текста: цвет кнопки **R.id.btn1** — красный (см. Листинг 1.5), а цвет текстового поля **R.id.edt2** — зеленый.

На правом снимке изображения Рис. 1.5 пользователь нажимает на текстовое поле **R.id.edt2** и цвет текста этого поля становится красным (см. Листинг 1.6).

Следующие примеры изменения состояний виджетов рассматриваемого примера изображены на Рис. 1.6, который также как и предыдущее изображение состоит из нескольких этапов, и каждый этап представлен соответствующим снимком экрана устройства.

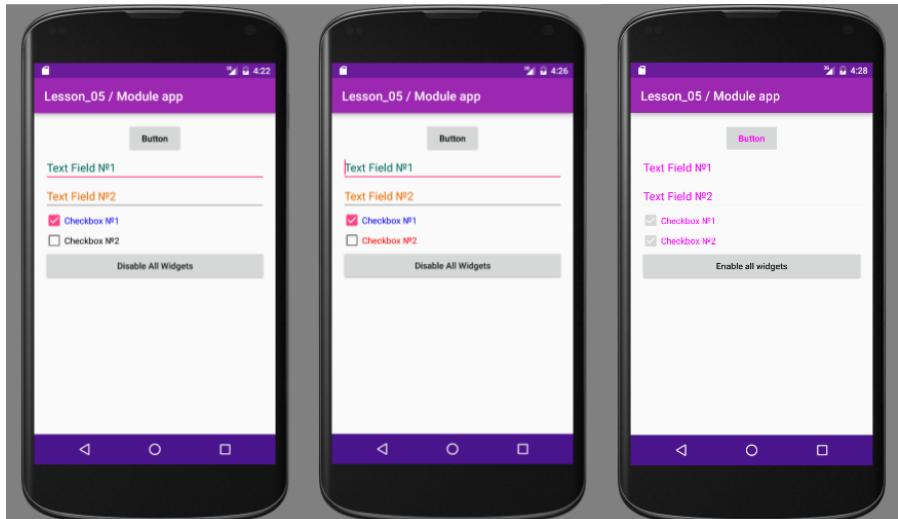


Рис. 1.6. Внешний вид работы примера из Листингов 1.5–1.8.

На левом снимке изображения Рис. 1.6 флажок R.id.cb1 (с надписью *Checkbox №1*) находится в отмеченном галочкой состоянии (android:state_checked="true") и, в соответствии с файлом ресурсов /res/color/checkbox_text.xml (Листинг 1.7), имеет цвет текста синий. На центральном снимке на флажок R.id.cb2 (с надписью *Checkbox №2*) осуществляется нажатие и поэтому цвет текста в соответствии с файлом ресурсов /res/color/checkbox_text.xml становится красным. На правом снимке все

тестируемые виджеты (идентификаторы `R.id.btn1`, `R.id.edt1`, `R.id.edt2`, `R.id.cb1`, `R.id.cb2`) находятся в запрещенном состоянии, и для этого состояния (`android:state_enabled="false"`) в файлах ресурсах *Color State List Resources* задан цвет текста виджетов — фиолетовый.

Код рассматриваемый в данном разделе находится в модуле *app* среди файлов с исходным кодом, которые прилагаются к данному уроку.

1.3. Drawable Resources.

Ресурсы графических изображений

Drawable Resources предназначены для размещения графических изображений, которые используются приложением как части пользовательского интерфейса. Графика, размещаемая в *Drawable Resources*, может быть файлами изображений (растровые изображения в форматах `*.png`, `*.jpg`, `*.gif`), может быть xml файлами ресурсов, которые содержат дополнительную информацию о растром изображении в виде xml данных (так называемый *Xml Bitmap Drawable* файл изображения), может быть xml файлами ресурсов содержащих информацию о рисовании фигур (так называемые *Shape Drawable*).

Главная информация о ресурсах *Drawable Resources* для файлов растровых изображений:

- Расположение файлов : `res/drawable/filename.png` (`.png`, `.jpg`, `.gif`).
- Имя файла ресурсов впоследствии используется как идентификатор (resource ID) ресурса.

- Данный тип ресурсов компилируется в объекты класса **android.graphics.drawable.BitmapDrawable**.
- Для ссылки из кода Java к ресурсам данного типа используется обращение: **R.drawable.filename**.
- Для ссылки из xml к ресурсам данного типа используется обращение: **@[package:]drawable/filename**.

Например, скопируем файл растрового изображения в каталог /res/drawable (на жестком диске этот каталог будет расположен: путь_к_проекту/имя_модуля/src/main/res/drawable, и скопировать файл в этот каталог можно с помощью обычного файлового менеджера). В рассматриваемом в данном разделе примере (модуль *app2*) файл с изображением называется **img_network.png** (изображение загружено с сайта <http://iconsearch.ru>). Так вот, после копирования этого файла в /res/drawable/img_network.png, отобразим этот файл с помощью виджета **android.widget.ImageView**, как показано в Листинге 1.9. Внешний вид работы примера из Листинга 1.9 изображен на Рис. 1.7.

Листинг 1.9. Отображение в виджете **android.widget.ImageView** файла с растровым изображением, размещенного в ресурсах Drawable Resources приложения

```
<ImageView  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
    android:layout_gravity="center_horizontal"  
    android:src="@drawable/img_network"  
    android:id="@+id/ivOne"  
/>
```



Рис. 1.7. Внешний вид работы примера из Листинга 1.9.

Растровые файлы ресурсов *Drawable Resources* можно получать и с помощью Java программного кода, как показано в Листинге 1.10.

Листинг 1.10. Программное извлечение файла с растровым изображением из ресурсов *Drawable Resources*

```
Resources res = getResources();
BitmapDrawable drawable = (BitmapDrawable)
    res.getDrawable(R.drawable.img_network,
    this.getTheme());
Bitmap bmp = drawable.getBitmap();
ImageView ivTwo = (ImageView) this.findViewById
    (R.id.ivTwo);
ivTwo.setImageBitmap(bmp);
```

Необходимо отметить, что получаемые из *Drawable Resources* файлы в виде объектов `android.graphics.Bitmap`, являются не модифицируемыми объектами и попытка внести в `android.graphics.Bitmap` объект изменения (например, нарисовать пиксели) приведет к ошибке исполнения программы. Поэтому, для получения объектов `android.graphics.Bitmap`, в которые требуется внести изменения, используется метод класса `android.graphics.Bitmap`:

```
Bitmap copy(Bitmap.Config config,
           boolean isMutable);
```

он создает копию объекта `android.graphics.Bitmap`, которую можно сделать модифицируемой с помощью параметра `isMutable`. Пример программного извлечения из *Drawable Resources* ресурсов файла с растровым изображением, в который программно вносятся изменения (рисуются дополнительные пиксели) приведен в Листинге 1.11.

Внешний вид работы примера из Листинга 1.11 изображен на Рис. 1.8 — в левой части изображения /res/drawable/img_network нарисованы пиксели коричневого цвета, в виде вертикальной линии толщиной 3 пикселя.

Листинг 1.11. Программное извлечение
растрового изображения из ресурсов,
с внесением в это изображение изменений

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

```
/*
 * Software extraction of a raster image from
 * resources with the introduction of changes
 * into this image.
 *
 * Программное извлечение растрового
 * изображения из ресурсов, с внесением в это
 * изображение изменений.
 *
 * -----
 */
try
{
    Resources res = getResources();
    BitmapDrawable drawable = (BitmapDrawable)
        res.getDrawable(R.drawable.img_network,
        this.getTheme());
    Bitmap bmp = drawable.getBitmap();
    // ----- Getting a copy of the Bitmap object
    // ----- for modification -----
    // ----- Получение копии объекта Bitmap для
    // ----- модификации -----
    bmp = bmp.copy(Bitmap.Config.ARGB_8888,
        true);
    int clr = Color.rgb(0x79, 0x55, 0x48);

    // ----- Drawing pixels in a Bitmap object ---
    // ----- Рисование пикселей в объекте Bitmap --
    for (int i = 0; i < bmp.getHeight(); i++)
    {
        bmp.setPixel(0, i, clr);
        bmp.setPixel(1, i, clr);
        bmp.setPixel(2, i, clr);
    }
}
```

```
// ----- Place the Bitmap object in the widget
// ----- android.widget.ImageView -----
// ----- Размещение объекта Bitmap в виджете
// ----- android.widget.ImageView -----
    ImageView ivTwo = (ImageView) this.
        findViewById(R.id.ivTwo);
    ivTwo.setImageBitmap(bmp);
}

catch (Exception e)
{
}
}
```



Рис. 1.8. Внешний вид работы примера из Листинга 1.11.

Далее, как уже упоминалось выше, в ресурсах Drawable Resources могут быть размещены и xml файлы, содержащие дополнительные параметры для растровых изображений. В справочнике по API для Android разработчиков такой подвид Drawable Resources называют Xml Bitmap Drawable. Давайте познакомимся с таким подвидом ресурсов Drawable Resources.

Ресурсы **Xml Bitmap Drawable**, описанные в xml файлах, отображаются в приложении как обычные растровые файлы изображений. По сравнению с ресурсами для растровых файлов, для данного типа ресурсов можно указать дополнительные режимы, например можно указать режим «замощения плиткой» (tiling). А еще можно использовать элемент **<bitmap>** в качестве дочернего элемента **<item>** для ресурсов Color State List Resources.

Главная информация о ресурсах Xml Bitmap Drawable:

- Расположение файлов: res/drawable/filename.xml.
- Имя файла ресурсов впоследствии используется как идентификатор (resource ID) ресурса.
- Данный тип ресурсов компилируется в объекты класса [android.graphics.drawable.BitmapDrawable](#).
- Для ссылки из программного кода Java к ресурсам данного типа используется обращение: **R.drawable.filename**.
- Для ссылки из xml к ресурсам данного типа используется обращение: **@[package:]drawable/filename**.

Синтаксис содержимого файлов для ресурсов данного типа приведен в Листинге 1.12.

Листинг 1.12. Синтаксис xml файлов ресурсов для Xml Bitmap Drawable изображений

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:src=
        "@[package:]drawable/drawable_resource"
    android:antialias=["true" | "false"]
    android:dither=["true" | "false"]
    android:filter=["true" | "false"]
    android:gravity=["top" | "bottom" | "left" |
                    "right" | "center_vertical" |
                    "fill_vertical" |
                    "center_horizontal" |
                    "fill_horizontal" |
                    "center" | "fill" |
                    "clip_vertical" |
                    "clip_horizontal"]
    android:tileMode=["disabled" | "clamp" |
                      "repeat" | "mirror"] />
```

Описание элементов xml из Листинга 1.12:

- **<bitmap>** — обязательный атрибут.

Описывает растровое изображение и его свойства.

Атрибуты элемента **<bitmap>** :

- **xmlns:android** — определяет пространство имен xml.
Он должен иметь значение: <http://schemas.android.com/apk/res/android>. Этот атрибут является обязательным, если элемент **<bitmap>** является корневым элементом, если элемент **<bitmap>** является вложенным в элемент **<item>**, то этот атрибут не нужен.

- **android:src** — содержит ссылку на графический ресурс в виде значения `@[package:]drawable/идентификатор_ресурса`. Графический ресурс может быть как файлом растрового изображения, так и *Shape Drawable* ресурсом.
- **android:antialias** — тип Boolean. Разрешает или запрещает сглаживание (antialiasing) цвета граничных пикселей.
- **android:dither** — тип Boolean. Разрешает или запрещает сглаживание (dithering) для растрового изображения, когда количество битов на пиксель экрана не соответствует количеству битов на пиксель растрового изображения (например, изображение с глубиной 32-бита на пиксель необходимо отобразить на экране с глубиной цвета 16 бит).
- **android:filter** — тип Boolean. Разрешает или запрещает режим сглаживания (filtering) растрового изображения для случая, когда растровое изображение растянуто или сжато, по сравнению со своими оригинальными размерами.
- **android:gravity** — определяет расположение растрового изображения внутри родительского контейнера для случая, если растровое изображение по своим размерам меньше, чем размер родительского контейнера.
- **android:tileMode** — определяет режим «замощения плиткой», то есть повторения растрового изображения по вертикали и горизонтали. При использовании «замощения плиткой» значение атрибута

`android:gravity` игнорируется. Значение этого атрибута должно содержать одно из следующих значений: *disabled* (не использовать режим «замощения плиткой»), *clamp* (за пределами растрового изображения заполнить оставшуюся область цветом, который является граничным, располагающимся на краю растрового изображения), *repeat* (размножить растровое изображение по горизонтали и вертикали, то есть «замостить плиткой»), *mirror* (размножить растровое изображение по горизонтали и вертикали, при этом чередуя образ оригинального изображения с образом его зеркального отображения).

Рассмотрим пример для *Xml Bitmap Drawable* ресурсов. Для примера с сайта <http://iconsearch.ru> загружено изображение netbeans.png (см. Рис. 1.9), которое скопировано в каталог модуля *app2*: /res/drawable.



Рис. 1.9. Растворное изображение для использования в примере для *Xml Bitmap Drawable* ресурсов

Добавим в каталог /res/drawable два файла *Xml Bitmap Drawable* ресурсов. Для этого необходимо кликнуть правой кнопкой мыши по каталогу *drawable* модуля и, в появившемся контекстном меню, выбрать пункты *New/Drawable Resource File* (см. Рис. 1.10). В появившемся окне необходимо указать имя xml файла ресурсов.

Файлы ресурсов *Xml Bitmap Drawable* для рассматриваемого нами примера назовем *my_netbeans.xml* (*/res/drawable/my_netbeans.xml*) и *my_netbeans_mirror.xml* (*/res/drawable/my_netbeans_mirror.xml*). Содержимое этих файлов приведено в Листингах 1.13 и 1.14 соответственно.

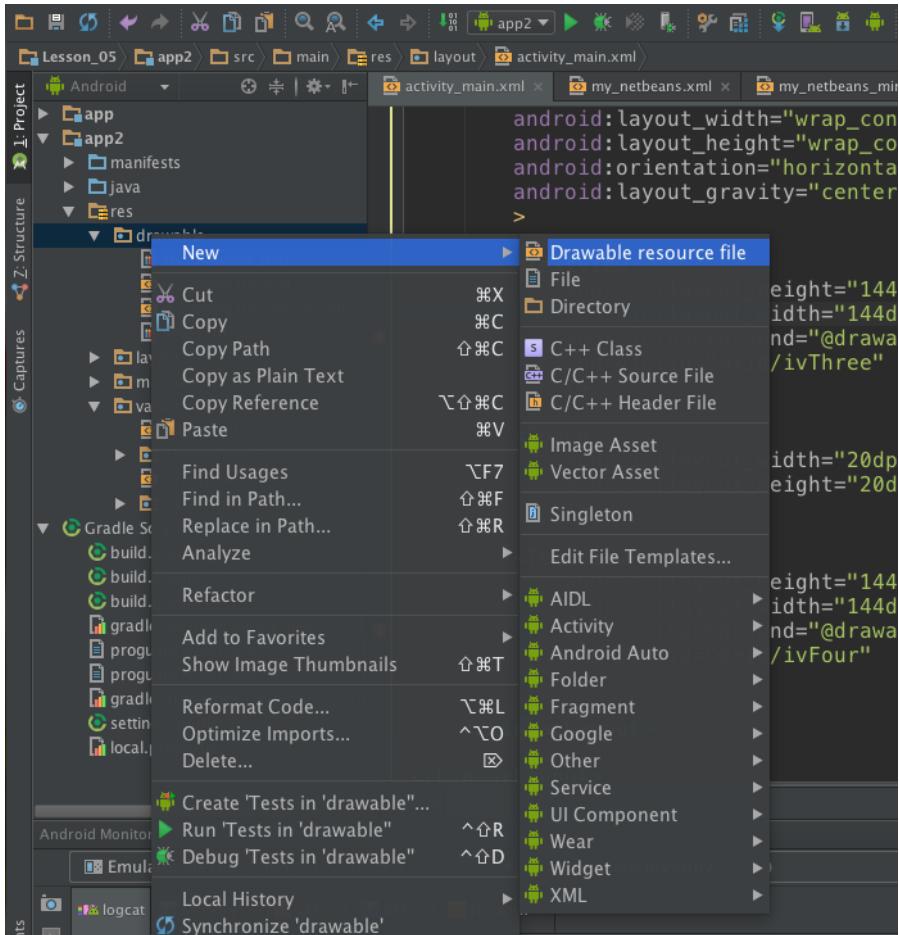


Рис. 1.10. Добавление в модуль проекта файла ресурсов *Xml Bitmap Drawable*

Листинг 1.13. Содержимое файла

/res/drawable/my_netbeans.xml, которое описывает
Xml Bitmap Drawable ресурс для рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:src="@drawable/netbeans"
    android:tileMode="repeat">
</bitmap>
```

Листинг 1.14. Содержимое файла

/res/drawable/my_netbeans_mirror.xml, которое описывает
Xml Bitmap Drawable ресурс для рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:src="@drawable/netbeans"
    android:tileMode="mirror">
</bitmap>
```

В Листинге 1.13 описывается графический ресурс приложения *Xml Bitmap Drawable* (/res/drawable/my_netbeans.xml), который на основе растрового изображения (/res/drawable/netbeans.png) задает режим «замощения плиткой» этого изображения. В Листинге 1.14 описывается графический ресурс приложения *Xml Bitmap Drawable* (/res/drawable/my_netbeans_mirror.xml), который на основе растрового изображения (/res/drawable/netbeans.png) задает режим «замощения плиткой» этого изображения с зеркальным

отображением. Для применения этих ресурсов в файле макета Активности /res/layout/activity_main.xml добавим два виджета `android.widget.TextView`, которым в качестве фона (с помощью атрибута `android:background`) назначим ресурсы из Листингов 1.13 и 1.14. Создание виджетов `android.widget.TextView` (идентификаторы `R.id.tvThree` и `R.id.tvFour`) и назначение им *Xml Bitmap Drawable* ресурсов приведено в Листинге 1.15.

Листинг 1.15. Файл /res/layout/activity_main.xml
с виджетами `android.widget.TextView`, которым назначены
в качестве фона ресурсы *Xml Bitmap Drawable*

```
<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"
    tools:context="itstep.com.myapp2.MainActivity">

    ..

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

        <TextView
            android:layout_height="144dp"
            android:layout_width="144dp"
            android:background="@drawable/my_netbeans"
            android:id="@+id/tvThree" />
```

```
<Space  
    android:layout_width="20dp"  
    android:layout_height="20dp" />  
  
<TextView  
    android:layout_height="144dp"  
    android:layout_width="144dp"  
    android:background=  
        "@drawable/my_netbeans_mirror"  
    android:id="@+id/tvFour" />  
  
</LinearLayout>  
  
</LinearLayout>
```

Внешний вид работы примера из Листингов 1.13, 1.14, 1.15 изображен на Рис. 1.11.

Слева на Рис. 1.11 находится виджет `android.widget.TextView` (идентификатор `R.id.tvThree`), которому назначен в качестве фона Xml Bitmap Drawable ресурс `@drawable/my_netbeans` (см. Листинг 1.13), с режимом «замощения плиткой».

Справа на Рис. 1.11 находится виджет `android.widget.TextView` (идентификатор `R.id.tvFour`), которому назначен в качестве фона Xml Bitmap Drawable ресурс `@drawable/my_netbeans_mirror` (см. Листинг 1.14), с режимом «замощения плиткой» и зеркаливанием.

Полный исходный код примера находится в модуле `app2` среди файлов исходного кода, которые прилагаются к данному уроку.



Рис. 1.11. Внешний вид работы примера из Листингов 1.13, 1.14, 1.15

Теперь перейдем к рассмотрению следующего типа графических растровых ресурсов — **Shape Drawable**. С помощью данного подвида графических ресурсов, можно создавать изображения простейших графических фигур при помощи xml файлов.

Главная информация о ресурсах Shape Drawable следующая:

- Расположение файлов: res/drawable/filename.xml.
- Имя файла ресурсов впоследствии используется как идентификатор (resource ID) ресурса.

- Этот тип ресурсов компилируется в объекты класса **android.graphics.drawable.GradientDrawable**.
- Для ссылки из кода Java к ресурсам данного типа используется обращение: **R.drawable.filename**.
- Для ссылки из xml к ресурсам данного типа используется обращение: **@[package:]drawable/filename**.

Синтаксис *Shape Drawable* ресурсов приведен в Листинге 1.16.

Листинг 1.16. Синтаксис xml файла для ресурсов Shape Drawable

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:shape=[ "rectangle" | "oval" |
                  "line" | "ring" ] >

    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
        android:bottomRightRadius="integer" />

    <gradient
        android:angle="integer"
        android:centerX="integer"
        android:centerY="integer"
        android:centerColor="integer"
        android:endColor="color"
        android:gradientRadius="integer"
        android:startColor="color"
        android:type=[ "linear" | "radial" | "sweep" ]
        android:useLevel=[ "true" | "false" ] />
```

```
<padding  
    android:left="integer"  
    android:top="integer"  
    android:right="integer"  
    android:bottom="integer" />  
  
<size  
    android:width="integer"  
    android:height="integer" />  
  
<solid  
    android:color="color" />  
  
<stroke  
    android:width="integer"  
    android:color="color"  
    android:dashWidth="integer"  
    android:dashGap="integer" />  
/</shape>
```

Описание элементов xml из Листинга 1.16:

- Элемент **<shape>** — корневой элемент xml файла.
Атрибуты этого элемента:
 - **android:shape** — определяет тип геометрической фигуры. Принимает одно из следующих значений: **rectangle** (Прямоугольник, заполняющий виджет, которому назначен этот ресурс), **oval** (Овал, соответствующий размерам виджета, которому назначен ресурс), **line** (Горизонтальная линия, которая проходит по всей ширине виджета которому назначен ресурс. Эта фигура требует наличия в xml документе элемента **<stroke>** определяющего толщину линии — см. Листинг 1.16), **ring** (Кольцо).

Следующие атрибуты используются только для типа фигуры «Кольцо» (`android:shape="ring"`).

- ▷ **android:innerRadius** — тип данных — размер (Dimension). Размер радиуса внутренней части кольца. Размер можно задать непосредственно значением или ссылкой на ресурс `@dimen/имя_ресурса`.
- ▷ **android:innerRadiusRatio** — тип — вещественное число. Радиус внутренней части кольца в виде отношения к внешнему радиусу (ширине) кольца. Например, если значение этого атрибута `android:innerRadiusRatio="3"`, то это означает, что величина внутреннего радиуса равна внешнему радиусу кольца, поделенному на 3. Этот атрибут игнорируется, если задан атрибут **android:innerRadius**. Значение по умолчанию для этого атрибута равно 9.
- ▷ **android:thickness** — размер. Толщина кольца. Размер можно задать непосредственно значением или ссылкой на ресурс `@dimen/имя_ресурса`.
- ▷ **android:thicknessRatio** — тип — вещественное число. Толщина кольца (разница между внешним и внутренним радиусом), в виде отношения к внешнему радиусу (ширине) кольца. Например, если значение этого атрибута `android:thicknessRatio="2"`, то толщина кольца равна ширине кольца, поделенной на 2. Значение по умолчанию для этого атрибута равно 3. Этот атрибут игнорируется если задан атрибут **android:innerRadius**.
- Элемент `<corners>` — описывает закругленные углы фигуры. Этот элемент используется только для типа

фигуры «Прямоугольник»(**android:shape="rectangle"**).

Атрибуты этого элемента:

- **android:radius** — тип — размер. Задает радиус всех углов фигуры.
- **android:topLeftRadius** — размер. Радиус левого верхнего угла фигуры.
- **android:topRightRadius** — размер. Радиус правого верхнего угла фигуры.
- **android:bottomLeftRadius** — размер. Радиус левого нижнего угла фигуры.
- **android:bottomRightRadius** — размер. Радиус правого нижнего угла фигуры.

Все размеры радиусов можно задать непосредственно значением или ссылкой на ресурс **@dimen/имя_ресурса**.

Каждый радиус угла должен иметь значение большее 1, в противном случае — закругление угла не произойдет.

- Элемент **<gradient>** — определяет параметры градиента цвета для фигуры. Атрибуты этого элемента:
 - **android:angle** — тип — целое число (*integer*). Угол наклона градиента, где 0 соответствует направлению градиента слева направо, 90 — соответствует направлению градиента снизу вверх. Величина градиента для этого атрибута должна быть кратна 45. Значение по умолчанию для этого атрибута 0.
 - **android:centerX** — тип — *float*. Относительное расположение центра градиента по оси X. Значение этого атрибута должно быть в диапазоне от 0 до 1.0.

- **android:centerY** — тип — *float*. Относительное расположение центра градиента по оси Y. Значение этого атрибута должно быть в диапазоне от 0 до 1.0.
 - **android:centerColor** — тип — *Color*. Дополнительный цвет, являющийся промежуточным цветом между цветами начала и конца градиента. Величина этого цвета задается в виде шестнадцатеричного значения или идентификатора ресурса *Color Resources*.
 - **android:endColor** — тип — *Color*. Цвет конца градиента. Величина этого цвета задается в виде шестнадцатеричного значения или идентификатора ресурса *Color Resources*.
 - **android:gradientRadius** — тип — *float*. Радиус градиента. Используется только для типа фигуры *android:type="radial"*.
 - **android:startColor** — тип — *Color*. Цвет начала градиента. Величина этого цвета задается в виде шестнадцатеричного значения или идентификатора ресурса *Color Resources*.
 - **android:type** — тип градиента. Допустимо одно из следующих значений: **linear** (Линейный градиент), **radial** (Радиальный градиент. Цвет начала градиента находится в центре), **sweep** (Обводной градиент).
- Элемент **<padding>** — внутренние отступы для виджета, содержащего фигуру. Атрибуты этого элемента:
- **android:left** — размер. Величина отступа слева. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.

- **android:top** — размер. Величина отступа сверху. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.
 - **android:right** — размер. Величина отступа справа. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.
 - **android:bottom** — размер. Величина отступа снизу. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.
- Элемент **<size>** — размер фигуры. Атрибуты этого элемента:
- **android:height** — размер. Высота фигуры. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.
 - **android:width** — размер. Ширина фигуры. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.

Примечание. Размеры фигуры растягиваются (масштабируются) до границ виджета, в котором размещена фигура пропорционально размерам, описанными этими атрибутами. Это происходит по умолчанию. Поэтому необходимо запрещать масштабирование фигуры, например для виджета `android.widget.ImageView` запрет масштабирования осуществляется с помощью `android:scaleType="center"`.

- Элемент **<solid>** — цвет сплошной заливки фигуры. Атрибуты этого элемента:
 - **android:color** — тип — Color. Величина цвета заливки. Величина этого цвета задается в виде шестнадцатеричного значения или идентификатора ресурса *Color Resources*.
- Элемент **<stroke>** — линия, очерчивающая фигуру по контуру. Атрибуты этого элемента:
 - **android:width** — размер. Толщина линии. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*.
 - **android:color** — тип — Color. Цвет линии. Величина этого цвета задается в виде шестнадцатеричного значения или идентификатора ресурса *Color Resources*.
 - **android:dashGap** — размер. Расстояние между штрихами линии. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*. Этот атрибут действителен, только если установлен атрибут **android:dashWidth**.
 - **android:dashWidth** — размер. Размер каждого штриха линии. Значение этого атрибута задается в виде значения или в виде идентификатора ресурсов *Dimension Resources*. Этот атрибут действителен, только если установлен атрибут **android:dashGap**.

Теперь перейдем к рассмотрению примеров создания и применения *Shape Drawable* ресурсов. В проект файл ресурсов *Shape Drawable* добавляется таким же способом как и добавление *Xml Bitmap Drawable* (см. Рис. 1.10). Для первого примера создадим файл ресурсов *ShapeDrawable*

с именем my_rectangle.xml (/res/drawable/my_rectangle.xml). Содержимое этого файла приведено в Листинге 1.17.

Листинг 1.17. Содержимое файла ресурсов
Shape Drawable/res/drawable/my_rectangle.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <size
        android:height="100dp"
        android:width="200dp" />

    <corners android:radius="10dp" />
    <solid android:color="#003366" />

    <stroke
        android:color="#FFFF00"
        android:width="2dp"
        android:dashGap="2dp"
        android:dashWidth="6dp" />
</shape>
```



Рис. 1.12. Внешний вид фигуры из Листинга 1.17

Как видно из Листинга 1.17, в файле ресурсов /res/drawable/my_rectangle.xml создается фигура «Прямоугольник» размером 200dp по ширине и 100dp по высоте. Фигура «Прямоугольник» имеет одинаковые закругленные углы радиусом 10dp. Фигура залита сплошным цветом, со значением #003366, и имеет желтую (#FFFF00) линию обводки, толщиной 2dp. Линия обводки является штриховой, с длиной штриха 6dp и расстоянием между штрихами 2dp. Внешний вид фигуры «Прямоугольник» изображен на Рис. 1.12. Попробуйте самостоятельно применить к фигуре из Листинга 1.12 разные значения радиусов для разных углов прямоугольника (атрибуты android:topLeftRadius, android:topRightRadius, android:bottomLeftRadius, android:bottomRightRadius).

Следующий пример создание фигуры «Кольцо». Для этой цели создадим файл ресурсов Shape Drawable /res/drawable/my_ring.xml. Содержимое этого файла приведено в Листинге 1.18.

Листинг 1.18. Содержимое файла ресурсов Shape Drawable /res/drawable/my_ring.xml

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:shape="ring"
    android:useLevel="false"
    android:innerRadius="50dp"
    android:thickness="90dp" >
    <solid android:color="#FF8F00" />
</shape>
```



Рис. 1.13. Внешний вид фигуры из Листинга 1.18

Теперь рассмотрим, как выглядят градиенты. Начнем с линейного градиента (`<gradient android:type="linear" />`). Для этой цели создадим файл ресурсов *Shape Drawable* /res/drawable/my_gradient.xml. Содержимое этого файла приведено в Листинге 1.19.

Листинг 1.19. Пример фигуры с применением линейного градиента

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <size
        android:height="80dp"
        android:width="150dp" />
    <corners android:radius="10dp" />
    <gradient
        android:type="linear"
        android:angle="90"
        android:startColor="#FFFFFF"
        android:endColor="#FF0000" />
</shape>
```

Как видно из Листинга 1.19, создаваемая в файле ресурсов Shape Drawable фигура является прямоугольником размером 150dp на 80dp и радиусом закругления углов 10dp. Начальным цветом градиента является белый (#FFFFFF), конечным цветом градиента является красный (#FF0000). Угол наклона линейного градиента 90 градусов (направление снизу-вверх). Внешний вид фигуры с линейным градиентом изображен на Рис. 1.14.



Рис. 1.14. Внешний вид фигуры с линейным градиентом из Листинга 1.19

Далее познакомимся с радиальным (`<gradient android:type = "radial" />`) типом градиента и с обводным (`<gradient android:type="sweep" />`). Пример применения радиального типа градиента приведен в Листинге 1.20 (создан файл ресурсов `/res/drawable/my_gradient1.xml`).

Листинг 1.20. Пример радиального градиента

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

```
<gradient
    android:startColor="#0000FF"
    android:endColor="#00FF00"
    android:type="radial"
    android:gradientRadius="60dp" />

<size android:height="100dp" android:width="100dp" />
<stroke android:width="2dp"
    android:color="#FFFF00" />

</shape>
```

Как видно из Листинга 1.20, для примера создана фигура «Овал» с шириной 100dp и высотой 100dp. Фигура имеет желтую (#FFFF00) линию обводки, толщиной 3dp. Центр радиального градиента находится в центре фигуры и градиент имеет радиус 60dp. Начальный цвет градиента синий (#0000FF). Конечный цвет градиента — зеленый (#00FF00). Внешний вид фигуры с радиальным градиентом изображен на Рис. 1.15. Для этого типа градиента так же можно использовать атрибуты **android:centerX**, **android:centerY**, **android:centerColor**. Попробуйте применить их самостоятельно.

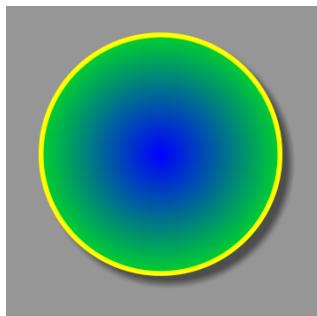


Рис. 1.15. Внешний вид фигуры Shape Drawable из Листинга 1.20

Пример применения обводного типа градиента приведен в Листинге 1.21 (создан файл ресурсов /res/drawable/my_gradient2.xml).

Листинг 1.21. Пример обводного градиента

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

    <gradient
        android:startColor="#0000FF"
        android:endColor="#00FF00"
        android:type="sweep" />

    <size android:height="100dp" android:width="100dp" />
    <stroke android:width="2dp"
        android:color="#FFFF00" />

</shape>
```

Как мы видим из Листинга 1.21, для примера создана фигура «Овал», с шириной 100dp и высотой 100dp. Фигура имеет желтую (#FFFF00) линию обводки, толщиной 3dp. Начальный цвет градиента синий (#0000FF). Конечный цвет градиента — зеленый (#00FF00). Внешний вид фигуры с обводным градиентом вы можете увидеть на Рис. 1.16.

Для этого типа градиента так же можно использовать атрибуты **android:centerX**, **android:centerY**, **android:centerColor**. Попробуйте применить их самостоятельно.

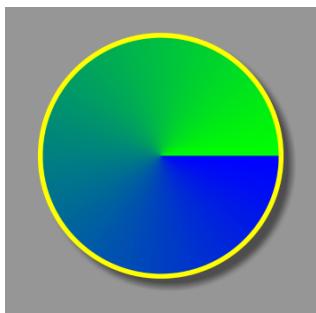


Рис. 1.16. Внешний вид фигуры Shape Drawable из Листинга 1.21

Рассмотренные в Листингах 1.17–1.21 примеры можно считать так называемой «пробой пера». Следующим примером будет создание ресурса *Shape Drawable*, для применения в виджете `android.widget.ImageButton` (кнопка, внешний вид которой задается с помощью изображений, [описание класса доступно по ссылке](#)). В этом примере будем использовать градиентную заливку фигуры «Прямоугольник». Создадим новый файл ресурсов `/res/drawable/my_rectangle2.xml`. Внешний вид этого файла представлен в Листинге 1.22.

Листинг 1.22. Фигура «Прямоугольник» с градиентной заливкой

```
<shape  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    android:shape="rectangle">  
    <size  
        android:height="80dp"  
        android:width="150dp" />  
    <corners android:radius="10dp" />
```

```
<gradient
    android:angle="0"
    android:startColor="#FFEA00"
    android:endColor="#FFAB40" />

<stroke
    android:color="#E65100"
    android:width="3dp" />
</shape>
```

Как видно из Листинга 1.22, создаваемая фигура является прямоугольником, с размером 150dp по ширине и 80dp по высоте. Имеет оранжевую (#E65100) линию обводки, толщиной 3dp. Заливка фигуры осуществляется градиентом, с углом наклона в 0 градусов (направление — слева на право), от желтого цвета (#FFEA00) до оранжевого (#FFAB40). Внешний вид фигуры изображен на Рис. 1.17.



Рис. 1.17. Внешний вид фигуры Shape Drawable из Листинга 1.22

Теперь применим фигуру из файла `/res/drawable/my_rectangle2.xml` (Листинг 1.22) в качестве фонового изображения для виджета `android.widget.ImageButton`. В качестве первичного изображения для виджета `android.widget.ImageButton` будет использовано изображение галочки зеленого цвета, которая, как правило,

символизирует действие «Применить» (Apply). В Листинге 1.23 приведен xml код создания интересующего нас виджета **android.widget.ImageButton**.

Листинг 1.23. Создание виджета android.widget.ImageButton с использованием ресурса Shape Drawable

```
<ImageButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:padding="0dp"  
    android:background="@drawable/my_rectangle2"  
    android:src="@drawable/apply"  
    android:onClick="btnClick"  
    />
```

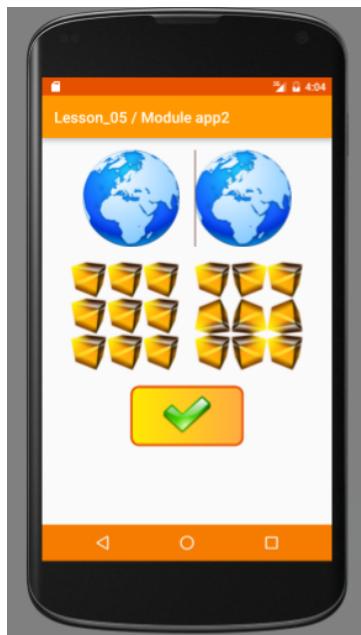


Рис. 1.18. Внешний вид виджета android.widget.ImageButton из Листинга 1.23

Описанный в Листинге 1.23 виджет находится в файле макета Активности /res/layout/acitivity_main.xml модуля app2. Внешний вид виджета **android.widget.ImageButton** изображен на Рис. 1.18. Виджет является кнопкой и имеет обработчик события нажатия — метод **btnClick**.

Можно ли использовать изображения из Shape Drawable не только для виджета **android.widget.ImageButton**, но и для других виджетов, например для виджетов **android.widget.Button** и **android.widget.EditText**? Конечно можно! И сейчас мы это сделаем. Давайте применим фигуру из Листинга 1.22 для виджета **android.widget.Button**. В Листинге 1.24 приведен интересующий нас пример.

Листинг 1.24. Применение фигуры из Листинга 1.22 для виджета android.widget.Button

```
<Button  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="@drawable/my_rectangle2"  
    android:text="Click Me"  
    android:textAllCaps="false"  
    android:textSize="12pt"  
    android:textColor="#FF0000"  
    android:onClick="btnClick"  
/>
```

Как видно из Листинга 1.24, применение фигуры *Shape Drawable* для виджета **android.widget.Button** (в качестве фонового изображения) ничем не отличается от применения для виджета **android.widget.ImageButton**. Внешний вид примера изображен на Рис. 1.19.

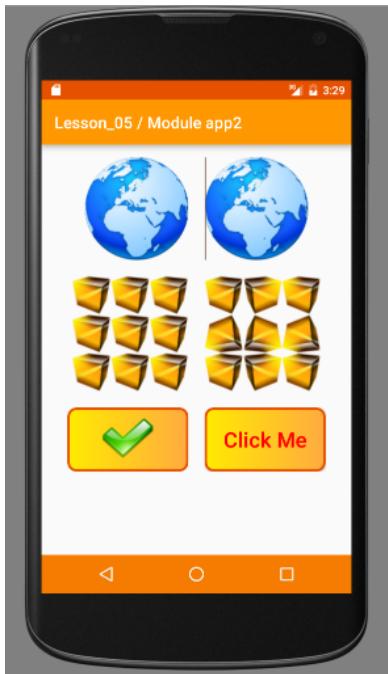


Рис. 1.19. Внешний вид виджета `android.widget.Button` из Листинга 1.24

Далее, рассмотрим применение ресурсов *Shape Drawable* для виджета `android.widget.EditText`. Особенностью этого примера является тот факт, что в отличие от кнопок (`android.widget.Button`, `android.widget.ImageButton`), текстовое поле `android.widget.EditText` может иметь переменный размер. То есть, если для кнопки приемлем фиксированный размер фигуры, то для текстового поля `android.widget.EditText` фиксированный размер фигуры не приемлем. Чтобы фигура из ресурсов *Shape Drawable* не имела фиксированного размера, ей не нужно задавать размер с помощью элемента

<size>. В этом случае лучше воспользоваться отступами (padding). Создадим файл для *Shape Drawable* ресурсов /res/drawable/my_rectangle3.xml. Содержимое этого файла приведено в Листинге 1.25.

Листинг 1.25. Создание фигуры Shape Drawable для применения в качестве фонового изображения для виджета android.widget.EditText

```
<?xml version="1.0" encoding="utf-8"?>

<shape
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <corners android:radius="15dp" />

    <gradient
        android:type="linear"
        android:angle="90"
        android:startColor="#9E9D24"
        android:endColor="#FFFFFF" />

    <stroke
        android:width="2dp"
        android:color="#3E2723" />

    <padding
        android:top="15dp"
        android:right="15dp"
        android:bottom="15dp"
        android:left="15dp" />

</shape>
```

Как видно из Листинга 1.25, созданная фигура является прямоугольником, с радиусом закругленных углов 15dp. Фигура имеет линию обводки, толщиной 2dp коричневого (#3E2723) цвета. Фигура заливается с помощью линейного градиента с углом 90 градусов (снизу-вверх). Начальным цвет градиента является хаки (#9E9D24), конечным цветом градиента является белый (#FFFFFF) цвет. Фигуре «Прямоугольник» не задан размер, зато заданы величины внутренних отступов (элемент `<padding>`). Размер внутренних отступов со всех сторон одинаков и равен 15dp. Отсутствие размера (элемент `<size>`) у фигуры обеспечивает растягивание фигуры под размеры виджета, а внутренний отступ (элемент `<padding>`) обеспечивает расстояние между содержимым виджета и рамкой фигуры.

Применим созданную (см. Листинг 1.25) фигуру в качестве фонового изображения для виджета `android.widget.EditText`. Пример этого применения приведен в Листинге 1.26. Сам виджет `android.widget.EditText` из Листинга 1.26 находится в файле верстки внешнего вида макета Активности (`/res/layout/activity_main.xml`).

Листинг 1.26. Применение фигуры из Листинга 1.25
в качестве фонового изображения для виджета
`android.widget.EditText`

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:background="@drawable/my_rectangle3"  
    android:text="Hello World"  
    android:textColor="#827717"  
    android:textSize="10pt"  
/>
```

Внешний вид работы примеров из Листингов 1.25 и 1.26 изображен на Рис. 1.20. Обратите внимание, что при вводе текста большого объема текстовое поле `android.widget.EditText` увеличивает свой размер и вместе с его размером увеличивается и размер фонового изображения (Рис. 1.20).

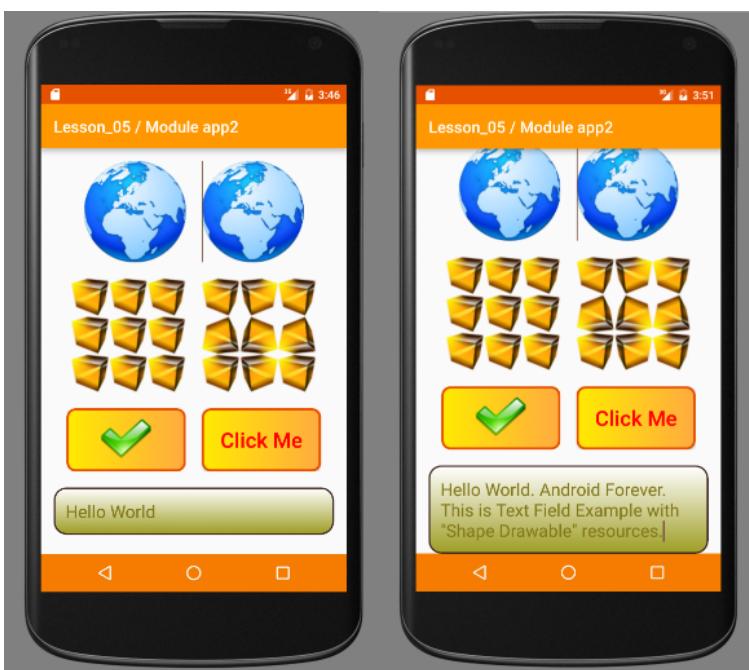


Рис. 1.20. Внешний вид работы примеров из Листингов 1.25 и 1.26.

Все примеры, рассмотренные в данном разделе урока, можно найти в модуле *app2* среди файлов с исходными кодами, которые прилагаются к данному уроку.

Можно объединять фигуры Shape Drawable, Xml Bitmap и Bitmap Drawable в так называемые массивы при

помощи **Layer List** («Список слоев») ресурсов. Layer List ресурсы компилируются в объекты [android.graphics.drawable.LayerDrawable](#). Элементы «Списка слоев» отображаются в порядке их нахождения в массиве — самый первый элемент будет отображен под всеми остальными элементами, а самый последний элемент будет отображен над всеми элементами. Синтаксис файлов ресурсов Layer List приведен в Листинге 1.27. Файлы ресурсов Layer List размещаются в папке /res/drawable/filename.xml.

Листинг 1.27. Синтаксис файла ресурсов Layer List

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android=
        "http://schemas.android.com/apk/res/android" >
    <item
        android:drawable=
            "@[package:]drawable/drawable_resource"
        android:id="@[+] [package:]id/resource_name"
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" >

        <!-- Description of the Drawing Resource -->

    </item>
</layer-list>
```

Как мы можем видеть из Листинга 1.27, корневым элементом файла ресурсов *Layer List* является элемент

<layer-list>. Дочерними элементами для элемента **<layer-list>** являются элементы **<item>**. Элемент **<item>** описывает изображение, которое размещается в ресурсе *Layer List* в позиции, которая указывается в атрибутах элемента **<item>**.

Элемент **<item>** имеет следующие атрибуты:

- **android:drawable** — ссылка на *Drawable Resources*.
- **android:id** — идентификатор изображения, которое описывается текущим элементом **<item>**. Идентификатор имеет вид "@+id/название_идентификатора". Идентификатор можно использовать для получения ссылки на объект изображения при помощи вызова метода **findViewById()**.
- **android:top** — смещение в пикселях изображения сверху.
- **android:right** — смещение в пикселях изображения справа.
- **android:bottom** — смещение в пикселях изображения снизу.
- **android:left** — смещение в пикселях изображения слева.

Элемент **<item>** вместо ссылки на изображение (**android:drawable**) может содержать дочерние элементы (например: **<bitmap>**, **<shape>**), которые непосредственно описывают изображение для элемента **<item>**.

Давайте рассмотрим пример *Layer List* ресурсов. Для этого создадим в модуле *app2* еще один файл ресурсов *Drawable Resources*: */res/drawable/my_layer_list.xml*. Содержимое этого файла приведено в Листинге 1.28.

Листинг 1.28. Пример Layer List ресурса — файл ресурсов /res/drawable/my_layer_list.xml

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

    <item android:top="0dp" android:left="0dp">
        <bitmap
            android:src="@drawable/img_network"
            android:gravity="center" />
    </item>

    <item android:top="-40dp" android:left="-40dp">
        <bitmap
            android:src="@drawable/apply"
            android:gravity="center" />
    </item>

    <item android:top="40dp" android:left="40dp">
        <bitmap
            android:src="@drawable/netbeans"
            android:gravity="center" />
    </item>
</layer-list>
```

Как видно из Листинга 1.28, файл ресурсов /res/drawable/my_layer_list.xml описывает *Layer List*, состоящий из трех изображений. Эти изображения уже знакомы нам по предыдущим примерам текущего модуля *app2* и находятся в папке /res/drawable. Первое изображение (впервые можно увидеть на Рис. 1.7, располагается в файле /res/drawable/img_network.png и имеет идентификатор **@drawable/img_network**) располагается в ресурсе *Layer List* со смещением 0dp сверху и 0dp слева.



Рис. 1.21. Внешний вид ресурса Layer List из Листинга 1.28

Второе изображение (зеленая галочка *Apply* из Рис. 1.19, располагается в файле ресурсов /res/drawable/apply.png и имеет идентификатор @drawable/apply) располагается в ресурсе *Layer List* со смещением -40dp сверху и -40dp слева. Такое смещение сдвигает изображение @drawable/apply вверх и влево относительно центра *Layer List*. Если установить смещение для @drawable/apply в 0dp сверху и 0dp слева, то изображение @drawable/apply разместится ровно по центру *Layer List* (вам рекомендуется попробовать это самостоятельно).

И последнее, третье изображение (изображено на Рис. 1.9, располагается в файле ресурсов /res/drawable/netbeans.png и имеет идентификатор @drawable/netbeans) располагается в ресурсе *Layer List* со смещением 40dp сверху и 40dp слева. Такое смещение сдвигает изображение @drawable/netbeans вниз и вправо относительно центра *Layer List*. Внешний вид ресурса *Layer List* из Листинга 1.28 изображен на Рис. 1.21. Применить полученное, составное с помощью *Layer List*, изображение можно, как показано в Листинге 1.29.

Листинг 1.29. Применение ресурса Layer List для виджета android.widget.ImageView

```
<ImageView
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/my_layer_list"
    />
```

Внешний вид работы примера из Листингов 1.28, 1.29 изображен на Рис. 1.23.

Рассмотрим еще один пример *Layer List* ресурсов. Для этого создадим в модуле *app2* еще один файл *Drawable* ресурсов: */res/drawable/my_layer_list2.xml*. Содержимое этого файла приведено в Листинге 1.30.

Листинг 1.30. Пример Layer List ресурса — файл ресурсов */res/drawable/my_layer_list2.xml*

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

    <item android:top="0dp" android:left="0dp">
        <shape android:shape="oval">
            <solid android:color="#B39DDB" />
            <size
                android:height="128dp"
                android:width="128dp" />
        </shape>
    </item>

    <item android:top="64dp" android:left="64dp">
        <shape android:shape="rectangle">
            <size
```

```
        android:height="64dp"
        android:width="64dp" />
    <corners android:radius="0dp" />
    <solid android:color="#673AB7" />

</shape>
</item>
</layer-list>
```

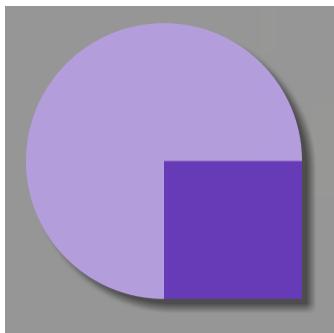


Рис. 1.22. Внешний вид ресурса Layer List из Листинга 1.30

Как видно из Листинга 1.30, изображения, которые являются элементами списка *Layer List* ресурса, описываются как *Shape Drawable* изображения (с помощью элемента `<shape>`). Внешний вид работы примера из Листинга 1.30 изображен на Рис. 1.22. Применить такое составное изображение можно так же, как показано в Листинге 1.29. Внешний вид работы примеров из Листингов 1.28, 1.29, 1.30 изображен на Рис. 1.23.

Все примеры, рассмотренные в данном разделе урока, можно найти в модуле *app2*, среди файлов с исходными кодами, которые прилагаются к данному уроку.



Рис. 1.23. Внешний вид работы примеров из Листингов 1.28, 1.29, 1.30

1.4. String Resources. Строковые ресурсы приложения. Локализация приложений

Строковые ресурсы предназначены для размещения строк, которые используются в приложении. В строковых ресурсах можно размещать как единичные строки, так и строковые массивы.

Единичные строки могут быть доступны из Java кода вашего приложения и из Xml файлов ресурсов (таких как, например, файлы с макетами внешнего вида). Размещаются в файлах res/values/имя_файла.xml. Каждая единичная строка должна находиться в элементе `<string>`:

```
<string name="greeting">Good morning!</string>
```

Атрибут **name** элемента **<string>** используется как идентификатор ресурсов для доступа к этому ресурсу:

- из программного кода Java:
R.string.значение_атрибута_name;
- из файлов ресурсов Xml:
@string/значение_атрибута_name.

Учитывая, что приложение может использовать достаточно много единичных строк и их размещение в отдельных файлах значительно усложняет работу со строковыми ресурсами, необходимо размещать множество строк в одном файле ресурсов с корневым элементом **<resources>**. При создании модуля проекта автоматически создается Xml файл для строковых ресурсов */res/values/strings.xml*. Синтаксис Xml файла строковых ресурсов приведен в Листинге 1.31.

Листинг 1.31. Синтаксис Xml файла для строковых ресурсов

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="name_of_string">string_text</
    string>
</resources>
```

Как видно из Листинга 1.31, корневым элементом xml файла ресурсов является элемент **<resources>**. В этом элементе размещаются элементы **<string>**. У каждого элемента **<string>** должен быть атрибут **name**, указывающий

название (т.е. идентификатор) строкового ресурса. Значением элемента <string> является текст строкового ресурса. Пример строкового ресурса приведен в Листинге 1.32.

Листинг 1.32. Пример Xml файла строковых ресурсов с единичными строками

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="greeting">Good morning!</string>
    <string name="bye">Good bye!</string>
</resources>
```

Примеры использования строковых ресурсов из Листинга 1.32 приведены в Листингах 1.33 и 1.34.

Листинг 1.33. Пример применения строкового ресурса из Листинга 1.32 в Xml файле макета раскладки виджетов Активности

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

Листинг 1.34. Пример применения строкового ресурса из Листинга 1.32 в Java коде класса Активности

```
String greeting = this.getString(R.string.greeting);
```

Отдельно необходимо отметить, что существует возможность создавать форматированные строки в строковых ресурсах. Например, для форматирования можно использовать язык разметки HTML, как показано в Листинге 1.35.

Листинг 1.35. Пример форматирования строкового ресурса при помощи элементов HTML

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title"><b>Android</b> <i>Forever</i>!
    </string>
</resources>
```

Более подробно о форматировании строковых ресурсов можно прочитать на сайте для разработчиков Android приложений по [ссылке](#).

Кроме единичных строк в Xml файлах строковых ресурсов можно размещать массивы строк. **Строковый массив** — это ресурс, на который можно ссылаться используя значение, указанное в атрибуте **name**, а не имя Xml файла. Таким образом, можно в одном Xml файле размещать как единичные строки, так и строковые массивы. При этом, корневым элементом Xml файла ресурсов будет элемент **<resources>**. Размещаются строковые массивы так же, как и единичные строки в файлах res/values/имя_файла.xml. Синтаксис строкового массива приведен в Листинге 1.36.

Листинг 1.36. Синтаксис строкового массива

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="string_array_name">
        <item>text_for_item_1</item>
        <item>text_for_item_2</item>
        ...
        <item>text_for_item_N</item>
    </string-array>
</resources>
```

Как видно из Листинга 1.36, элементом для обозначения строкового массива является элемент `<string-array>`. У этого элемента есть обязательный атрибут `name`, который задает имя строкового массива. Каждый элемент строкового массива размещается в элементе `<item>`. Имя строкового массива используется для получения значений ресурса строкового массива. Пример создания строкового массива приведен в Листинге 1.37.

Листинг 1.37. Пример создания строкового массива в файле ресурсов `/res/values/strings.xml`

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <string name="app_name"><?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">
        <b>Android</b> <i>Forever</i>!
    </string>

    <string-array name="months">
        <item>January</item>
        <item>February</item>
        <item>March</item>
        <item>April</item>
        <item>May</item>
        <item>June</item>
        <item>July</item>
        <item>August</item>
        <item>September</item>
        <item>October</item>
        <item>November</item>
        <item>December</item>
    </string-array>
</resources>
</resources>
```

```
<string name="greeting">Good morning!</string>
<string name="bye">Good bye!</string>
</resources>
```

Для получения значений из строкового массива используется код Java, который приведен в Листинге 1.38.

Листинг 1.38. Получение значений строкового массива из Листинга 1.37 в методах класса Активности

```
Resources res = getResources();
String[] months = res.getStringArray(R.array.months);
```

Разработчикам предоставляется доступ к некоторым строкам, хранящимся в системных ресурсах. Среди этих строк есть системные сообщения об ошибках, но такие строки как *Ok* или *Cancel* удобно использовать во многих приложениях. Чтобы использовать системные строки, нужно перед идентификатором строкового системного ресурса поставить префикс "**android:**". В Листинге 1.39 приведен пример использования строки из системных ресурсов.

Листинг 1.39. Использование строки Cancel из системных ресурсов

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

Внимание! Примеры из Листингов 1.31–1.39 не включены в файлы исходного кода, которые прилагаются к данному уроку.

Теперь рассмотрим, каким образом использование строковых ресурсов дает возможность реализовывать Локализацию приложения. Что такое «Локализация программного обеспечения»? Сайт [wikipedia.org](https://en.wikipedia.org) дает следующее определение этому понятию:

«Локализация программного обеспечения — процесс адаптации программного обеспечения к культуре какой-либо страны. Как частность — перевод пользовательского интерфейса, документации и сопутствующих файлов программного обеспечения с одного языка на другой.»

То есть, локализация программного обеспечения предназначена для того, чтобы пользователи других стран смогли полноценно, без переводчиков, использовать программное обеспечение, которое было разработано не в их стране. Чтобы реализовать такую удобную возможность минимальными усилиями, Google предоставляет разработчикам Android приложений специальный инструмент, с которым мы сейчас и будем знакомиться. Реализация локализации приложения непосредственно осуществляется с использованием строковых ресурсов. В ресурсах необходимо создать новую папку с именем values-XX и разместить там файлы со строковыми ресурсами из папки values. XX — это двухсимвольный идентификатор языка (список поддерживаемых значений для языков приведен в Листинге 1.40). После этого строки в скопированном файле нужно перевести на соответствующий язык.

Как добавлять в модуль новый каталог для ресурсов уже обсуждалось в данном уроке выше (см. Рис 1.1): нужно кликнуть правой кнопкой мыши по узлу **res** в правой панели Android Studio, и в появившемся контекстном меню

выбрать пункт *New resource directory*. Появится диалоговое окно (см. Рис. 1.24) в котором необходимо установить опцию «Тип ресурса» (*Resource Type*) в значение *values* и для опции «Название каталога» (*Directory name*) ввести название создаваемого каталога.

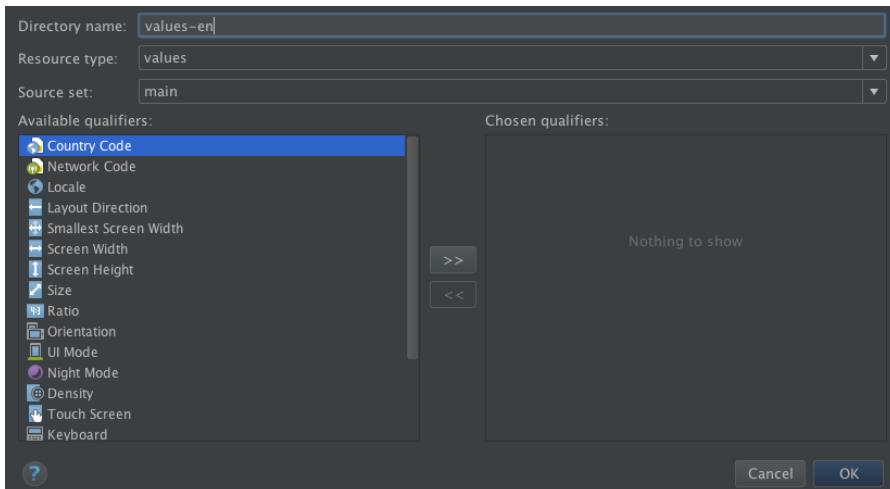


Рис. 1.24. Диалоговое окно с опциями для создания каталога для строковых ресурсов, используемых для локализации приложения

Листинг 1.40. Список кодовых значений для поддерживаемых языков

Английский (en), Арабский (ar), Болгарский (bg),
Венгерский (hu), Вьетнамский (vi), Голландский (nl),
Греческий (el), Датский (da), Иврит (iw),
Индонезийский (in), Испанский (es), Итальянский (it),
Каталонский (ca), Китайский (zh), Корейский (ko),
Латышский (lv), Литовский (lt), Немецкий (de),
Норвежский-Букмол (nb), Польский (pl),
Португальский (pt), Румынский (ro), Русский (ru),
Сербский (sr), Словацкий (sk), Словенский (sl),
Тагальский (tl), Тайский (th), Турецкий (tr),
Украинский (uk), Финский (fi), Французский (fr),
Хинди (hi), Хорватский (hr), Чешский (cs),
Шведский (sv), Японский (ja)

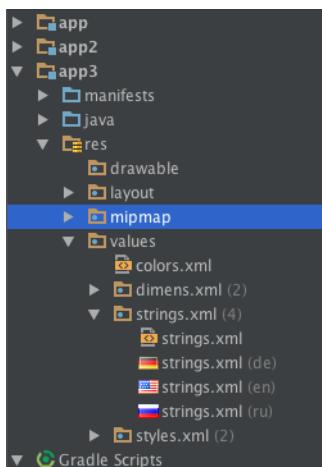


Рис. 1.25. Строковые ресурсы для разных языков

Созданный каталог мы не увидим в Android Studio, но сможем увидеть при помощи файлового менеджера. Для нашего примера (модуль app3 проекта, который прилагается к данному уроку) создадим каталоги /res/values-en, /res/values-de, /res/values-ru. В каждый из этих каталогов, с помощью файлового менеджера, скопируем файл strings.xml. И вот теперь мы сможем увидеть в Android Studio языковые файлы со строковыми ресурсами (см. Рис. 1.25). Как видно из Рис. 1.25, напротив каждого языкового файла strings.xml, из соответствующего каталога, Android Studio отобразила флаг страны этого языка. Исключение составляет только файл /res/values(strings.xml — напротив этого файла нет соответствующего флага, так как этот файл будет использоваться по умолчанию, то есть для извлечения строковых ресурсов в случае, если приложение запустится на устройстве с другим языком, которого нет в наших ресурсах.

В Листинге 1.41 приведено содержимое файлов /res/values/strings.xml, /res/values-en/strings.xml, /res/values-de/strings.xml, /res/values-ru/strings.xml.

Листинг 1.41. Содержимое файлов /res/values/strings.xml, /res/values-en/strings.xml, /res/values-de/strings.xml, /res/values-ru/strings.xml для рассматриваемого примера локализации приложения (модуль app3)

```
File /res/values/strings.xml
-----
<resources>
    <string name="app_name">Lesson_05 /
        Module app3 (Default)</string>
    <string name="app_title">
        Localization (Default)</string>
    <string name="button_title">
        Button (Default)</string>
    <string name="edit_view_title">
        Text Field (Default)</string>
</resources>

File /res/values-en/strings.xml
-----
<resources>
    <string name="app_name">Lesson_05 /
        Module app3 </string>
    <string name="app_title">Localization</string>
    <string name="button_title">Button</string>
    <string name="edit_view_title">Text Field</string>
</resources>

File /res/values-de/strings.xml
-----
<resources>
    <string name="app_name">Lektion_05 /
        Modul app3</string>
```

```
<string name="app_title">
    Lokalisieren von Anwendungen</string>
<string name="button_title">Taste</string>
<string name="edit_view_title">Textbox</string>
</resources>
```

File /res/values-ru/strings.xml

```
<resources>
    <string name="app_name">Урок_05 /
        Модуль app3</string>
    <string name="app_title">
        Локализация приложения </string>
    <string name="button_title">Кнопка</string>
    <string name="edit_view_title">Текстовое поле
        </string>
</resources>
```

Как видно из Листинга 1.41, для языкового файла по умолчанию (/res/values/strings.xml) строки содержат надписи на английском языке. Это значит, что языком по умолчанию в нашем примере является английский язык. Однако, для нашего примера, чтобы иметь возможность визуально различать загруженные строковые значения из файла с языком по умолчанию (/res/values/strings.xml) и из файла с английским языком (/res/values-en/strings.xml), в строковые значения файла по умолчанию, в конец каждой строки добавлена подстрока (*Default*).

Добавим, что Android Studio предоставляет возможность работать с содержимым файлов строковых ресурсов для локализации приложения при помощи удобного

пользовательского интерфейса. На Рис. 1.26 вы можете увидеть ссылку в правом верхнем углу редактора xml файла с надписью голубого цвета *Open editor*.

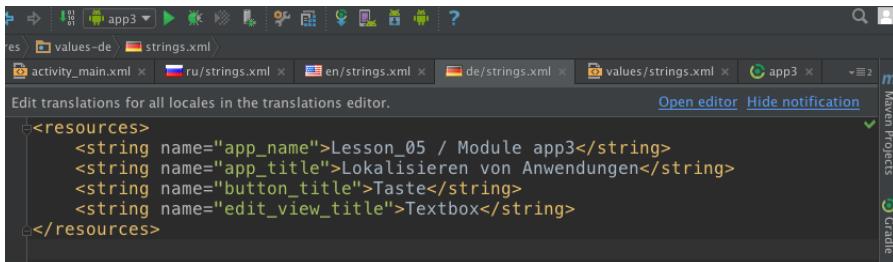


Рис. 1.26. Предлагаемая Android Studio возможность редактирования строковых ресурсов при помощи удобного пользовательского интерфейса (ссылка голубого цвета с надписью *Open editor*).

Если кликнуть на ссылку *Open editor*, то Android Studio отобразит окно редактора, см. Рис. 1.27.

Show only keys needing translations					
Key	Default Value	Untranslatable	German (de)	English (en)	
app_name	Lesson_05 / Module app3 (Default)		Lektion_05 / Modul app3	Lesson_05 / Module app3	Урок_05 / Модуль app3
app_title	Localization (Default)		Lokalisieren von Anwendungen	Localization	Локализация приложения
button_title	Button (Default)		Taste	Button	Кнопка
edit view title	Text Field (Default)		Textbox	Text Field	Текстовое поле

Рис. 1.27. Редактор в Android Studio для локализуемых строковых значений

Как видно из Рис. 1.27, все локализуемые строки представлены в виде таблицы. Каждый язык представлен в этой таблице в виде отдельного столбца, а каждое строковое значение — в виде строки таблицы. Для добавления нового строкового значения необходимо всего лишь добавить новую строку в таблицу редактора. Так же есть возможность указать, какие строковые значения не должны

быть локализированными (столбец *Untranslatable*). Ну, и для добавления нового языка, необходимо кликнуть на пиктограмму с изображением зеленого плюсика (в левом верхнем углу на Рис. 1.27).

Давайте перейдем непосредственно к использованию локализированных строковых ресурсов. Для этой цели в модуле app3 сверстаем xml макет Активности (файл /res/layout/activity_main), содержимое которого приведено в Листинге 1.42.

Листинг 1.42. Макет с версткой внешнего вида для Активности (файл /res/layout/activity_main.xml) рассматриваемого примера по локализации приложения

```
<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"

    tools:context="itstep.com.myapp3.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="12pt"
        android:textColor="#004D40"
        android:text="@string/app_title" />
```

```
<Button  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:textAllCaps="false"  
    android:text="@string/button_title" />  
  
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:text="@string/edit_view_title" />  
  
</LinearLayout>
```

Как видно из Листинга 1.42, в макете Активности находятся следующие виджеты: Текстовое поле `android.widget.TextView`, Кнопка `android.widget.Button`, Редактируемое текстовое поле `android.widget.EditText`. В Листинге 1.42 жирным шрифтом выделены фрагменты кода, в которых перечисленным выше виджетам назначаются строковые значения из ресурсов. Как видите, способ применения локализированных строковых ресурсов ничем не отличается от способа применения обычных строковых ресурсов.

Далее, Android Studio предоставляет возможность наблюдать локализацию приложения на этапе разработки приложения. На Рис. 1.28 изображена возможность выбора языка в Android Studio для отображения строковых значений для режима предпросмотра внешнего вида верстки виджетов на этапе разработке. Для этого необходимо кликнуть на пиктограмму с изображением земного шара.

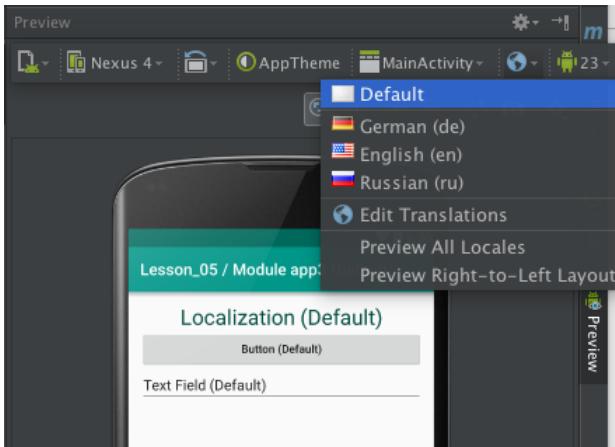


Рис. 1.28. Выбор языка для предпросмотра результатов локализации на этапе разработки в Android Studio

В появившемся выпадающем меню необходимо выбрать интересующий язык, например *German (de)* (немецкий), и Android Studio отобразит макет с использованием строковых значений для надписей выбранного языка (см. Рис. 1.29).

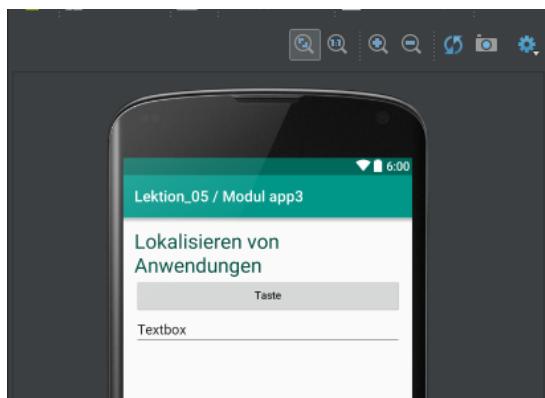


Рис. 1.29. Предпросмотр в Android Studio макета Активности для локализации German (de)

На Рис. 1.30 изображен внешний вид работы примера из модуля *app3* для разных локализаций: английской, немецкой, русской (слева направо).

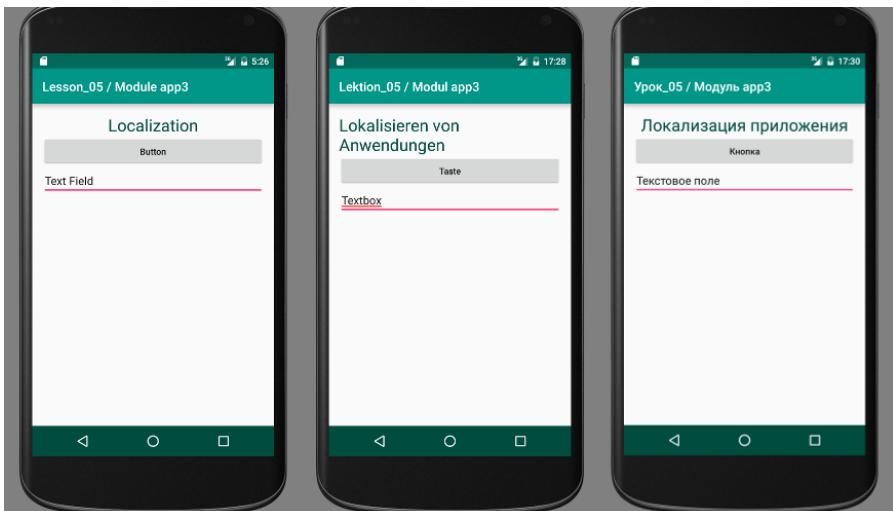


Рис. 1.30. Внешний вид рассматриваемого в разделе примера для разных локализаций: английской, немецкой, русской (слева направо)

Возникает вопрос — а как выбирается конкретная локализация? Так вот, приложение самостоятельно определяет язык, который выбран для Операционной Системы, и на основе этого языка выбирает соответствующую локализацию.

Если для языка Операционной Системы не найдена локализация в приложении (то есть не найден файл ресурсов strings.xml для соответствующего языка), то выбирается локализация по умолчанию (т.е. для отображаемых в приложении строк выбирается файл /res/values/strings.xml).

На этапе отладки и тестирования приложения разработчик может самостоятельно в эмуляторе выбрать соответствующую локализацию. Как это сделать (последовательность действий изображена на Рис. 1.31 и Рис. 1.32).

Необходимо кликнуть на кнопке «Back» (*Назад*), затем необходимо кликнуть на пиктограмме «Приложения» и, в появившемся списке приложений, выбрать приложение Custom Locale (см. Рис. 1.31). В появившемся списке языков (локалей) необходимо выбрать интересующий язык и нажать на кнопку «Select», после этого можно вернуться к тестируемому приложению и убедиться в правильности надписей для выбранной локали.

Существует так же возможность программно выбирать локализацию перед использованием строк из ресурсов. Пример, как это сделать, приведен в Листинге 1.43. Пояснения к Листингу 1.43 в данном уроке даваться не будут, так как код примера достаточно однозначен.

Листинг 1.43. Программный выбор локализации перед использованием строк из ресурсов приложения

```
Locale locale = new Locale("de");
Locale.setDefault(locale);
Configuration config = new Configuration();
config.locale = locale;
this.getBaseContext().getResources().
    updateConfiguration(config, null);
TextView TV = (TextView)
    this.findViewById(R.id.tv1);
TV.setText(R.string.app_title);
```

Обратите внимание, что код из Листинга 1.43 не меняет локализацию для всех строк приложения, а назначает

1. Ресурсы приложения. Типы ресурсов Android-приложения

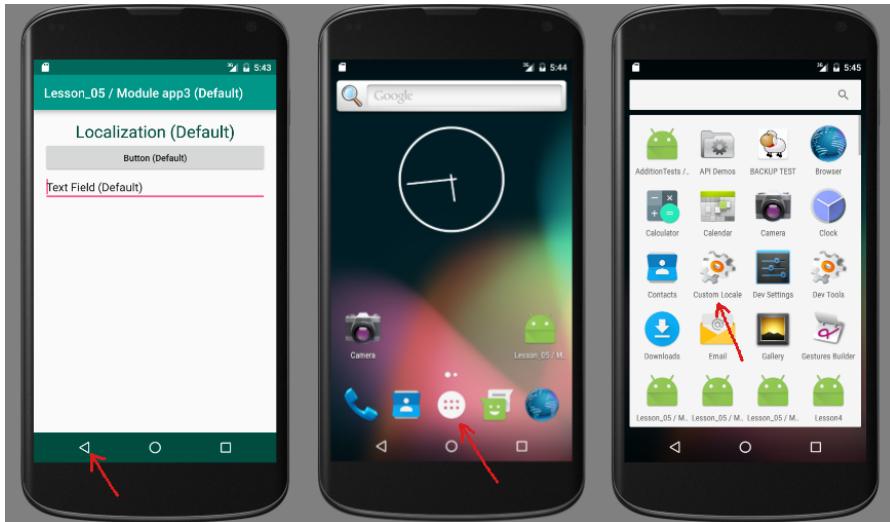


Рис. 1.31 Последовательность действий для смены языка Операционной Системы

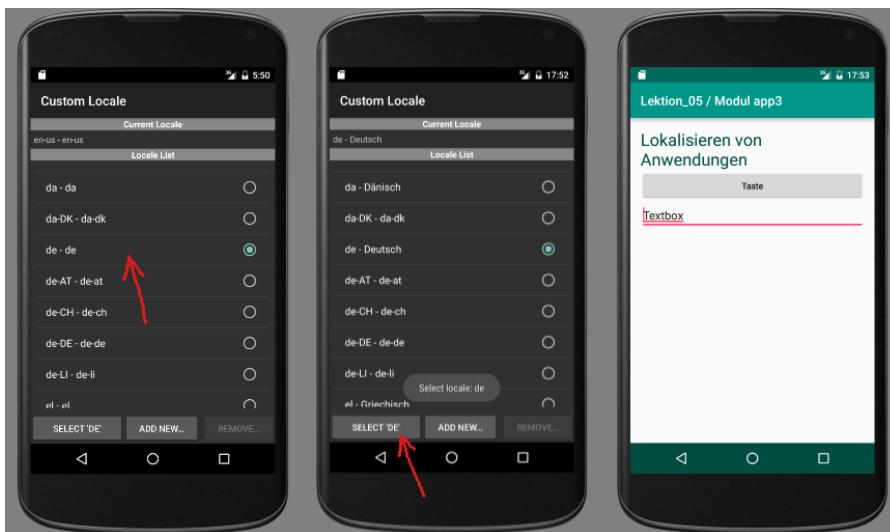


Рис. 1.32 Последовательность действий для смены языка Операционной Системы (Продолжение)

конкретную локализацию перед отображением строк из ресурсов. Так, например, если код из Листинга 1.43 поместить в метод `onCreate()` класса Активности, то получится результат, изображенный на Рис. 1.33, то есть изменится только локализация для строки, помещаемой в текстовое поле `android.widget.TextView`.

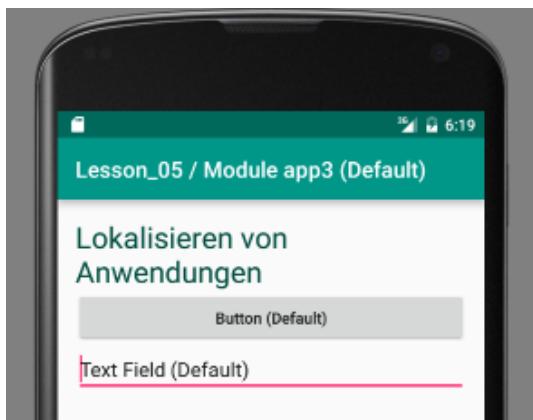


Рис. 1.33. Внешний вид работы примера из Листинга 1.43

Полный исходный код примеров из Листингов текущего раздела можно найти в модуле *app3* проекта, который прилагается к данному уроку.

1.5. Style Resources. Ресурсы стилей для внешнего вида виджетов и приложения

Ресурсы стилей определяют внешний вид элементов пользовательского интерфейса. Стили могут применяться к отдельному виджету, к Активности или ко всему приложению (стиль для приложения назначается в манифесте приложения).

Файлы со стилевыми ресурсами располагаются в каталоге /res/values : res/values/имя_файла.xml. Имя файла может быть любым, по усмотрению разработчика, и в дальнейшем будет использоваться как идентификатор ресурса (resource ID). Для ссылки на стилевой ресурс из файлов xml: **@[package:]style/имя_стиля**.

Синтаксис файла для ресурса стилей приведен в Листинге 1.44.

Листинг 1.44. Синтаксис файла для ресурса стилей

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style
        name="style_name" parent=
            "@[package:]style/style_to_inherit">
        <item name="[package:]style_property_name">
            style_value</item>
    </style>
</resources>
```

Как видно из Листинга 1.44, корневым элементом для xml файла ресурса стилей является элемент **<resources>**. В него вкладываются дочерние элементы **<style>**, каждый из которых описывает один стиль. У элементов **<style>** есть обязательный атрибут **name**, который задает имя стиля. Также есть атрибут **parent**, который используется для указания имени родительского стиля, от которого осуществляется наследование текущий стиль. Дочерними элементами для элемента **<style>** являются элементы **<item>**. Элемент **<item>** описывает единичное свойство стиля. У элемента **<item>** есть обязательный атрибут **name**, который содержит имя для свойства стиля,

которое может быть указано с префиксом пакета (например, `android:textSize`). Значением для элемента `<item>` является значение для свойства стиля.

Для первого примера создадим файл с именем `mystyle.xml` ресурсов в каталоге `/res/values` (т.е. создадим файл `/res/values/mystyle.xml`) и определим в этом файле стиль, как показано в Листинге 1.45.

Листинг 1.45. Создание стиля для текста

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="MyTextStyle">
        <item name="android:textSize">14pt</item>
        <item name="android:textColor">#003366</item>
        <item name="android:layout_gravity">
            center_horizontal</item>
    </style>
</resources>
```

Как видно из Листинга 1.45, созданному стилю назначено имя `MyTextStyle`. Стиль задает значение свойств `android:textSize` равным `14pt`, `android:textColor` равным `#003366` и `android:layout_gravity` равным `center_horizontal`. Применим стиль из Листинга 1.45 к виджету `android.widget.TextView` находящемуся в макете Активности (см. Листинг 1.46).

Листинг 1.46. Применение стиля из Листинга 1.45 к виджету `android.widget.TextView`

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    ...

```

```
    android:orientation="vertical"
    tools:context="itstep.com.myapp4.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        style="@style/MyTextStyle"
        android:text="Hello World!"
    />
</LinearLayout>
```

Внешний вид работы примера из Листингов 1.45 и 1.46 изображен на Рис. 1.34.



Рис. 1.34. Внешний вид работы примеров из Листингов 1.45 и 1.46

Как уже упоминалось выше, существует возможность создавать новые стили на основе уже существующих

стилей, то есть существует возможность для новых стилей наследовать свойства уже существующих стилей. Кстати, значения наследуемых свойств еще можно переопределять. Стили можно наследовать как от пользовательских стилей, так и от стандартных (системных) стилей. Давайте создадим новый стиль, который будет наследовать ранее созданный нами стиль **MyTextStyle** из Листинга 1.45. Создание нового стиля отображено в Листинге 1.47 жирным шрифтом.

Листинг 1.47. Создание в файле /res/values/mystyle.xml нового стиля с именем **MonoFontStyle**, который наследует существующий стиль с именем **MyTextStyle**

```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <style name="MyTextStyle">
        <item name="android:textSize">14pt</item>
        <item name="android:textColor">#003366</item>
        <item name="android:layout_gravity">
            center_horizontal</item>
    </style>

    <style name="MonoFontStyle" parent="MyTextStyle">
        <item name="android:typeface">monospace</item>
        <item name="android:textColor">#008000</item>
    </style>
</resources>
```

Для тестирования нового стиля **MonoFontStyle** создадим в макете Активности еще один виджет **android.widget.TextView**, в котором разместим надпись *Android Forever!*. Внешний вид нового стиля изображен на Рис. 1.35.



Рис. 1.35. Внешний вид стиля с именем MonoFontStyle из Листинга 1.47

Следующий пример продемонстрирует создание нового стиля, который наследует стандартный стиль. Новый стиль имеет имя **RedFont** и также находится в файле `/res/values/mystyle.xml` (Листинг 1.48).

Листинг 1.48. Создание нового стиля с именем RedFont, который является наследником стандартного стиля `TextAppearance.Medium`

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    ...
    <style name="RedFont" parent=
        "@android:style/TextAppearance.Medium">
        <item name="android:layout_width">
            match_parent</item>
```

```

<item name="android:layout_height">
    wrap_content</item>
<item name="android:textColor">#800000</item>
<item name="android:gravity">right</item>

</style>
</resources>

```

Как видно из Листинга 1.48, новый стиль кроме определения свойств выравнивания текста **android:gravity** и цвета текста **android:textColor**, определяет такие свойства, как **android:layout_width** и **android:layout_height**. Это позволяет применять стиль для виджетов, не указывая обязательные атрибуты **android:layout_width** и **android:layout_height** (см. Листинг 1.49).

Листинг 1.49. Применение стиля из Листинга 1.48.

```

<TextView
    style="@style/RedFont"
    android:text="Another style in the wall" />

```

Внешний вид работы примера из Листингов 1.48 и 1.49 изображен на Рис. 1.36.

Теперь сделаем более сложный пример на создание и применение стилей. В этом примере, чтобы продемонстрировать предоставляемую разработчикам широту возможностей, мы будем также использовать полученные ранее в этом уроке знания о других ресурсах приложения: ресурсах цветов, ресурсах Shape Drawable, ресурсах Color State List и других. В этом примере будут созданы стили и ресурсы для виджета **android.widget.Button**. Пусть пример так и называется — «Стилизация кнопок».



Рис. 1.36. Внешний вид работы примера из Листингов 1.48 и 1.49

Вначале откроем файл ресурсов /res/values/colors.xml и добавим туда несколько значений для цветов (см. Листинг 1.50).

Листинг 1.50. Добавленные в файл ресурсов /res/values/colors.xml цвета для рассматриваемого примера «Стилизация кнопок»

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color name="colorPrimaryDark">#303F9F</color>
    <color name="colorAccent">#FF4081</color>

    <color name="btn_normal_gradient_color_start">
        #D81B60</color>
```

```

<color name="btn_normal_gradient_color_end">
    #F48FB1</color>
<color name="btn_pressed_gradient_color_start">
    #8E24AA</color>
<color name="btn_pressed_gradient_color_end">
    #BA68C8</color>
<color name="btn_text_color">#FFFFFF</color>
</resources>

```

Как видно из Листинга 1.50, созданные цвета будут использованы в качестве начальных и конечных цветов для градиентов фона разных состояний кнопок (цвета с именами `btn_normal_gradient_color_start` и `btn_normal_gradient_color_end` — для фона нормального состояния кнопок, а цвета с именами `btn_pressed_gradient_color_start` и `btn_pressed_gradient_color_end` — для фона нажатого состояния кнопок), а так же в качестве цвета текста для кнопок (цвет с именем `btn_text_color`).

Следующим шагом мы создадим два файла ресурсов *Shape Drawable* (`/res/drawable btn_normal.xml` и `/res/drawable btn_pressed.xml`). Содержимое этих файлов приведено в Листингах 1.51 и 1.52 соответственно.

Листинг 1.51. Файл `/res/drawable btn_normal.xml` с ресурсом *Shape Drawable* для фонового изображения кнопок при не нажатом (нормальном) состоянии для примера «Стилизация кнопок»

```

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

```

```
    android:shape="rectangle">

<corners android:radius="14px" />

<gradient
    android:angle="90"
    android:endColor=
        "@color/btn_normal_gradient_color_end"
    android:startColor=
        "@color/btn_normal_gradient_color_start"
    android:type="linear"
/>

<padding
    android:top="10px"
    android:bottom="10px"
    android:left="30px"
    android:right="30px"
/>
</shape>
```

Листинг 1.52. Файл /res/drawable btn_pressed.xml с ресурсом Shape Drawable для фонового изображения кнопок при нажатом состоянии для примера «Стилизация кнопок»

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

<corners android:radius="14px" />

<gradient
    android:angle="270"
    android:endColor=
        "@color/btn_pressed_gradient_color_end"
```

```

        android:startColor=
            "@color/btn_pressed_gradient_color_start"
        android:type="linear"
    />

<padding
    android:top="10px"
    android:bottom="10px"
    android:left="30px"
    android:right="30px"
    />
</shape>
```

Как видно из Листингов 1.51 и 1.52, файлы *Shape Drawable* описывают фигуры, которые будут использоваться для фонового изображения кнопок в рассматриваемом примере «Стилизация кнопок».

Далее, создадим файл ресурсов *Color State List* (/res/drawable/btn_color_list.xml). Содержимое файла приведено в Листинге 1.53. Его задача описать внешний вид фонового изображения кнопок для разных состояний: для нормального и для нажатого состояния.

Листинг 1.53. Файл ресурсов Color State List (/res/drawable/btn_color_list.xml), который описывает внешний вид фона кнопок для разных состояний для примера «Стилизация кнопок»

```

<?xml version="1.0" encoding="utf-8" ?>
<selector xmlns:android=
    "http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/btn_pressed"
          android:state_pressed="true" />
    <item android:drawable="@drawable/btn_normal" />
</selector>
```

Следующим шагом будет создание стиля для кнопок в файле ресурсов /res/values/mystyle.xml. Стиль назовем **ButtonStyle**. Xml код этого стиля приведен в Листинге 1.54.

Листинг 1.54. Стиль ButtonStyle из файла ресурсов /res/style/mystyle.xml для рассматриваемого примера «Стилизация кнопок»

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    ...

<style name="ButtonStyle">
    <item name="android:layout_width">
        wrap_content</item>
    <item name="android:layout_height">
        wrap_content</item>
    <item name="android:layout_gravity">
        center_horizontal</item>
    <item name="android:background">@drawable/
        button_color_list</item>
    <item name="android:layout_margin">5dp</item>
    <item name="android:textColor">@color/
        btn_text_color</item>
    <item name="android:textSize">9pt</item>
    <item name="android:textAllCaps">false</item>
</style>

</resources>
```

И последнее, в макете Активности /res/layout/activity_main.xml создадим два виджета **android.widget.Button** и применим к ним стиль из Листинга 1.54. Создание кнопок в макете Активности показано в Листинге 1.55.

Листинг 1.55. Создание кнопок android.widget.Button для применения к ним стиля ButtonStyle в рассматриваемом примере «Стилизация кнопок»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    ...
    <Button
        style="@style/ButtonStyle" android:text=
                    "Button One" />
    <Button
        style="@style/ButtonStyle" android:text=
                    "Button Two" />
</LinearLayout>
```

Внешний вид примера «Стилизация кнопок» из Листингов 1.50–1.55 изображена на Рис. 1.37.

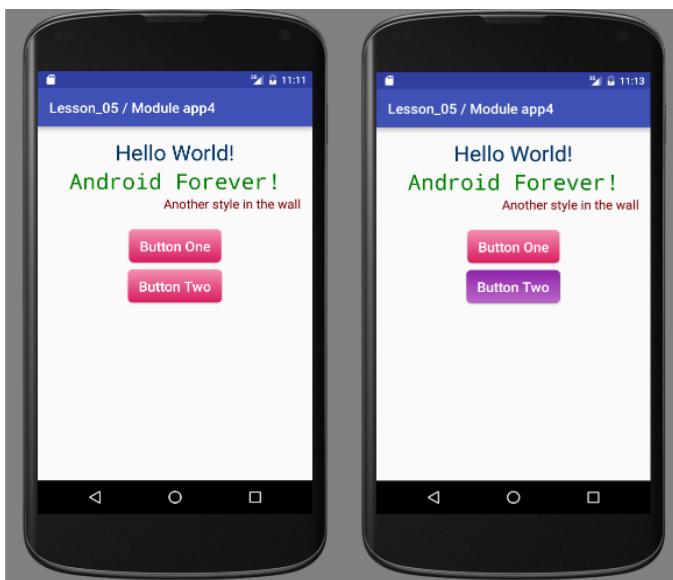


Рис. 1.37. Внешний вид примера «Стилизация кнопок» из Листингов 1.50–1.55

На Рис. 1.37 слева изображены кнопки `android.widget.Button`, которые находятся в нормальном состоянии, а справа — кнопка, с надписью «*Button Two*», находится в нажатом состоянии, а кнопка, с надписью «*Button One*», находится в нормальном состоянии. При нажатии на кнопку внешний вид кнопки меняется в соответствии с Листингами 1.50–1.54.

Также обратите внимание, что описав один раз набор файлов ресурсов и стилей для виджета, применять этот набор можно сразу к нескольким виджетам, что делается очень легко и лаконично, как показано в Листинге 1.55 (выделено жирным шрифтом).

Исходные коды примера «Стилизация кнопок» находятся в модуле *app4*, среди файлов с исходными кодами, которые прилагаются к данному уроку.

Можно создавать стили не только для виджетов или Активности, но и для приложения. При создании модуля, Android Studio создает файл `/res/values/styles.xml`, в котором создается стиль с именем `AppTheme`, который наследуется от стандартного стиля (в модулях проекта, прилагаемого к этому уроку, наследование происходит от стандартного стиля `Theme.AppCompat.Light.DarkActionBar`). Кстати, в файл со стилевыми ресурсами `/res/values/styles.xml` можно добавлять и свои стили — не обязательно для своих стилей создавать новые файлы ресурсов. Так вот, разработчикам предоставляется возможность разнообразить внешний вид приложения путем изменения стилей. Давайте посмотрим на содержимое файла `/res/values/styles.xml` который создает Android Studio при создании модуля (см. Листинг 1.56).

Листинг 1.56. Содержимое файла ресурсов /res/values/styles.xml после создания модуля в проекте Android Studio

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent=
        "Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">
            @color/colorPrimary</item>
        <item name="colorPrimaryDark">
            @color/colorPrimaryDark</item>
        <item name="colorAccent">
            @color/colorAccent</item>
    </style>
</resources>
```

Как видно из Листинга 1.56, в файле /res/values/styles.xml создается стиль для приложения с именем **AppTheme**, который наследуется от стандартного стиля **Theme.AppCompat.Light.DarkActionBar**. В этом стиле назначаются цвета для панелей состояния, навигации, панели инструментов. Этот стиль назначается приложению в Манифесте приложения. В Листинге 1.57 приведен Манифест приложения и жирным шрифтом выделен фрагмент кода в котором приложению назначается стиль **AppTheme**.

Листинг 1.57. Демонстрация назначения стиля из Листинга 1.56 для приложения в Манифесте приложения

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="itstep.com.myapp4"
    xmlns:android=
        "http://schemas.android.com/apk/res/android">
```

```

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

</manifest>

```

Давайте создадим свои стили для внешнего вида приложения. В файле /res/values/styles.xml создадим стиль с именем **MyTheme**, который сделаем наследником от стиля **Theme.AppCompat.Light.DarkActionBar**, и изменим значение свойства цвета **colorPrimaryDark**, как это показано в Листинге 1.58.

Листинг 1.58. Создание своего стиля для внешнего вида приложения

```

<resources>
    <style name="AppTheme" parent=
        "Theme.AppCompat.Light.DarkActionBar">
        ...
    </style>
    <style name="MyTheme" parent=
        "Theme.AppCompat.Light.DarkActionBar">

```

```
<item name="colorPrimary">  
    @color/colorPrimary</item>  
<item name="colorPrimaryDark">#827717</item>  
<item name="colorAccent">@color/colorAccent  
    </item>  
</style>  
  
</resources>
```

Назначим стиль с именем **MyTheme** из Листинга 1.58 приложению в Манифесте приложения, как это показано в Листинге 1.57. Запустим наше приложение в эмуляторе и увидим результат, который изображен на Рис. 1.38 слева — панель состояния отображается цветом хаки.

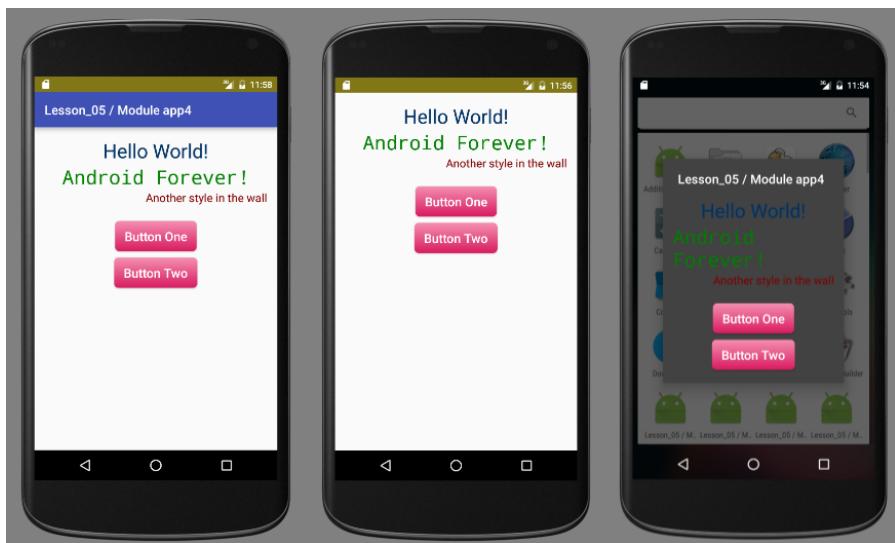


Рис. 1.38. Применение разных стилей для приложения

Далее, можно выбрать тему без **ActionBar**. Для этого создадим еще один стиль с именем **MyTheme1** в файле `/res`/

values/styles.xml. Для этого необходимо указать в качестве родительского стиля какой-нибудь стиль без **ActionBar** (как правило, в названии таких стилей явно указывается слово **NoActionBar**). Стиль **MyTheme1** приведен в Листинге 1.59., внешний вид применения стиля изображен на Рис. 1.38 в центре.

Листинг 1.59. Стили для приложений MyTheme1 и MyTheme2

```
<resources>
    ..
    <style name="MyTheme1" parent=
        "Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">
            @color/colorPrimary</item>
        <item name="colorPrimaryDark">#827717</item>
        <item name="colorAccent">@color/colorAccent
            </item>
    </style>

    <style name="MyTheme2" parent=
        "Theme.AppCompat.Light.Dialog">
        <item name="colorPrimary">
            @color/colorPrimary</item>
        <item name="colorPrimaryDark">#827717</item>
        <item name="colorAccent">@color/colorAccent
            </item>
    </style>
</resources>
```

Также в Листинге 1.59 приведен пример создания стиля (имя **MyTheme2**) на основе стиля Диалогового окна.

Внешний вид применения стиля **MyTheme2** из Листинга 1.59 изображен на Рис. 1.38 справа.

На самом деле стили из Листингов 1.58 и 1.59 принято называть не стилями, а темами. **Тема** — это стиль, который применяется ко всей Активности или приложению, а не к отдельному компоненту или виджету приложения. Тема имеет свои атрибуты и свою область применения. В Листинге 1.57 приведен пример (выделено жирным шрифтом) назначения стиля всему приложению. Если необходимо, чтобы тема относилась не ко всему приложению, а к конкретной Активности, то атрибут **android:theme** нужно добавить в тег **<activity>** в Манифесте приложения:

```
<activity android:theme="@style/CustomTheme">
```

Нет необходимости создавать свою тему «с нуля». У Android есть множество встроенных тем — поэтому нужно всего лишь подобрать наиболее подходящую тему для вашего приложения. В выборе тем необходимо помнить о такой важной детали, как диапазон поддерживаемых версий.

Дело в том, что в 2014 году компания Google представила набор правил и рекомендаций к созданию дизайну программного обеспечения для Операционной Системы Android. Этот набор называется **Material Design** и представляет собой комплексную концепцию создания визуальных, движущихся и интерактивных элементов для различных платформ и устройств.

Material Design дает возможность разработчикам приложений создавать приложения, имеющие удобный,

наглядный, понятный дизайн. Но, поддержка *Material Design* для Android приложений начинается с версии API 21. Поэтому, если вы создали проект и указали, что минимальной версией для вашего приложения должна быть версия API 15, то использовать возможности *Material Design* у вас просто так не получится (например, при попытке назначить тему *Material Design* для приложения или Активности, у вас произойдет исключительная ситуация, с рекомендациями использовать темы **AppCompat** — для совместимости ранних и поздних версий API с внешним видом *Material Design*).

К теме *Material Design* мы еще вернемся в наших уроках. Пока только приведем в качестве примера названия нескольких тем *Material Design* для приложений:

- `@android:style/Theme.Material` — темная тема;
- `@android:style/Theme.Material.Light` — светлая тема;
- `@android:style/Theme.Material.Light.DarkActionBar` — светлая тема с темным заголовком;
- `@android:style/Theme.Material.Light.NoActionBar` — светлая тема, без Action Bar.

Для тем *Material Design* были разработаны новые атрибуты (см. Рис. 1.39), с помощью которых можно настраивать цвета при наследовании темы от темы *Material Design*:

- `android:colorPrimary` — основной цвет для интерфейса программы — панель, кнопки и т.д.;
- `android:colorPrimaryDark` — цвет для системных элементов — строка состояния;

- **android:colorAccent** — цвет по умолчанию, для компонентов, которые находятся в фокусе или активны;
- **android:colorControlNormal** — цвет для неактивных компонентов;
- **android:colorControlActivated** — цвет для активных компонентов;
- **android:colorControlHighlight** — цвет для нажатых элементов интерфейса.

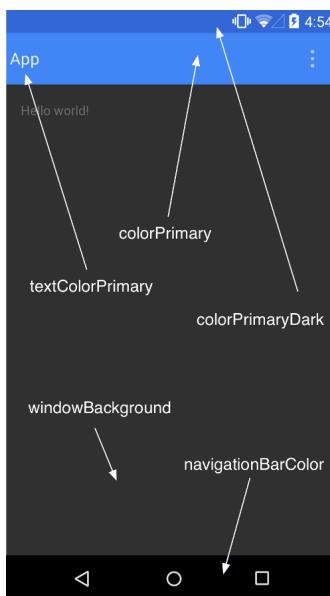


Рис. 1.39. Предназначение атрибутов настройки цветов для тем Material Design

Настраивание цветов для приложения позволяет придать вашему приложению неповторимый стиль. Для этого необходимо в файле `/res/values/style.xml` назначить цветовым атрибутам (размещаются в элементе `<item>`) значения цветов (см. Листинг 1.60).

Листинг 1.60. Настройка темы Material Design

```
<resources>
    ...
    <style name="AppTheme"
        parent="android:Theme.Material">
        <item name="android:colorPrimary">
            @color/primary</item>
        <item name="android:colorPrimaryDark">
            @color/primary_dark</item>
        <item name="android:colorAccent">
            @color/accent</item>
    </style>
    ...
</resources>
```

Настройка цветов происходит по определенным правилам. На сайте <https://material.io/guidelines/style/color.html#color-color-palette> есть таблицы цветов (см. Рис. 1.40).

Light Blue		Cyan		Teal	
500	#03A9F4	500	#00BCD4	500	#009688
50	#E1F5FE	50	#EOF7FA	50	#E0F2F1
100	#B3E5FC	100	#B2EBF2	100	#B2DFDB
200	#81D4FA	200	#80DEEA	200	#80CBC4
300	#4FC3F7	300	#4DD0E1	300	#4DB6AC
400	#29B6F6	400	#26C6DA	400	#26A69A
500	#03A9F4	500	#00BCD4	500	#009688
600	#0398E5	600	#00ACC1	600	#00897B
700	#0288D1	700	#0097A7	700	#00796B
800	#0277BD	800	#00838F	800	#00695C
900	#01579B	900	#006064	900	#004D40
A100	#80D8FF	A100	#84FFFF	A100	#A7FFEB
A200	#40C4FF	A200	#18FFFF	A200	#64FFDA
A400	#00B0FF	A400	#00E5FF	A400	#1DE9B6
A700	#0091EA	A700	#0088D4	A700	#00BFAS

Рис. 1.40. Фрагмент страницы сайта <https://material.io/guidelines/style/color.html#color-color-palette> с таблицами цветов

Основным цветом (для атрибута `android:colorPrimary`) считается цвет под номером 500. Этот цвет должен использоваться в качестве заголовка (для `ActionBar` или для `Toolbar`).

Для панели состояния, которая находится выше заголовка (`ActionBar` или `Toolbar`) приложения, нужно использовать цвет со значением 700 (атрибут `android:colorPrimaryDark`). Поскольку (как упоминалось выше), поддержка *Material Design* появилась в Android, начиная с версии API 21, то у нас могут возникнуть проблемы несовместимости, при использовании тем *Material Desing* для проектов с минимальной версией API менее 21.

Поэтому, для таких приложений, рекомендуется использовать темы `AppCompat`, которые также предоставляют возможность настраивать цвета. Рекомендуем в данном случае воспользоваться специальным интерактивным редактором, который встроен в `Android Studio`. Вызвать этот редактор можно следующим образом: откройте файл `/res/values/style.xml` для редактирования, и в правом верхнем углу вы увидите ссылку синего цвета с надписью *Open editor* (см. Рис. 1.41).

Кликнув на ссылке *Open editor* появится редактор, работа с которым интуитивно понята (см. Рис. 1.42). Не забудьте только вначале выбрать модуль (опция *Module*) вашего проекта, для которого вы хотите настраивать цвета, иначе настройка цветов может быть осуществлена для другого модуля вашего проекта, о чем вы с досадой заметите позже. Также в опции *Theme* выберите тему (из списка тем, которые вы создали в файле `/res/values/style.xml`) для которой вы будете менять цвета.

Давайте попробуем придать примеру из модуля *app4* неповторимый внешний вид, с помощью рекомендаций *Material Design*.

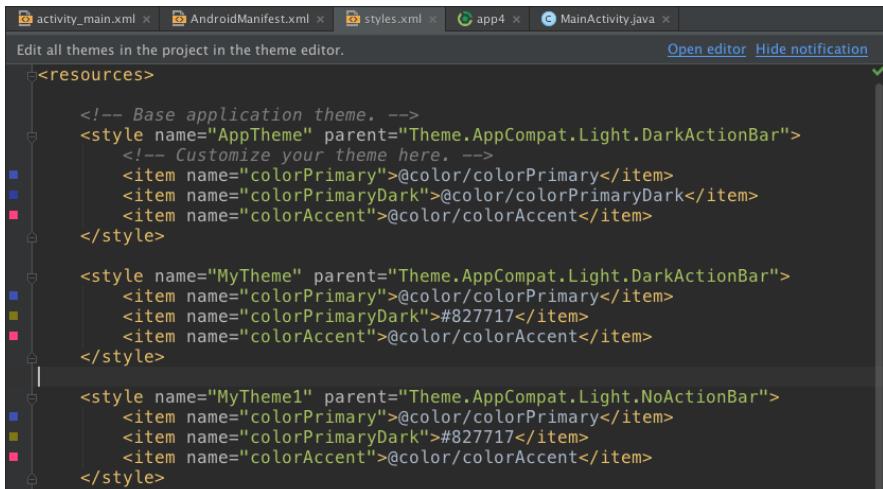


Рис. 1.41. Ссылка на редактор для настройки цветов для темы приложения в Android Studio

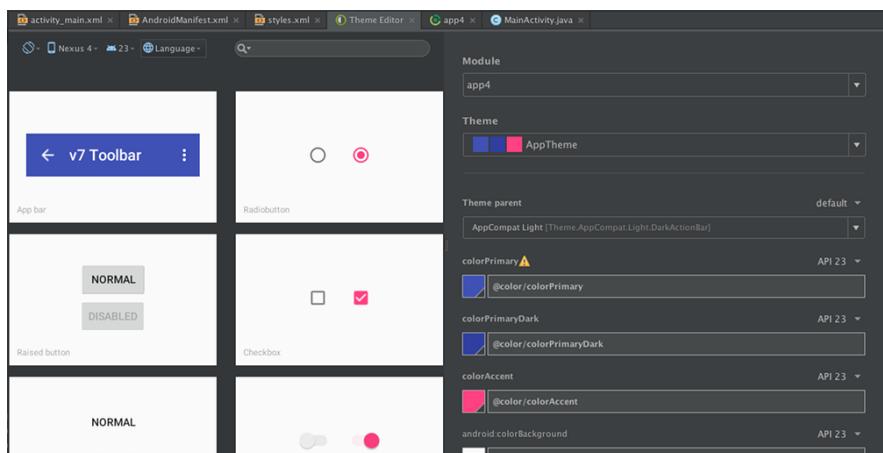


Рис. 1.42. Редактор тем (Theme Editor) в Android Studio для настройки цветов тем

С помощью сайта [Material Design](#) выберем цветовую гамму *Light Blue* (см. Рис. 1.40), и с помощью редактора тем (Рис. 1.42) изменим цвета темы **AppTheme** для заголовка приложения (назначим цвет 500), панели состояния (назначим цвет 700) и панели навигации (атрибут **android:navigationBarColor**, — назначим цвет 900). Результат можно увидеть на Рис. 1.43.



Рис. 1.43. Настройка цветов для темы приложения AppTheme модуля app4

К *Material Design* мы еще вернемся в наших уроках, но вы можете уже начинать знакомится с этой концепцией самостоятельно (<https://material.io/guidelines>), так как любой профессиональный разработчик приложений должен владеть *Material Design* для создания современных Android приложений.

Все примеры из данного раздела можно найти в модуле *app4* проекта с исходными кодами, прилагаемого к данному уроку.

1.6. MipMap ресурсы

Наверняка, вы многократно обращали внимание на папку */res/mipmap*, которая присутствует в каждом модуле вашего проекта. И наверняка, вы уже заглядывали в эту папку и видели, что там располагается изображение для вашего разрабатываемого приложения. Это изображение по умолчанию, которое автоматически создает Android Studio при создании нового модуля. И как вы, наверное, догадались, это изображение создается в разных размерах, под разные разрешения экранов устройств. В этой главе мы более детально познакомимся с этим типом ресурсов.

Термин *MipMap* происходит от термина «MIP-текстурирование» (англ. *MIP mapping*). Сайт [wikipedia.org](https://en.wikipedia.org) дает следующее определения понятия «MIP-текстурирования»:

*«МIP-текстурирование — метод текстурирования, использующий несколько копий одной текстуры с разной детализацией. Название происходит от лат. *multum in parvo* — "много в малом". »*

Другими словами, *MipMap* метод использует разные копии одного и того же изображения для отображения на экранах с разным разрешением с целью, чтобы отображаемое изображение имело приблизительно одинаковую детализацию (изображение лучше всего выглядит, когда детализация текстуры близка к разрешению экрана).

Поясним: предположим, изображение размером 32dp по ширине и 32dp по высоте занимает N миллиметров на экране с разрешением 1024×768 пикселей. Чтобы это же изображение занимало тот же размер на экране с вдвое большим разрешением, необходимо растянуть (увеличить масштаб) изображения в два раза. Но двухкратное увеличение масштаба растрового изображения приводит к существенной потере качества этого изображения.

Решением этой проблемы является метод MIP-текстурирования. Согласно этому методу, приложению необходимо иметь копии одного и того же изображения с различной детализацией (с различным размером) для нескольких разрешений экрана. Для Android приложений эти копии изображений размещаются в ресурсах приложения (каталог /res/mipmap). В зависимости от разрешения экрана выбирается соответствующее этому разрешению изображение. Таким образом достигается сохранение качества и размеров изображения на экранах с различным разрешением.

Давайте подробнее рассмотрим изображения из каталога /res/mipmap, которые автоматически создает для каждого нового модуля Android Studio (см. Рис. 1.44). Как видно из Рис. 1.44, после создания модуля в каталоге /res/mipmap находится изображение ic_launcher.png (пиктограмма с узнаваемым изображением зеленого робота-androида). Таких изображений 5 — для разных разрешений экранов. Разрешения экранов указаны в круглых скобках после названия изображения (см. Рис. 1.44): **hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi**.

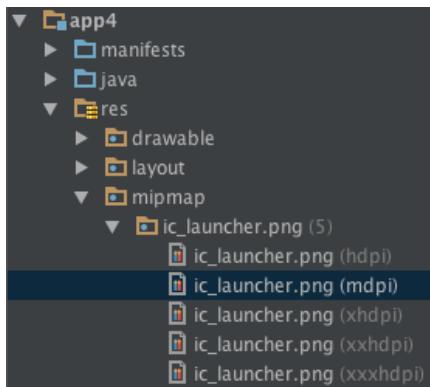


Рис. 1.44. Каталог /res/mipmap и его содержимое в модуле проекта Android Studio

Это изображение (ic_launcher.png), как вы уже знаете, является пиктограммой нашего разрабатываемого приложения. Мы можем самостоятельно добавлять другие свои изображения с разным размером (детализацией) в каталог ресурсов /res/mipmap (можете даже заменить пиктограмму для вашего приложения). При этом Android приложение автоматически будет использовать соответствующее текущему разрешению экрана изображение — разработчику всего лишь необходимо разместить в каталоге /res/mipmap набор изображений для всех пяти разрешений экранов.

На самом деле структура каталога /res/mipmap, которая отображается в Android Studio отличается от структуры файлового каталога. На Рис. 1.45 изображен скриншот файлового менеджера, показывающий файловую структуру каталогов для каталога ресурсов /res/mipmap.

Как видно из Рис. 1.45, на диске в каталоге res располагается не каталог mipmap, а пять подкаталогов:

mipmap-hdpi, mipmap-mdpi, mipmap-xhdpi, mipmap-xxhdpi, mipmap-xxxhdpi. В каждом из этих каталогов находится файл с растровым изображением ic_launcher.png соответствующего размера. В Листинге 1.61 приведена полная структура каталогов и файлов для каталога ресурсов /res/mipmap (см. Рис. 1.44), с указанием рекомендуемых размеров для изображений под каждое разрешение экрана.



Рис. 1.45. Файловая структура каталогов для каталога ресурсов /res/mipmap

Листинг 1.61. Файловая структура каталога /res/mipmap, с указанием рекомендуемых размеров для изображений

```
/res
  /mipmap-mdpi/ic_launcher.png      (48x48 pixels)
  /mipmap-hdpi/ic_launcher.png     (72x72 pixels)
  /mipmap-xhdpi/ic_launcher.png    (98x98 pixels)
  /mipmap-xxhdpi/ic_launcher.png   (144x144 pixels)
  /mipmap-xxxhdpi/ic_launcher.png  (192x192 pixels)
```

Теперь узнаем, что же это за обозначения: **mdpi**, **dpi**, **xhdpi** и так далее. Это обозначения для плотности экрана. Причем аббревиатура **dpi** означает *dots per inch* (количество точек на дюйм). А стоящая перед **dpi** буква — величину плотности.

Основные плотности экрана: **ldpi** (*low* — низкий), **mdpi** (*medium* — средний), **hdpi** (*high* — высокий), **xhdpi** (*eXtra hight* — дополнительный высокий), **xxhdpi** (*eXtra hight* — дополнительный дополнительный высокий),

xxxhdpi (*eXtra eXtra eXtra hight* — дополнительный дополнительный дополнительный высокий).

Базовой является плотность **mdpi**, когда $1\text{px} = 1\text{dp}$. Остальные являются множителями: **ldpi** (0.75x), **mdpi** (1x), **hdpi** (1.5x), **xhdpi** (2.0x), **xxhdpi** (3.0x), **xxxhdpi** (4.0x).

Плотность **mdpi** соответствует разрешению экрана 320×480 пикселей. Если считать плотность **mdpi** за единицу (базовая плотность), то можно, используя множители указанные выше, рассчитать разрешения для других типов. Получим: **hdpi** — 480×720 пикселей, **xhdpi** — 640×960 пикселей, **xxhdpi** — 960×1440 пикселей, **xxxhdpi** — 1280×1920 пикселей.

Размер устройства в **dpi** (или **dp**) рассчитывается по формуле: разрешение экрана делится на множитель, указанный выше. Соответственно, устройство с экраном 1080×1920 соответствует типу **xxhdpi** — $360 \times 640\text{dp}$ ($1080 / 3 = 360$; $1920 / 3 = 640\text{dp}$).

Итак, разобравшись с плотностями экранов, давайте научимся применять изображения из **MipMap** ресурсов. Во-первых, чтобы применить пиктограмму для приложения, необходимо в Манифесте приложения (файл `/manifests/AndroidManifests.xml`) назначить приложению пиктограмму, как показано в Листинге 1.62.

Листинг 1.62. Назначение приложению пиктограммы из **MipMap** ресурсов в Манифесте приложения

```
...
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
```

```
    android:label="@string/app_name"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
...

```

Компания Google для разработчиков Android приложений предлагает целые наборы иконок и пиктограмм, которые доступны по ссылке: <https://material.io/icons/>. Изображения упорядочены по тематикам. Скриншот части страницы с этого сайта изображен на Рис. 1.46.

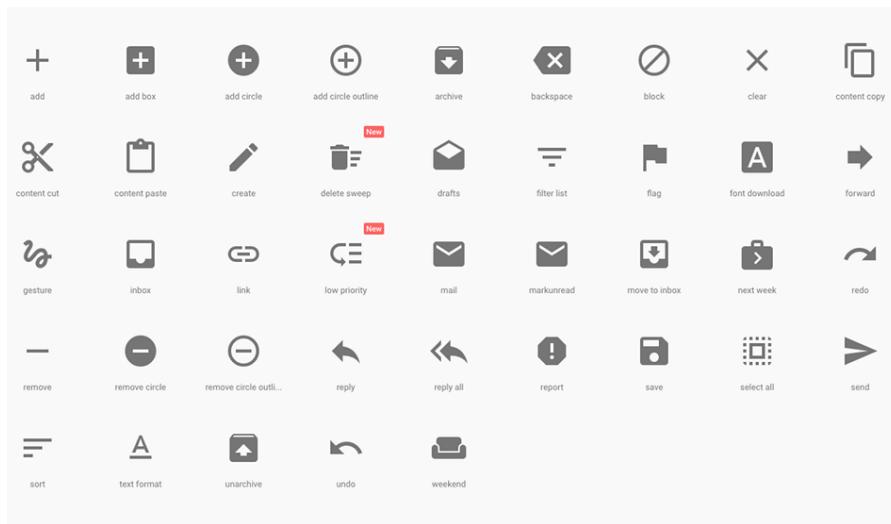


Рис. 1.46. Фрагмент страницы <http://material.io/icons>

Если кликнуть по понравившемуся изображению, то снизу окна браузера всплывет панель, с помощью которой можно указать цвет изображения (светлый или темный), размер изображения в **dp**, формат изображения **png** или **svg** (см. Рис. 1.47).

1. Ресурсы приложения. Типы ресурсов Android-приложения

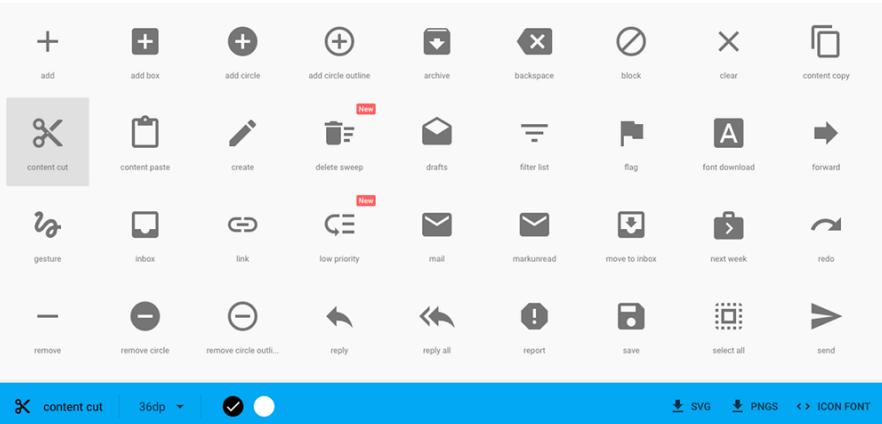


Рис. 1.47. Меню настроек для скачиваемого изображения на странице <http://material.io/icons>

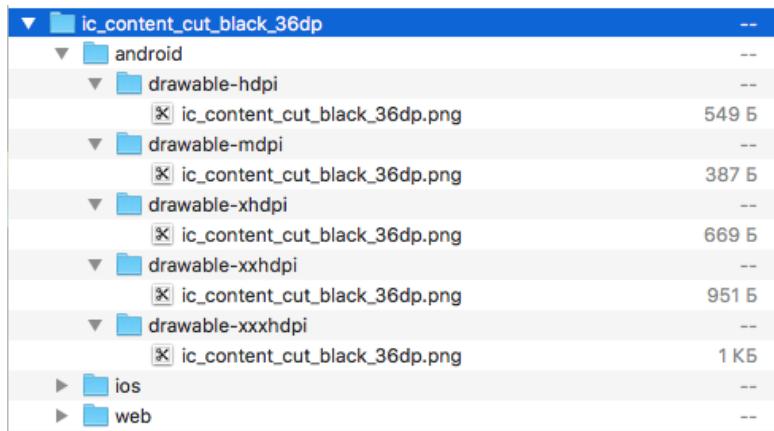


Рис. 1.48. Загруженный с сайта <http://material.io/icons> архив с изображением для разных плотностей экранов

Указав нужные опции будет скачан zip архив, распаковав который мы увидим (см. Рис. 1.48), что изображение, во-первых, можно использовать не только для Android приложений, но и для iOS и Web приложений,

а во-вторых, что самое главное, изображение в архиве представлено для разных плотностей экранов!

Скопируем изображения из каталогов **drawable** распакованного архива в соответствующие каталоги **mipmap** модуля нашего проекта (*app4*). Мы увидим, что Android Studio «подхватит» эти изображения и отобразит в структуре модуля нашего проекта (см. Рис. 1.49).

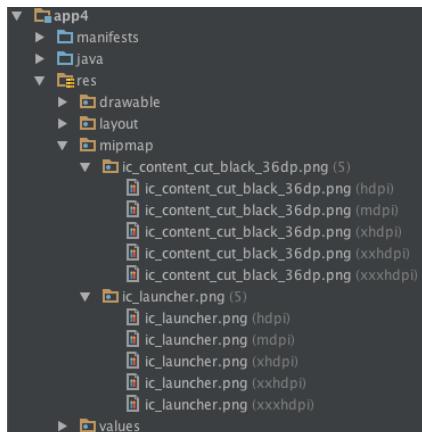


Рис. 1.49. Отображение в Android Studio добавленных в каталог ресурсов /res/mipmap изображений для разных плотностей экранов

Теперь давайте применим загруженные изображения в нашем приложении. На этот раз отобразим изображения с помощью программного кода Java (как отображать с помощью xml показано в Листинге 1.62). Для этой цели добавим в макет Активности нашего приложения два виджета **android.widget.ImageView** (см. Листинг 1.63). Виджетам присвоим идентификаторы **@+id/imgOne** и **@+id/imgTwo** соответственно.

Листинг 1.63. Виджеты для отображения иконок из каталога ресурсов /res/mipmap рассматриваемого примера

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imgOne" />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imgTwo" />
```

В методе **onCreate()** класса Активности назначим виджетам из Листинга 1.63 иконки из каталога ресурсов /res/mipmap.

Листинг 1.63. Назначение виджетам android.widget.ImageView иконок из каталога ресурсов /res/mipmap

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    ImageView imgOne = (ImageView)
        this.findViewById(R.id.imgOne);
    imgOne.setBackgroundDrawable(R.drawable.ic_launcher);
    ImageView imgTwo = (ImageView)
        this.findViewById(R.id.imgTwo);
    imgTwo.setBackgroundDrawable
        (R.drawable.ic_content_cut_black_36dp);
}
```

Как уже говорилось выше, приложение автоматически выберет изображение из каталога /res/mipmap наиболее

соответствующее плотности экрана устройства, на котором исполняется приложение.

Внешний вид работы примера из Листинга 1.63 изображен на Рис. 1.50.



Рис. 1.50. Внешний вид работы примера из Листинга 1.63 (иконка Android и ножницы)

1.7. Dimension Resources. Ресурсы для хранения величин размеров

В каталоге ресурсов /res/values еще находится файл /res/values/dimens.xml. Этот файл предназначен для хранения величин размеров, которые используются для верстки макетов внешнего вида Активности и других элементов пользовательского интерфейса. Содержимое этого файла приведено в Листинге 1.64.

Листинг 1.64. Содержимое файла /res/values/dimens.xml

```
<resources>
    <!-- Default screen margins, per the Android
        Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
</resources>
```

Как видно из Листинга 1.64, элемент `<dimen>` описывает одну величину размера. У элемента `<dimen>` есть обязательный атрибут `name`, который задает название для величины, чтобы впоследствии использовать эту величину, ссылаясь на нее через это название: в xml коде при помощи конструкции `@dimen/название`, в Java коде при помощи конструкции `R.dimen.название`. Пример использования величин из файла ресурсов `/res/values/dimes.xml` в макете внешнего вида Активности (`/res/layout/activity_main.xml`) приведен в Листинге 1.65.

Листинг 1.65. Применение величин размеров из файла ресурсов /res/values/dimens.xml
в макете Активности /res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom=
        "@dimen/activity_vertical_margin"
```

```
    android:paddingLeft=
        "@dimen/activity_horizontal_margin"
    android:paddingRight=
        "@dimen/activity_horizontal_margin"
    android:paddingTop=
        "@dimen/activity_vertical_margin"
    android:orientation="vertical"
    tools:context="itstep.com.myapp4.MainActivity">
    ...

```

Использование этого типа ресурсов является несложным и понятным, поэтому мы с вами переходим к рассмотрению материала следующего раздела.

1.8. Каталоги ресурсов raw и assets

Данные типы ресурсов же использовались нами в предыдущих уроках, однако, все равно уделим им немного времени, поскольку разработчику Android приложений нужно знать об этих типах ресурсов.

Каталоге /res/raw предназначен для хранения любых файлов (например, mp3, ogg, pdf и т.д.) в их исходной форме. Эти файлы предназначены для открытия и чтения с помощью классов **InputStream**. Для получения ссылки на объект **InputStream** предназначен статический метод класса [android.content.res.Resources](#):

```
InputStream Resources.openRawResource (int id);
```

Этот метод принимает идентификатор ресурса, который имеет вид: **R.raw.имя_файла**. То есть, всем файлам, размещенным в каталоге /res/raw присваиваются идентификаторы.

В каталоге /assets также можно размещать файлы в их исходной форме. Файлы в каталоге /assets не получают идентификатора ресурса. Доступ к этим файлам возможен с помощью объекта класса [android.content.res.AssetManager](#). Есть еще одна возможность, которую разработчик получает при размещении файлов в каталоге /assets: можно получать доступ к исходным именам файлов и иерархии файлов.

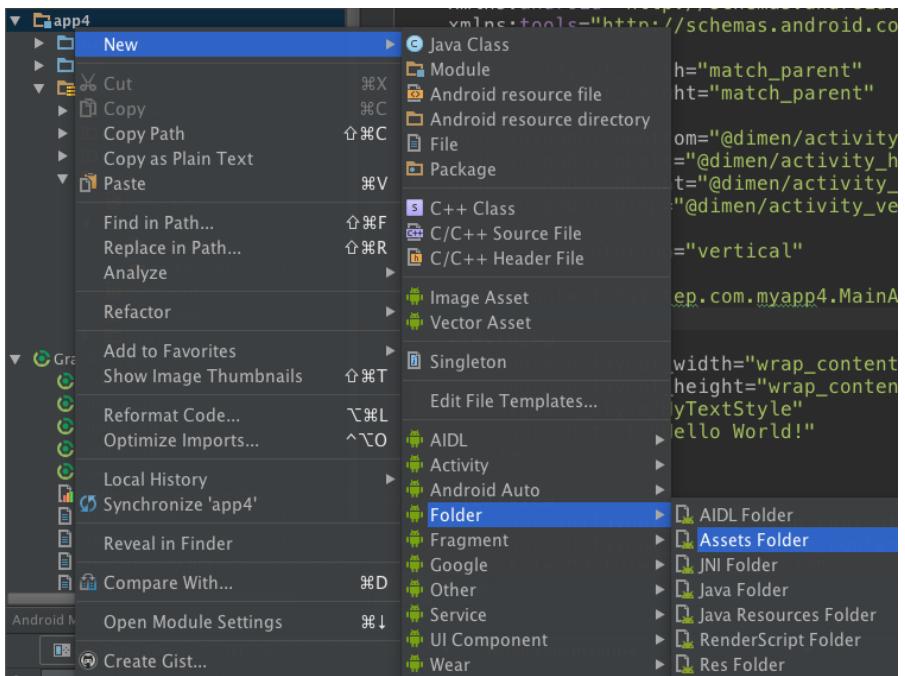


Рис. 1.51. Создание каталога /assets в Android Studio

В отличие от других каталогов ресурсов, каталог /assets не размещается внутри каталога /res. Каталог /assets располагается на уровне каталогов /manifests, /java, /res.

Чтобы создать каталог /assets, необходимо кликнуть правой кнопкой мыши на названии модуля, затем в выпадающем меню необходимо выбрать пункт *Folder* и в следующем выпадающем меню выбрать пункт *Assets Folder* (см. Рис. 1.51).

Получить ссылку на объект `android.content.res.AssetManager` можно с помощью метода объекта Активности:

```
AssetManager getAssets();
```

Рассмотрим методы класса `android.content.res.AssetManager`:

- `final String[] list(String path)` — возвращает массив названий файлов и каталогов по указанному пути.
- `final InputStream open(String fileName, int accessMode)` — открывает файл, находящийся в каталоге /assets и возвращает ссылку на байтовый поток чтения `InputStream`, с помощью которого можно прочитать содержимое файла. Принимает название файла и режим, указывающий способ извлечения данных. Для режима извлечения данных предусмотрены константы: `ACCESS_UNKNOWN` (режим извлечения данных не определен), `ACCESS_STREAMING` (последовательное извлечение данных с чтением только вперед), `ACCESS_RANDOM` (извлечение данных с возможностью перемещения в байтовом потоке вперед и назад), `ACCESS_BUFFER` (содержимое файла необходимо загрузить в память для частых операций чтения небольшими порциями).
- `final InputStream open(String fileName)` — открывает

файл, находящийся в каталоге /assets с режимом извлечения данных **ACCESS_STREAMING**.

Попробовать на практике работу с файлами из каталогов /res/raw и /assets вам будет предложено самостоятельно в качестве одного из домашних заданий.

2. Применение виджета Toolbar вместо ActionBar

Виджет **Toolbar** позволяет сохранить функциональность знакомой вам панели **ActionBar**, и, в то же время, дает возможность очень легко и просто стилизовать этот элемент. **Toolbar**, как будет показано ниже, является частью макета раскладки Активности, и это облегчает работу с ним, так как упрощает доступ к его атрибутам. Виджет **Toolbar** начинает играть в приложениях все более значимую роль, вытесняя панель **ActionBar**. Более того, в некоторых версиях Android Studio, после создания нового модуля можно наблюдать такую картину, которая изображена на Рис. 2.1.

Такое сообщение появляется отчасти потому, что **ActionBar** в текущей конфигурации приложения отменен. Можно исправить ситуацию, заменив версию библиотеки **AppCompat** в файле **build.gradle** для модуля (см. Рис. 2.2) на версию, которая указана в Листинге 2.1.

Листинг 2.1. Изменение версии библиотеки AppCompat для поддержки панели ActionBar

```
dependencies
{
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:23.0.1'
}
```

2. Применение виджета Toolbar вместо ActionBar

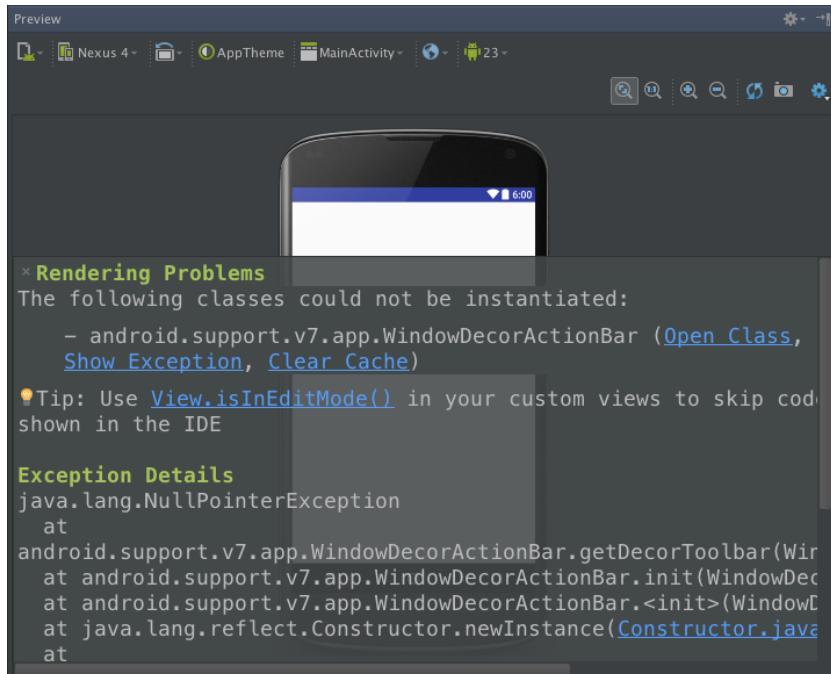


Рис. 2.1. Сообщение об ошибке в Android Studio, связанное с использованием ActionBar

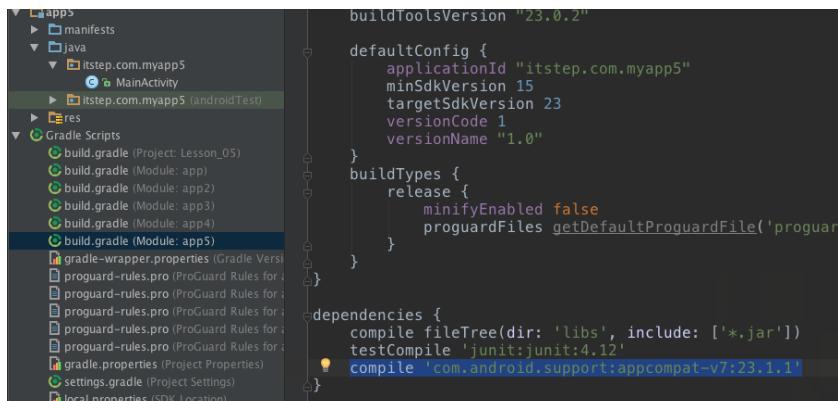


Рис. 2.2. Содержимое файла build.gradle для модуля app5, которое подлежит редактированию для поддержки ActionBar

Из Листинга 2.1 видно, для поддержки **ActionBar** пришлось указать более старую версию библиотеки **AppCompat** (минимальная версия, в которой сохранилась поддержка **ActionBar** это 23.0.1). Такое решение имеет право на существование, но мы рекомендуем использовать виджет **Toolbar**, чemu и посвящен этот раздел урока.

Примечание. Если захотите попробовать решение из Листинга 2.1, не забудьте синхронизировать gradle, с помощью клика на появившейся в правом верхнем углу Android Studio ссылке голубого цвета с надписью Sync Gradle.

Для того, чтобы вставить виджет [android.support.v7.widget.Toolbar](#) в наше приложение вместо панели **ActionBar**, необходимо выполнить несколько шагов.

ПЕРВОЕ. Необходимо в файле ресурсов /res/values/styles.xml выбрать в качестве родительской темы для приложения тему без **ActionBar**. В рассматриваемом нами примере (модуль app5) мы выбрали тему **Theme.AppCompat.Light.NoActionBar** (см. Листинг 2.2).

Листинг 2.2. Выбор темы приложения без панели **ActionBar**

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent=
        "Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">
            @color/colorPrimary</item>
```

```
<item name="colorPrimaryDark">  
    @color/colorPrimaryDark</item>  
<item name="colorAccent">  
    @color/colorAccent</item>  
</style>  
</resources>
```

Сразу же сообщение об ошибке, которое показано на Рис. 2.1, исчезло, а предпросмотр внешнего вида Активности стал выглядеть, как показано на Рис. 2.3.

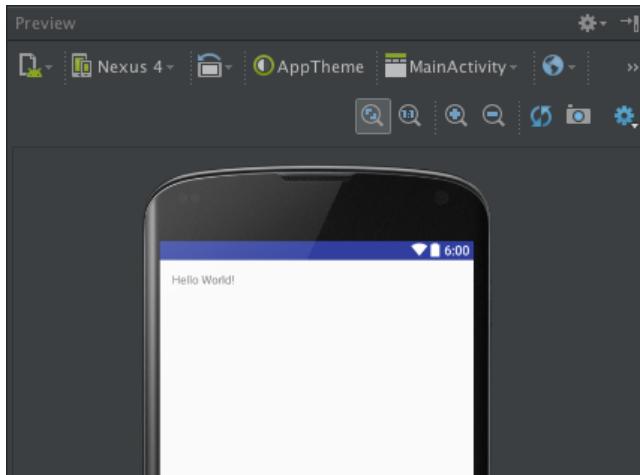


Рис. 2.3. Результат применения темы приложения без панели ActionBar

ВТОРОЕ. Добавляем виджет `android.support.v7.widget.Toolbar` в макет Активности, как показано в Листинге 2.3. Обратите внимание, что используется класс `android.support.v7.widget.Toolbar` из библиотеки поддержки v7 (для совместимости со старыми и новыми версиями API).

Листинг 2.3. Добавление виджета
android.support.v7.widget.Toolbar в файл макета Активности
/res/layout/activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom=
        "@dimen/activity_vertical_margin"
    android:paddingLeft=
        "@dimen/activity_horizontal_margin"
    android:paddingRight=
        "@dimen/activity_horizontal_margin"
    android:paddingTop=
        "@dimen/activity_vertical_margin"

    android:orientation="vertical"

    tools:context="itstep.com.myapp5.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="#009688"
    />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"/></LinearLayout>
```

Как видно из Листинга 2.3, виджет **Toolbar** имеет высоту 60dp и зеленоватый цвет фона (#009688). Внешний вид приложения из Листинга 2.3 изображен на Рис. 2.4 слева.

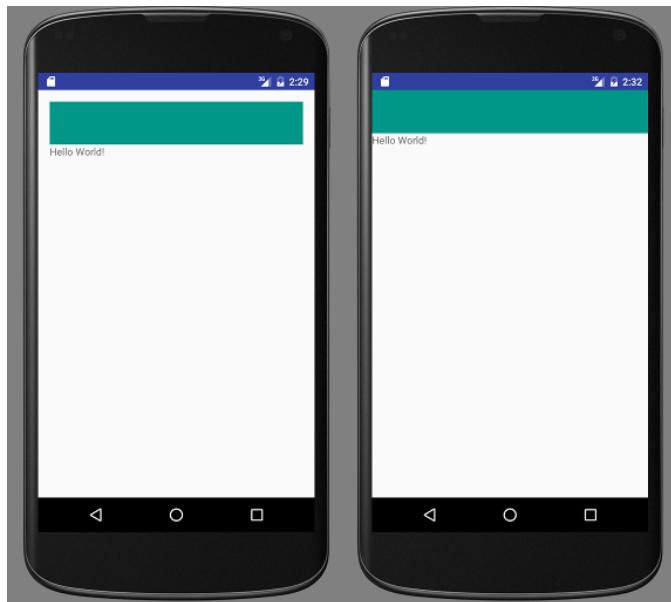


Рис. 2.4. Внешний вид размещенного в Листинге 2.3 виджета Toolbar. Слева у главного контейнера Активности есть отступы, справа отступы удалены

Как видно из левого скриншота Рис. 2.4, эстетичному размещению виджета **Toolbar** мешают внешние отступы главного контейнера Активности `android.widget.LinearLayout` (см. Рис. 2.5).

Эти отступы необходимо убрать. Внешний вид виджета **Toolbar** без отступов главного контейнера изображен на Рис. 2.4 справа.

```
<?xml version="1.0" encoding="utf-8"?>
<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:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"

    android:orientation="vertical"

    tools:context="itstep.com.myapp5.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:background="#009688"
    />
```

Рис. 2.5. Внутренние отступы главного контейнера Активности, которые необходимо удалить

ТРЕТЬЕ. Необходимо установить **Toolbar** в качестве панели действий (**ActionBar**) для Активности. Для этого необходимо воспользоваться методом класса Активности:

```
void setSupportActionBar (Toolbar toolbar);
```

В этот метод необходимо передать ссылку на объект **Toolbar**. Как это делается показано в Листинге 2.4.

Листинг 2.4. Назначение виджета Toolbar в качестве ActionBar для Активности

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

```
// ----- android.support.v7.widget.Toolbar -----
Toolbar toolBar = (Toolbar) this.findViewById(R.id.toolbar);
this.setSupportActionBar(toolBar);
}
```

Теперь у приложения на виджете **Toolbar** отобразится название приложения (см. Рис. 2.6, скриншот слева).

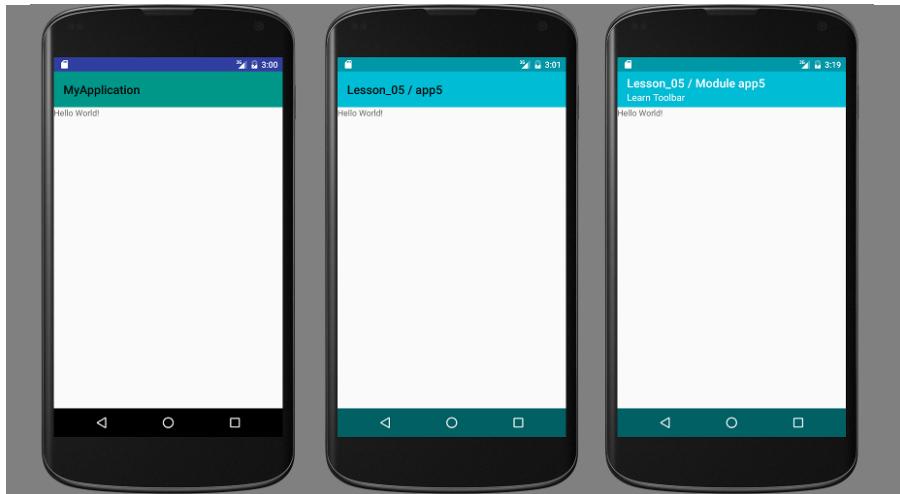


Рис. 2.6. Внешний вид работы примеров из Листингов 2.4, 2.5, 2.6

Давайте кастомизируем внешний вид нашего приложения, а именно подберем цвета для панели состояния, панели тулбара и панели навигации, как это рассматривалось в разделе 1.5 данного урока (см. Рис. 1.45).

Цветовую гамму выберем с рекомендованного компанией Google сайта <https://material.io/guidelines/style/color.html>. Для нашего примера мы выбрали цветовую гамму *Суан* (см. Рис. 2.6, скриншот в центре). Причем виджету

Toolbar, цвет фона из цветовой гаммы *Cyan*, необходимо задать непосредственно в xml файле макета Активности (*/res/layout/activity_main.xml*), как показано в Листинге 2.5.

Также поменяем цвет текста для заголовка (и сразу же для подзаголовка) приложения, который располагается на виджете **Toolbar** (Листинг 2.5). Текст подзаголовка можно задать в xml файле (атрибут **android:subtitle** для элемента `<android.support.v7.widget.Toolbar>`), но по каким-то причинам он не отображается, поэтому зададим текст подзаголовка в методе **onCreate()** класса Активности (Листинг 2.6). Кстати, там же в **onCreate()** можно задать и цвета текста и заголовков (при помощи методов `toolbar.setTitleTextColor()` и `toolbar.setSubtitleTextColor()`).

Листинг 2.5. Назначение цвета текста для заголовка и подзаголовка Toolbar

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="60dp"
    android:background="#00BCD4"
    android:subtitleTextColor="#FFFFFF"
    android:titleTextColor="#FFFFFF"
    />
```

Листинг 2.6. Программное назначение текста и цвета текста для заголовка и подзаголовка тулбара

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

```
// ----- android.support.v7.widget.Toolbar -----
Toolbar toolBar = (Toolbar)
        this.findViewById(R.id.toolbar);
this.setSupportActionBar(toolBar);

// ----- Assign title and subtitle text -----
// ----- Назначаем текст заголовка и подзаголовка
toolBar.setTitle("Lesson_05 / Module app5");
toolBar.setSubtitle("Learn Toolbar");

// ----- Assign title and subtitle text color -----
// ----- Назначаем текст заголовка и подзаголовка
toolBar.setTitleTextColor(Color.WHITE);
toolBar.setSubtitleTextColor(Color.WHITE);
}
```

Внешний вид работы примера из Листинга 2.6 изображен на Рис. 2.6 справа.

ЧЕТВЕРТОЕ. Создадим файл ресурсов для Меню приложения, которое будет отображаться в виджете **Toolbar**. Для этого добавим каталог `/res/menu` в модуль нашего проекта, и в этот каталог добавим файл ресурсов `/res/menu/menu_main.xml`. Содержимое файла `/res/menu/menu_main.xml` приведено в Листинге 2.7.

Листинг 2.7. Содержимое файла ресурсов `/res/menu/menu_main.xml` рассматриваемого примера

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".MainActivity">
```

```
<item android:id="@+id/action_settings"
      android:title="@string/action_settings"
      android:orderInCategory="100"
      app:showAsAction="never" />
<item android:id="@+id/action_options"
      android:title="@string/action_options"
      android:orderInCategory="110"
      app:showAsAction="never" />
/>
```

Как видно из Листинга 2.7, в файле ресурсов `/res/menu/menu_main.xml` создается два пункта меню: *Settings* (идентификатор пункта `@+id/action_settings` и название пункта задается строковым ресурсом `@string/action_settings`) и *Options* (идентификатор пункта `@+id/action_options` и название пункта задается строковым ресурсом `@string/action_options`).

Чтобы меню из файла ресурсов отобразилось в тулбаре, необходимо в класс Активности добавить методы `onCreateOptionsMenu()`, который будет автоматически вызываться для того, чтобы Активность смогла создать Меню приложения и назначить его приложению, и `onOptionsItemSelected()`, который будет вызываться при выборе пользователем пункта Меню. Добавляем эти методы в класс Активности как показано в Листинге 2.8.

Создание Меню приложения рассматривалось нами в одном из предыдущих уроков, поэтому в данном примере мы можем еще раз вспомнить, как это делается и убедиться в том, что создание Меню для виджета `Toolbar` ничем не отличается от создания Меню для `ActionBar`.

Листинг 2.8. Добавление в класс Активности методов onCreateOptionsMenu() и onOptionsItemSelected() для создания Меню приложения и для обработки события выбора пунктов меню соответственно

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

    @Override
    public boolean onOptionsItemSelected(MenuItem item)
    {

        int id = item.getItemId();
        switch (id)
        {
            case R.id.action_settings :
                Toast.makeText(this,
                    "Item 'Settings' selected",
                    Toast.LENGTH_LONG).show();
                return true;

            case R.id.action_options :
                Toast.makeText(this,
                    "Item 'Options' selected",
                    Toast.LENGTH_LONG).show();
                return true;
        }
        return super.onOptionsItemSelected(item);
    }
}
```

Как видно из Листинга 2.8, при выборе пользователем пункта меню появится Тост с соответствующим сообщением. Внешний вид работы Меню приложения из Листинга 2.8 изображен на Рис. 2.7.

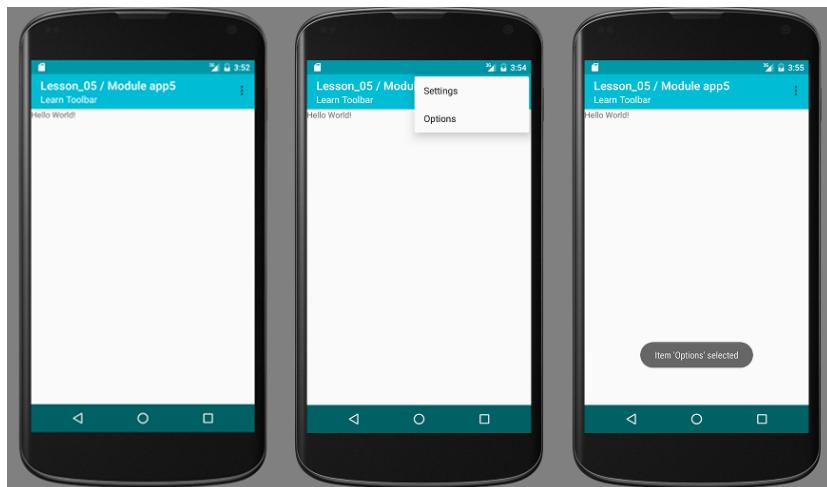


Рис. 2.7. Внешний вид работы Меню приложения, размещенного в виджете `android.support.v7.widget.Toolbar`

ПЯТОЕ. В тулбаре можно разместить иконку приложения в качестве логотипа. Это делается при помощи вызова метода класса `android.support.v7.widget.Toolbar`:

```
void setLogo(int resId);
```

Метод принимает идентификатор *Drawable* ресурсов. Можно передать и идентификатор изображения из *Map* ресурсов. В Листинге 2.9 виджету `android.support.v7.widget.Toolbar` назначается иконка логотипа приложения. Внешний вид приложения из Листинга 2.9 изображен на Рис. 2.8 слева.

Листинг 2.9. Назначение иконки с логотипом приложения виджету android.support.v7.widget.Toolbar

```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    ...
    // ----- Set logo image for Toolbar -----
    // ----- Устанавливаем пиктограмму для
    // ----- логотипа приложения -----
    toolbar.setLogo(R.mipmap.ic_launcher);
}

```

Также у виджета `android.support.v7.widget.Toolbar` есть возможность устанавливать так называемую пиктограмму (иконку) навигации, которая отображается в левой части тулбара и может генерировать события нажатия. Эта пиктограмма предназначена для предоставления возможности разработчику приложения создавать функциональность, позволяющую пользователю, например перемещаться вперед или назад между экранами Активности или размещать какую-нибудь специальную иконку, нажав на которую будет появляться панель поиска и так далее и тому подобное.

Добавить пиктограмму навигации можно с помощью метода класса `android.support.v7.widget.Toolbar`:

```
void setNavigationIcon(int resId);
```

Метод принимает идентификатор *Drawable Resources* ресурсов или идентификатор изображения из *MipMap* ресурсов. В Листинге 2.10 показан код добавления пиктограммы

навигации для тулбара рассматриваемого примера. Иконка для пиктограммы навигации (символ плюсик в кружочке) была скачана с сайта <http://material.io/icons> и размещена в *MipMap* ресурсы приложения, как было описано в разделе данного урока, посвященного *MipMap*-текстурированию. Внешний вид работы примера из Листинга 2.10 изображен на Рис. 2.8 в центре.

Листинг 2.10. Назначение виджету android.support.v7.widget.Toolbar пиктограммы навигации

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    ...
    // ----- Set logo image for Toolbar -----
    // ----- Устанавливаем пиктограмму для
    // ----- логотипа приложения -----
    toolbar.setLogo(R.mipmap.ic_launcher);

    // ----- Set navigation image for Toolbar -----
    // ----- Устанавливаем пиктограмму навигации
    // ----- для тулбара -----
    toolbar.setNavigationIcon(R.mipmap.
        ic_add_circle_outline_white_36dp);
}
```

Теперь давайте назначим обработчик события нажатия на пиктограмму навигации. Это делается вызовом метода класса **android.support.v7.widget.Toolbar**:

```
void setNavigationOnClickListener
    (View.OnClickListener listener);
```

Метод принимает ссылку на объект, который должен реализовать хорошо знакомый нам интерфейс **android.**

widget.View.OnClickListener. Назначение обработчика события нажатия на пиктограмму навигации приведено в Листинге 2.11. Внешний вид работы примера из Листинга 2.11 изображен на Рис. 2.8 справа.

Листинг 2.11. Назначение обработчика события нажатия на иконку навигации виджета android.support.v7.widget.Toolbar

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    ..
    // ----- Set navigation image for Toolbar -----
    // ----- Устанавливаем пиктограмму навигации
    // ----- для тулбара -----
    toolBar.setNavigationIcon(R.mipmap.ic_add_
                                _circle_outline_white_36dp);

    // ----- Set Navigation Icon Click Listener
    // ----- for Toolbar -----
    // ----- Назначаем обработчик события нажатия
    // ----- на иконку навигации тулбара -----
    toolBar.setNavigationOnClickListener(
        new View.OnClickListener()
    {
        @Override
        public void onClick(View view)
        {
            Toast.makeText(MainActivity.this,
                "Navigation Icon Clicked!!!",
                Toast.LENGTH_LONG).show();
        }
    });
}
```

Как видите, внедрение виджета **android.support.v7.widget.Toolbar** в приложение вместо **ActionBar** является

несложной задачей, и взамен разработчик приложения получает больше возможностей для создания функциональности приложения.

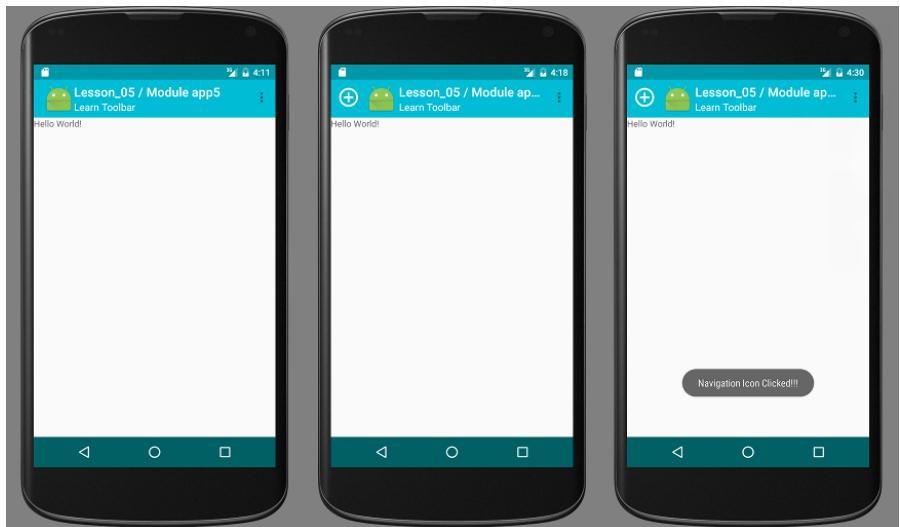


Рис. 2.8. Внешний вид работы примеров из Листингов 2.9, 2.10, 2.11

Исходный код данного раздела можно найти в модуле *app5* проекта, среди файлов с исходными кодами, которые прилагаются к данному уроку.

3. Анимация. Создание анимации в ресурсах приложения. Применение анимации

Возможности применения эффектов анимации (в дальнейшем просто Анимации) в Android приложениях весьма обширны. Обобщая, можно выделить следующие типы Анимации:

- **Property Animation** — анимация осуществляется при помощи изменения свойств некоторого виджета в течение указанного периода времени в указанном диапазоне значений. Плавное изменение свойств виджета (например, прозрачности) визуально создает эффект анимации;
- **Value Animation** — анимация осуществляется при помощи изменения некоторого значения в течение указанного периода времени в указанном диапазоне значений. Разработчику необходимо обрабатывать события изменения этого значения и применять текущую величину этого значения к свойству или свойствам виджета;
- **Frame Animation** — Анимация осуществляется при помощи смены последовательности изображений через указанные промежутки времени. Смена изобра-

жений в течение времени аналогична смене кадров в кинематографе.

Перечисленные выше Анимации доступны для применения как с помощью xml ресурсов приложения, так и с помощью программного кода Java. Поскольку данный урок посвящен теме *ресурсов приложения*, то акцент при рассмотрении вышеперечисленных Анимаций будет сделан на создание анимаций в ресурсах приложения.

3.1. Property Animation. Анимация путем изменения свойств виджета. Примеры

Property Animation — это анимация, позволяющая менять для виджетов в течение заданного времени величины таких свойств как прозрачность, цвет фона и так далее. Файлы с ресурсами *Property Animation* размещаются в каталоге /res/ animator. Название файла из каталога /res/ animator впоследствии используется как идентификатор для получения доступа к ресурсу анимации. Например, для файла res/ animator/ filename.xml будут такие идентификаторы: в Java коде — R.animator.filename, в xml коде — @[package:]animator/ filename.

Синтаксис файла ресурсов *Property Animation* приведен в Листинге 3.1.

Листинг 3.1. Синтаксис xml файла ресурсов Property Animation

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

```
    android:propertyName="string"
    android:duration="int"
    android:valueFrom="float | int | color"
    android:valueTo="float | int | color"
    android:startOffset="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]
    android:valueType=["intType" | "floatType" |
                      "colorType"]
  />
```

Как видно из Листинга 3.1, элемент `<objectAnimator>` описывает одну анимацию определенного свойства виджета. На основе элемента `<objectAnimator>` создается объект [android.animation.ObjectAnimator](#).

Атрибуты элемента `<objectAnimator>`:

- **android:propertyName** — обязательный атрибут строкового типа. Содержит название свойства объекта (виджета), которое подлежит изменению. Например, можно указать в качестве названия свойства виджета *alpha*, *x*, *y* или *backgroundColor*. К сожалению, далеко не все свойства виджетов доступны для этого типа анимации (например, свойства *width* и *height* модифицировать нельзя), поэтому для анимации большинства свойств виджетов можно использовать методы `ofТип()` (`ofInt()`, `ofFloat()`, `ofArgb()` и так далее) класса [android.animation.ObjectAnimator](#) или воспользоваться анимацией *Value Animation*, которая является более гибкой. Однако, применение анимации *Property Animation* к указанным свойствам *alpha* и *backgroundColor* будут рассмотрены в данном разделе, а об использовании

методов **ofType()** будет рассказано в разделе №4 данного урока.

- **android:valueTo** — обязательный атрибут целого типа, вещественного типа или типа **Color**. Содержит конечное значение для свойства, на котором анимация *Property Animation* должна остановиться. Значение цвета для этого атрибута задаются в виде #RRGGBB.
- **android:valueFrom** — обязательный атрибут целого типа, вещественного типа или типа **Color**. Содержит начальное значение, начиная с которого анимация *Property Animation* начнет выполняться. Значение цвета для этого атрибута задаются в виде #RRGGBB.
- **android:duration** — атрибут целого типа. Содержит длительность анимации в миллисекундах. Атрибут необязательный. Значением по умолчанию является 300 миллисекунд.
- **android:startOffset** — атрибут целого типа. Указывает время задержки перед началом выполнения *Property Animation*. Время указывается в миллисекундах. Атрибут необязательный. Значением по умолчанию является 0 миллисекунд.
- **android:repeatCount** — атрибут целого типа. Указывает количество раз повтора анимации. Если атрибуту присвоить значение -1, то анимация будет исполняться бесконечно. Значение 1 означает, что анимация повторится один раз после выполнения (то есть всего анимация отработает 2 раза). Атрибут необязательный. Значением по умолчанию является значение 0, что означает отсутствие повторений.

- **android:repeatMode** — атрибут целого типа. Указывает на поведение анимации после ее завершения. При этом значение атрибута **android:repeatCount** должно быть установлено в значение **-1** или в любое положительное значение. Атрибут **android:repeatMode** может принимать значения: **reverse** — означает, что анимация должна выполнится в обратном порядке от предыдущей итерации, **repeat** — означает, что анимация должна повторяться всегда в прямом порядке (от значения заданного в атрибуте **android:valueFrom** до значения, заданного в атрибуте **android:valueTo**).
- **android:valueType** — указывает тип изменяемого свойства. Допустимые значения: **intType** — свойство целого типа, **floatType** — вещественного, **colorType** — свойство типа Цвет (**Color**).

Для запуска анимации *Property Animation* необходимо проделать следующие шаги: получить ссылку на объект **android.animation.ObjectAnimator** при помощи статического метода класса **android.animation.AnimatorInflater**:

```
static Animator loadAnimator(Context context, int id);
```

Метод принимает ссылку на контекст приложения и идентификатор ресурса *Property Animation*.

Получив ссылку на объект **android.animation.ObjectAnimator** (нужно будет привести к типу **android.animation.ObjectAnimator** возвращаемое методом **loadAnimator()** значение), необходимо указать этому объекту ссылку на виджет, который подлежит применению

анимации при помощи метода `setTarget()`. И затем запустить анимацию при помощи вызова метода `start()` объекта `android.animation.ObjectAnimator`. Данная последовательность шагов приведена в Листинге 3.2.

Листинг 3.2. Запуск анимации Property Animation на исполнение

```
ObjectAnimator objectAnimator =  
    (ObjectAnimator) AnimatorInflater.loadAnimator(  
        this, R.animator.file_name);  
objectAnimator.setTarget(this.findViewById(  
    R.id.widget_identifier));  
objectAnimator.start();
```

Давайте на рассмотрим на примере использование анимации *Property Animation*. Для примера *Property Animation* создан модуль *app6*. В модуле *app6*, для закрепления полученного ранее материала, создан **Toolbar**.

Далее, поскольку в примере будет рассматриваться изменение таких свойств виджетов, как цвет фона (свойство `backgroundColor`), прозрачность (свойство `alpha`), расположение (свойства `x` и `y`), то для Активности создадим фоновое изображение (в виде *Bitmap Drawable* ресурса), которое будет нам визуально подчеркивать применяемые эффекты анимации.

С сайта <https://www.toptal.com/designers/subtlepatterns/> было скачано подходящее для фона изображение и размещено в папку ресурсов: `/res/drawable/texture.png`. Затем создан файл *Bitmap Drawable* (`/res/drawable/xml_texture.xml`), содержимое которого приведено в Листинге 3.3.

Листинг 3.3. Содержимое файла ресурсов
/res/drawable/xml_texture.xml

```
<?xml version="1.0" encoding="utf-8"?>
<bitmap
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:src="@drawable/texture"
    android:tileMode="repeat"
/>
```

Как видно из Листинга 3.3, файл ресурсов *Bitmap Drawable* /res/drawable/xml_texture.xml размножает изображение /res/drawable/texture.png по вертикали и горизонтали. Применим изображение из файла ресурсов (выделено жирным шрифтом в Листинге 3.4) в качестве фонового изображения для главного контейнера Активности (файл /res/layout/activity_main.xml). Внешний вид примененного фонового изображения показан на Рис. 3.1 слева.

Листинг 3.4. Файл макета внешнего вида Активности
(/res/layout/activity_main.xml) рассматриваемого примера

```
<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"
    android:background="@drawable/xml_texture"
    tools:context="itstep.com.myapp6.MainActivity">
```

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="60dp"  
    android:background="#CDDC39" />  
  
<Space  
    android:layout_width="10dp"  
    android:layout_height="10dp" />  
  
<FrameLayout  
    android:layout_width="200dp"  
    android:layout_height="100dp"  
    android:layout_gravity="center_horizontal"  
    android:layout_margin=  
        "@dimen/activity_horizontal_margin"  
    android:background="#33691E"  
    android:id="@+id/flOne" />  
  
<Space  
    android:layout_width="10dp"  
    android:layout_height="10dp" />  
  
<FrameLayout  
    android:layout_width="200dp"  
    android:layout_height="100dp"  
    android:layout_gravity="center_horizontal"  
    android:layout_margin=  
        "@dimen/activity_horizontal_margin"  
    android:background="#33691E"  
    android:id="@+id/flTwo" />  
  
<Space  
    android:layout_width="10dp"  
    android:layout_height="10dp" />
```

```
<FrameLayout  
    android:layout_width="200dp"  
    android:layout_height="100dp"  
    android:layout_gravity="center_horizontal"  
    android:layout_margin=  
        "@dimen/activity_horizontal_margin"  
    android:background="#33691E"  
    android:id="@+id/flThree" />  
</LinearLayout>
```

Также, как видно из Листинга 3.4, для рассматриваемых примеров созданы в макете Активности виджеты `android.widget.FrameLayout` с идентификаторами `flOne`, `flTwo`, `flThree`. Этим виджетам заданы размеры и цвет фона (см. Рис. 3.1, скриншот в центре). С этими виджетами мы и будем экспериментировать.

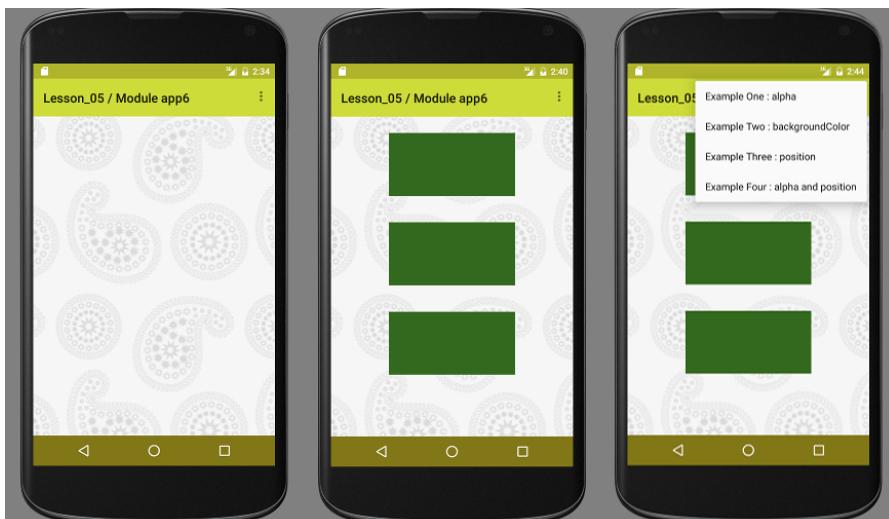


Рис. 3.1. Скриншоты, на которых изображены объекты, участвующие в рассматриваемом в данном разделе примере приложения

Так же создано меню приложения, в виде файлов ресурсов /res/menu/menu_main.xml (см. Листинг 3.5), чтобы иметь удобную возможность запускать на исполнение анимации из рассматриваемых в этом разделе примеров. Внешний вид меню приложения можно увидеть на Рис. 3.1 справа.

Листинг 3.5. Меню приложения из рассматриваемого
в данном разделе примера

```
<?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"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context=".MainActivity">

    <item android:id="@+id/action_animation_alpha"
          android:title="Example One : alpha"
          android:orderInCategory="100"
          app:showAsAction="never" />

    <item android:id=
          "@+id/action_animation_background"
          android:title="Example Two : backgroundColor"
          android:orderInCategory="110"
          app:showAsAction="never" />

    <item android:id="@+id/action_animation_position"
          android:title="Example Three : position"
          android:orderInCategory="120"
          app:showAsAction="never" />

    <item android:id="@+id/
              action_animation_combination"
```

```

        android:title=
            "Example Four : alpha and position"
        android:orderInCategory="130"
        app:showAsAction="never" />
</menu>
```

Запуск анимаций на исполнение будет осуществляться в методе обработчике событий выбора пунктов меню **onOptionsItemSelected()** класса Активности **MainActivity**.

Итак, все подготовительные работы закончены и теперь можно приступать к примерам. Первый пример будет на анимацию виджета, с помощью изменения свойства прозрачности **alpha**. Создадим файл ресурсов Property Animation с названием **alpha_value_animator.xml** (располагается: **/res/animator/alpha_value_animator.xml**). Содержимое файла **/res/animator/alpha_value_animator.xml** приведено в Листинге 3.6.

Листинг 3.6. Содержимое файла ресурсов Property Animation **/res/animator/alpha_value_animator.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="alpha"
    android:startOffset="200"
    android:duration="2000"
    android:valueType="floatType"
    android:valueFrom="1.0"
    android:valueTo="0.2"
/>
```

Как видно из Листинга 3.6, анимация *Property Animation* будет изменять свойство **alpha** (см. атрибут **android:propertyName**) виджета в течение 2000 миллисекунд (атрибут **android:duration**), начиная со значения 1.0 и заканчивая значением 0.2 (см. значения атрибутов **android:valueFrom**, **android:valueTo**). Тип изменяемого свойства указан в атрибуте **android:valueType** и указан в виде значения **floatType**. Анимация начнется через 200 миллисекунд (см. атрибут **android:startOffset**). Запуск анимации *Property Animation* /res/animator/alpha_value_animator.xml для виджета с идентификатором R.id.flOne осуществляется в методе **onOptionsItemSelected()** класса Активности **MainActivity**, при обработке события выбора пункта меню с идентификатором **action_animation_alpha** (см. Листинг 3.7).

Листинг 3.7. Запуск анимации Property Animation для свойства alpha виджета с идентификатором R.id.flOne

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    switch (id)
    {
        // ---- Start Property Animation for alpha property
        // ---- Запуск Property Animation для свойства
        // ---- alpha виджета -----
        case R.id.action_animation_alpha :
        {
            ObjectAnimator objectAnimator =
                (ObjectAnimator)
                    AnimatorInflater.
                        loadAnimator(
```

```
        this,  
        R.animator.  
        alpha_value_animator);  
  
    objectAnimator.setTarget(this.  
        findViewById(R.id.flOne));  
    objectAnimator.start();  
}  
return true;  
..  
}  
  
return super.onOptionsItemSelected(item);  
}
```

Последовательность исполнения анимации из Листингов 3.6 и 3.7 изображена на Рис. 3.2.

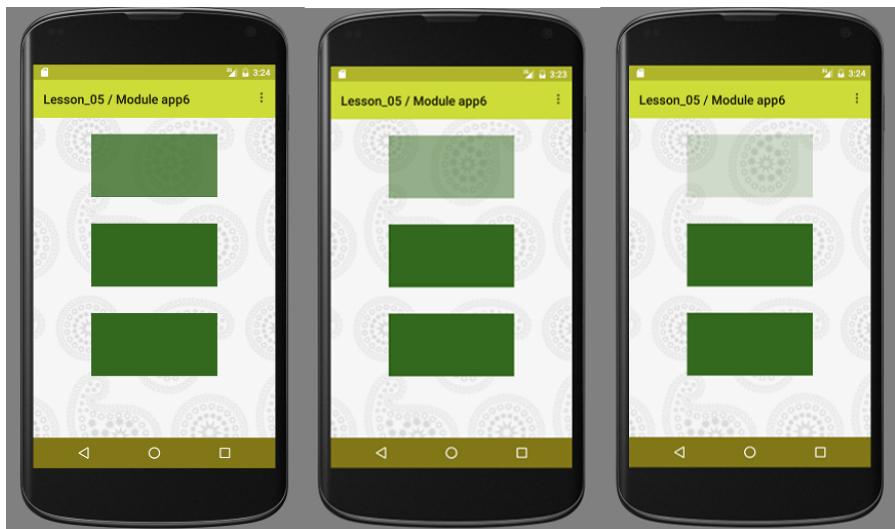


Рис. 3.2. Последовательность скриншотов, отображающая исполнение анимации Property Animation из Листинга 3.7 по изменению свойства alpha виджета

Чтобы запустить анимацию представленную в Листинге 3.7 на исполнение, необходимо выбрать пункт меню *Example One: alpha*.

Следующим примером Property Animation будет пример на изменение свойства **backgroundColor** виджета. Для этого создадим файл ресурсов Property Animation /res/animator/background_value_animator.xml. Содержимое этого файла показано в Листинге 3.8.

Листинг 3.8. Содержимое файла ресурсов Property Animation /res/animator/background_value_animator.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="backgroundColor"
    android:startOffset="200"
    android:duration="2000"
    android:valueType="colorType"
    android:valueFrom="#33691E"
    android:valueTo="#FF9800"
/>
```

Как видно из Листинга 3.8, анимация *Property Animation* будет изменять свойство **backgroundColor** (см. атрибут **android:propertyName**) виджета в течение 2000 миллисекунд (атрибут **android:duration**), начиная со значения #33691E (зеленый) и заканчивая значением #FF9800 (оранжевый) (см. значения атрибутов **android:valueFrom**, **android:valueTo**). Тип изменяемого свойства указан в атрибуте **android:valueType** и указан в виде значения

colorType. Анимация начнется через 200 миллисекунд (см. атрибут `android:startOffset`). Задержка анимации в 200 миллисекунд сделана для того, чтобы меню приложения успело исчезнуть с экрана, чтобы не мешать наблюдению за началом анимации.

Запуск анимации `Property Animation /res/animator/background_value_animator.xml` для виджета с идентификатором `R.id.flTwo` осуществляется в методе `onOptionsItemSelected()` класса Активности `MainActivity`, при обработке события выбора пункта меню с идентификатором `action_animation_background`.



Рис. 3.3. Последовательность исполнения анимации из Листинга 3.8

Листинг для запуска этой анимации в уроке не приводится, так как программный код запуска аналогичен программному коду из Листинга 3.6, за исключением

того, что в метод `AnimatorInflater.loadAnimator()` передается идентификатор ресурса `R.animator.background_value_animator`, а объектом, для которого применяется анимация, является виджет с идентификатором `R.id.flTwo`. Исходный код этого примера находится в файле `MainActivity.java` модуля `app6`.

Последовательность исполнения анимации из Листинга 3.8 изображена на Рис. 3.3. Чтобы запустить эту анимацию на исполнение, необходимо выбрать пункт меню *Example Two: backgroundColor*.

Рекомендуем выполнить этот пример в эмуляторе или реальном устройстве, так как Рис. 3.3 не может в полной степени отобразить исполнение этой анимации.

Следующим примером *Property Animation* будет пример на изменение позиции виджета (свойство `x`). Для этого создадим файл ресурсов *Property Animation* `/res/animator/position_x_value_animator.xml`. Содержимое этого файла показано в Листинге 3.9.

Листинг 3.9. Содержимое файла ресурсов *Property Animation* `/res/animator/position_x_value_animator.xml`

```
<?xml version="1.0" encoding="utf-8"?>

<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="x"
    android:startOffset="200"
    android:duration="2000"
    android:valueType="floatType"
    android:valueTo="500"
```

```
    android:repeatCount="1"  
    android:repeatMode="reverse"  
    />
```

Как видно из Листинга 3.8, анимация *Property Animation* будет изменять свойство **x** (см. атрибут **android:propertyName**) виджета в течение 2000 миллисекунд (атрибут **android:duration**), начиная с текущего значения свойства **x** виджета (атрибут **android:valueFrom** отсутствует) заканчивая значением 500 (см. значение атрибута **android:valueTo**). Тип изменяемого свойства указан в атрибуте **android:valueType** и указан в виде значения **floatType**. Анимация начнется через 200 миллисекунд (см. атрибут **android:startOffset**). Задержка анимации в 200 миллисекунд сделана для того, чтобы меню приложения успело исчезнуть с экрана, чтобы не мешать наблюдению за началом анимации. Обратите внимание (см. Листинг 3.9), что в этом примере используются атрибуты **android:repeatCount** и **android:repeatMode**. Согласно значениям, которые установлены в эти атрибуты, анимация после завершения повторится один раз в обратном направлении, что визуально будет выглядеть, как смещение виджета по горизонтали и затем возврат виджета обратно на начальную позицию.

Запуск анимации *Property Animation* /res/animator/position_x_value_animator.xml для виджета с идентификатором **R.id.flThree** осуществляется в методе **onOptionsItemSelected()** класса Активности **MainActivity**, при обработке события выбора пункта меню с идентификатором **action_animation_position**. Листинг для запуска этой

анимации в уроке не приводится, так как программный код запуска аналогичен программному коду из Листинга 3.6, за исключением того, что в метод `AnimatorInflater.loadAnimator()` передается идентификатор ресурса `R.animator.position_x_value_animator`, а объектом, для которого применяется анимация, является виджет с идентификатором `R.id.flThree`. Исходный код этого примера находится в файле `MainActivity.java` модуля `app6`.

Последовательность исполнения анимации из Листинга 3.9 изображена на Рис. 3.4. Чтобы запустить эту анимацию на исполнение, необходимо выбрать пункт меню *Example Three: position*. Рекомендуем исполнить этот пример в эмуляторе или реальном устройстве, так как Рис. 3.4 не может в полной степени отобразить исполнение этой анимации.

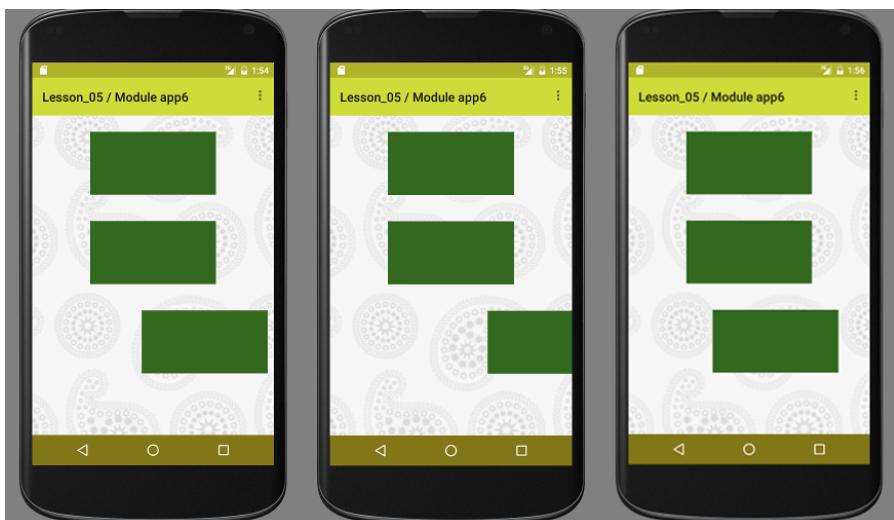


Рис. 3.4. Последовательность исполнения анимации из Листинга 3.9

И последним примером на применение *Property Animation* будет пример, демонстрирующий объединение нескольких анимаций в одну составную анимацию. Начиная с API 23, для элемента `<objectAnimator>` появился дочерний атрибут `<propertyValuesHolder>`, с помощью которого можно применять анимацию для нескольких разных свойств виджетов одновременно в одной анимации. Таким образом получается составная, комплексная анимация. Синтаксис применения элемента `<propertyValuesHolder>` показан в Листинге 3.10.

Листинг 3.10. Синтаксис создания комплексной анимации *Property Animation*, с помощью использования элементов `<propertyValuesHolder>`

```
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:startOffset="int"
    android:duration="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]>

    <propertyValuesHolder
        android:propertyName="string"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color" />

    <propertyValuesHolder
        android:propertyName="string"
        android:valueFrom="float | int | color"
        android:valueTo="float | int | color" />

    ...
</objectAnimator>
```

Как мы можем видеть из Листинга 3.10, атрибуты `android:propertyName`, `android:valueFrom`, `android:valueTo`, для комплексной анимации, необходимо указывать для элемента `<propertyValuesHolder>`. А атрибуты `android:startOffset`, `android:duration`, `android:repeatCount` и `android:repeatMode` указываются для элемента `<objectAnimator>`.

В модуле app6 создадим еще один файл ресурсов `PropertyAnimation` с названием `/res/animator/complex_animator.xml`. Содержимое этого файла приведено в Листинге 3.11.

Листинг 3.11. Содержимое файла ресурсов
Property Animation `/res/animator/complex_animator.xml`
с комплексной анимацией

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:startOffset="200"
    android:duration="2000"
    android:repeatCount="1"
    android:repeatMode="reverse">

    <propertyValuesHolder
        android:propertyName="x"
        android:valueTo="400" />

    <propertyValuesHolder
        android:propertyName="y"
        android:valueTo="200" />

    <propertyValuesHolder
```

```
    android:propertyName="alpha"  
    android:valueFrom="1.0"  
    android:valueTo="0.2" />  
  
</objectAnimator>
```

Как видно из Листинга 3.11, комплексная анимация должна выполняться в течение 2000 миллисекунд (см. атрибут **android:duration**), имеет одно повторение (см. атрибут **android:repeatCount**) с режимом повторения **reverse** (см. атрибут **android:repeatMode**). Комплексная анимация начнется через 200 миллисекунд после запуска (см. атрибут **android:startOffset**). Комплексная анимация *Property Animation* из Листинга 3.11 будет изменять такие свойства виджета: **x** (от текущего значения до значения указанного в атрибуте **android:valueTo="200"**), **y** (от текущего значения до значения указанного в атрибуте **android:valueTo="400"**), **alpha** (от значения **android:valueFrom="1.0"** до значения **android:valueTo="0.2"**). Запуск комплексной анимации из Листинга 3.11 аналогичен запуску обычной анимации *Property Animation*, который приведен в Листинге 3.6. Только в метод **AnimatorInflater.loadAnimator()** передается идентификатор ресурса **R.animator.complex_animator**, а объектом, для которого применяется анимация, является виджет с идентификатором **R.id.flThree**. Исходный код этого примера находится в файле **MainActivity.java** модуля **app6**.

Последовательность исполнения анимации из Листинга 3.11 изображена на Рис. 3.5. Чтобы запустить эту анимацию на исполнение, необходимо выбрать пункт меню *Example Four: alpha and position*.

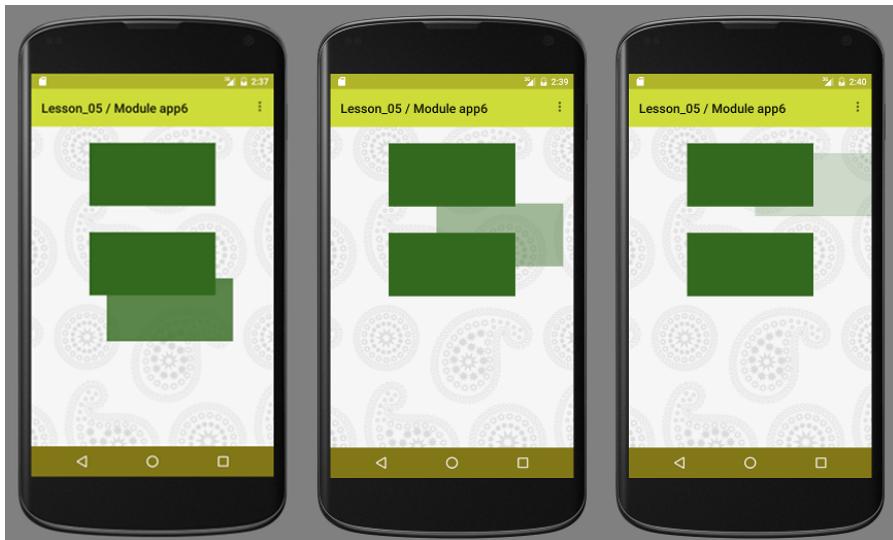


Рис. 3.5. Последовательность исполнения анимации из Листинга 3.11

Как видно на Рис. 3.5, комплексная анимация из Листинга 3.11 одновременно изменяет значения нескольких свойств виджета **R.id.flThree** (свойства **x**, **y**, **alpha**), что приводит к перемещению виджета с идентификатором **R.id.flThree** одновременно по горизонтали вправо и по вертикали вверх, а также во время перемещения этого виджета меняется его прозрачность (виджет становится более прозрачным). На Рис. 3.5 изображено последовательное применение анимации из Листинга 3.11 в прямом направлении, однако, после завершения анимации, начинается цикл повторения этой анимации в обратном направлении, при котором виджет с идентификатором **R.id.flThree** движется по горизонтали влево, по вертикали вниз и прозрачность этого виджета уменьшается.

Последовательность выполнения анимации в обратном направлении на Рис. 3.5 не отображена, но вы можете самостоятельно увидеть работу этой анимации, когда запустите пример из модуля appb на исполнение. Рекомендуем исполнить этот пример в эмуляторе или реальном устройстве, так как Рис. 3.5 не может в полной степени отобразить исполнение этой анимации.

И последнее, поскольку, как упоминалось выше, для анимации Property Animation создается объект **android.animation.ObjectAnimator**, то существует возможность программно управлять этим объектом или приостанавливать, или прерывать работу этого объекта, с помощью вызова соответствующих методов для этого объекта.

Поскольку класс **android.animation.ObjectAnimator** является наследником класса **android.animation.ValueAnimator**, то методы этих классов будут рассмотрены в следующем разделе данного урока, который посвящен классу **android.animation.ValueAnimator**. Также напомним, что с помощью анимации Property Animation можно модифицировать не все свойства объектов (например, свойства **width** и **height** модифицировать нельзя). Этот недостаток Property Animation устраняется при помощи методов **ofТип()** (**ofInt()**, **ofFloat()**, **ofArgb()**, ...) класса **android.animation.ObjectAnimator**. Применение методов **ofТип()** будет показано в разделе №4 данного урока.

Исходный код примеров, которые были рассмотрены в данном разделе урока, можно найти в модуле appb, среди файлов с исходными кодами, которые прилагаются к данному уроку.

3.2. Value Animation.

Анимация путем изменения значения. Примеры

Value Animation — это анимация, которая меняет некоторое, ни с чем не связанное, значение в течение заданного промежутка времени, от начальной величины и до конечной величины. Разработчику предоставляется возможность получать события изменения величины значения и применять эту величину к любому свойству виджета или виджетов. Файлы с ресурсами Value Animation размещаются в каталоге `/res/animator`. Название файла из каталога `/res/animator` впоследствии используется как идентификатор для получения доступа к ресурсу анимации. Например, для файла `res/animator/filename.xml` будут такие идентификаторы: в Java коде — `R.animator.filename`, в xml коде — `@[package:]animator/filename`.

Отличие Value Animation от Property Animation в том, что Property Animation изменяет конкретное свойство виджета, а Value Animation изменяет величину некоторого значения, и к какому свойству виджета применять это значение выбирает разработчик приложения. Т.е., использование Value Animation предоставляет разработчику приложений больше функциональной гибкости.

Синтаксис файла ресурсов Value Animation приведен в Листинге 3.12.

Листинг 3.12. Синтаксис файла ресурсов Value Animation

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

```
    android:ordering=["together" | "sequentially"]>
<animator
    android:duration="int"
    android:valueFrom="float | int | color"
    android:valueTo="float | int | color"
    android:startOffset="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]
    android:valueType=["intType" | "floatType" |
                       "colorType"]/>
<set>
    ...
</set>
</set>
```

Как видно из Листинга 3.12, корневым элементом для xml файла *Value Animation* является элемент **<set>**. Этот элемент задает набор анимаций, которые могут быть выполнены последовательно или одновременно. Атрибут **android:ordering** элемента **<set>** задает порядок выполнения последовательности анимаций. Значения, которые может принимать атрибут **android:ordering: together** (набор анимаций выполняется одновременно) и **sequentially** (набор анимация выполняется последовательно). Этот атрибут не является обязательным. Значением по умолчанию для этого атрибута является значение **together**.

Дочерними элементами для корневого элемента **<set>** могут быть элементы **<animator>** или **<set>**. Как вы уже догадались, дочерний элемент **<set>** предназначен для объединения нескольких анимаций в набор анимаций для последовательного или параллельного исполнения.

Допускается создание файла ресурсов *Value Animation* с корневым элементом **<animator>**. Такой файл ресурсов *Value Animation* описывает одну единичную анимацию. Синтаксис такого файла приведен в Листинге 3.13.

Листинг 3.13. Синтаксис файла ресурсов Value Animation, который описывает единичную анимацию

```
<?xml version="1.0" encoding="utf-8"?>
<animator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:duration="int"
    android:valueFrom="float | int | color"
    android:valueTo="float | int | color"
    android:startOffset="int"
    android:repeatCount="int"
    android:repeatMode=["repeat" | "reverse"]
    android:valueType=["intType" | "floatType" |
                      "colorType"]
/>
```

Атрибуты элемента **<animator>**:

- **android:valueTo** — обязательный атрибут целого типа, вещественного типа или типа **Color**. Содержит конечное значение для свойства, на котором анимация *Value Animation* должна остановиться. Значение цвета для этого атрибута задаются в виде #RRGGBB.
- **android:valueFrom** — обязательный атрибут целого типа, вещественного типа или типа **Color**. Содержит начальное значение, начиная с которого анимация *Value Animation* начнет выполняться. Значение цвета для этого атрибута задаются в виде #RRGGBB.

- **android:duration** — атрибут целого типа. Содержит длительность анимации в миллисекундах. Атрибут необязательный. Значением по умолчанию является 300 миллисекунд.
- **android:startOffset** — атрибут целого типа. Указывает время задержки перед началом выполнения *Value Animation*. Время указывается в миллисекундах. Атрибут необязательный. Значением по умолчанию является 0 миллисекунд.
- **android:repeatCount** — атрибут целого типа. Указывает количество раз повтора анимации. Если атрибуту присвоить значение -1, то анимация будет исполняться бесконечно. Значение 1 означает, что анимация повторится один раз после выполнения (то есть всего анимация отработает 2 раза). Атрибут необязательный. Значением по умолчанию является значение 0, что означает отсутствие повторений.
- **android:repeatMode** — атрибут целого типа. Указывает на поведение анимации после ее завершения. При этом значение атрибута **android:repeatCount** должно быть установлено в значение -1 или в любое положительное значение. Атрибут **android:repeatMode** может принимать значения: **reverse** — означает, что анимация должна выполниться в обратном порядке от предыдущей итерации, **repeat** — означает, что анимация должна повторяться всегда в прямом порядке (от значения заданного в атрибуте **android:valueFrom** до значения, заданного в атрибуте **android:valueTo**).

- **android:valueType** — указывает тип изменяемого свойства. Допустимые значения: *intType* — свойство целого типа, *floatType* — вещественного, *colorType* — свойство типа Цвет (**Color**).

Для запуска анимации *Value Animation*, которая описывается файлом, синтаксис которого представлен в Листинге 3.13, используется программный код показанный в Листинге 3.14.

Листинг 3.14. Запуск анимации Value Animation, которая представляет собой единичную анимацию, представленную корневым элементом <animator>

```
// ----- First Step -----
// ----- Первый шаг -----
ValueAnimator animator = (ValueAnimator)
    AnimatorInflater.loadAnimator(
        myContext,
        R.anim.value_animator);

// ----- Second step -----
// ----- Второй шаг -----
animator.addUpdateListener(new ValueAnimator.
    AnimatorUpdateListener()
{
    @Override
    public void onAnimationUpdate(ValueAnimator
        animation)
    {
        ...
    }
});

// ----- Start the animation -----
// ----- Запуск анимации на исполнение -----
set.start();
```

Как видно из Листинга 3.14, вначале необходимо создать объект `android.animation.ValueAnimator`, с помощью статического метода `loadAnimator()` класса `android.animation.AnimatorInflater`. Этому методу необходимо передать ссылку на Контекст приложения и идентификатор ресурса *Value Animation*.

Затем полученному объекту `android.animation.ValueAnimator` необходимо назначить обработчик события «изменения величины значения», с помощью вызова метода `addUpdateListener()`. В методе обработчика события «изменения величины значения» `onAnimationUpdate()` необходимо написать программный код, с помощью которого текущая величина, которая изменяется объектом `android.animation.ValueAnimator`, будет применена к какому-либо свойству виджета для достижения эффекта анимации. И после этого можно запускать анимацию *Value Animation* на исполнение с помощью вызова метода `start()`.

Для запуска анимации *Value Animation*, которая описывается файлом, синтаксис которого представлен в Листинге 3.12, используется похожий способ (см. Листинг 3.15), только на основе файла ресурсов мы получим не единичную анимацию в виде объекта `android.animation.ValueAnimator`, а набор анимаций, который находится в объекте `android.animation.AnimatorSet`.

Давайте рассмотрим содержимое Листинга 3.15.

Как видно из Листинга 3.15, вначале необходимо создать объект `android.animation.AnimationSet`, с помощью статического метода `loadAnimator()` класса `android.animation.AnimatorInflater`. Этому методу необходимо

передать ссылку на Контекст приложения и идентификатор ресурса Value Animation. Затем необходимо получить коллекцию типа `ArrayList<Animator>`, содержащую объекты `android.animation.Animator`, с помощью вызова метода `getChildAnimations()` объекта `android.animation.AnimationSet`. Далее, для каждого элемента коллекции необходимо назначить обработчик события «изменения величины значения» вызвав метод `addUpdateListener()`. В методе обработчика события «изменения величины значения» `onAnimationUpdate()` необходимо написать программный код, с помощью которого текущая величина будет применена к какому-либо свойству виджета для достижения эффекта анимации. И после этого можно запускать анимацию Value Animation на исполнение с помощью вызова метода `start()`.

Листинг 3.15. Запуск анимации Value Animation, которая представляет собой набор анимаций `<animator>`, объединенных с помощью элемента `<set>`

```
// ----- First Step -----
// ----- Первый шаг -----
AnimatorSet set = (AnimatorSet) AnimatorInflater.
    loadAnimator(mContext,
        R.anim.value_animator);

// ----- Second step -----
// ----- Второй шаг -----
ArrayList<Animator> arrL = set.getChildAnimations();
// ----- Third step -----
// ----- Третий шаг -----
ValueAnimator vaItem = (ValueAnimator)arrL.
    get(animatorItemIndex);
```

```
vaItem.addValueListener(new ValueAnimator.  
                                AnimatorUpdateListener()  
{  
    @Override  
    public void onAnimationUpdate(ValueAnimator  
                                    animation)  
    {  
        ...  
    }  
});  
// ----- Start the animation -----  
// ----- Запуск анимации на исполнение -----  
set.start();
```

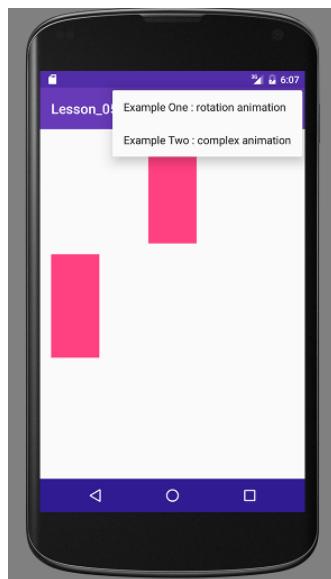


Рис. 3.6. Внешний вид приложения из модуля app7 перед использованием анимаций Value Animation

Давайте рассмотрим примеры применения Value Animation. Для примеров Value Animation создан модуль app7

в проекте Android Studio, который прилагается к данному уроку. В модуле app7 создан **Toolbar**, для которого назначено меню приложения, состоящее из двух пунктов: Example One: rotation animation (применение Value Animation для плавного поворота виджета) и Example Two: complex animation (комплексная Value Animation для одновременного поворота и перемещения виджета). В макете Активности добавлены два виджета **android.widget.FrameLayout**: с идентификаторами **R.id.flOne** и **R.id.flTwo**. Над этими виджетами и будет применяться анимация из перечисленых выше пунктов меню. Внешний вид первоначального приложения с пунктами меню и тулбаром из модуля app7 показан на Рис. 3.6. Виджет **R.id.flOne** находится вверху Активности и выровнен по центру, другой виджет **R.id.flTwo** находится под первым виджетом и выровнен по левому краю с внешним отступом в 16dp.

Первым примером на применение Value Animation будет пример на поворот виджета **R.id.flOne**. В ресурсах приложения модуля app7 создан каталог /res/animator и в этот каталог добавлен файл **rotation_value_animator.xml** (полный путь к файлу относительно модуля: /res/animator/rotation_value_animator.xml). Содержимое файла /res/animator/rotation_value_animator.xml можно увидеть в Листинге 3.16.

Листинг 3.16. Содержимое файла /res/animator/rotation_value_animator.xml

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

```
    android:valueType="floatType"  
    android:startOffset="200"  
    android:duration="1000"  
    android:valueFrom="0"  
    android:valueTo="90" />
```

Как видно из Листинга 3.16, тип изменяемого значения указан в атрибуте **android:valueType="floatType"**, длительность изменения величины значения составляет 1000 миллисекунд (см. атрибут **android:duration**), начальное значение (см. атрибут **android:valueFrom**) равняется 0, конечное значение (см. атрибут **android:valueTo**) равняется 90. Напомним, что с помощью этой анимации мы будем поворачивать виджет **R.id.flOne** вокруг оси Z (направлена перпендикулярно экрану устройства) по часовой стрелке, поэтому начальные и конечные значения являются величинами в градусах.

Запуск на исполнение анимации *Value Animation* из Листинга 3.16 осуществляется в методе **onOptionsItemSelected()**, который является обработчиком событий выбора пунктов меню приложения. Для этого необходимо выбрать пункт меню с надписью *Example One : rotation animation* (идентификатор этого пункта меню **R.id.action_animation_rotation**). Программный код запуска анимации *Value Animation* для пункта меню **R.id.action_animation_rotation** приведен в Листинге 3.17.

Напомним, что особенностью использования анимации *Value Animation* является то, что сам разработчик приложения выбирает, к какому свойству виджета применить изменяемое объектом анимации значение.

Листинг 3.17. Запуск на исполнение анимации Value Animation из файла ресурсов /res/ animator/ rotation_value_animator.xml

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    switch (id)
    {
        case R.id.action_animation_rotation :
        {
            final FrameLayout flOne =
                (FrameLayout) this.
                    findViewById(R.id.flOne);

            ValueAnimator anim = (ValueAnimator)
                AnimatorInflater.loadAnimator(
                    this,
                    R.animator.
                        rotation_value_animator);

            anim.addUpdateListener(
                new ValueAnimator.
                    AnimatorUpdateListener()
            {
                @Override
                public void onAnimationUpdate(
                    ValueAnimator animation)
                {
                    float value =
                        float) animation.
                            getAnimatedValue();
                    flOne.
                        setRotation(value);
                }
            });
            anim.start();
        }
    }
}
```

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

В Листинге 3.17 изменяемое объектом анимации значение применяется для изменения угла поворота виджета (выделено жирным шрифтом). Последовательность исполнения анимации *Value Animation* из Листингов 3.16 и 3.17 изображена на Рис. 3.7.

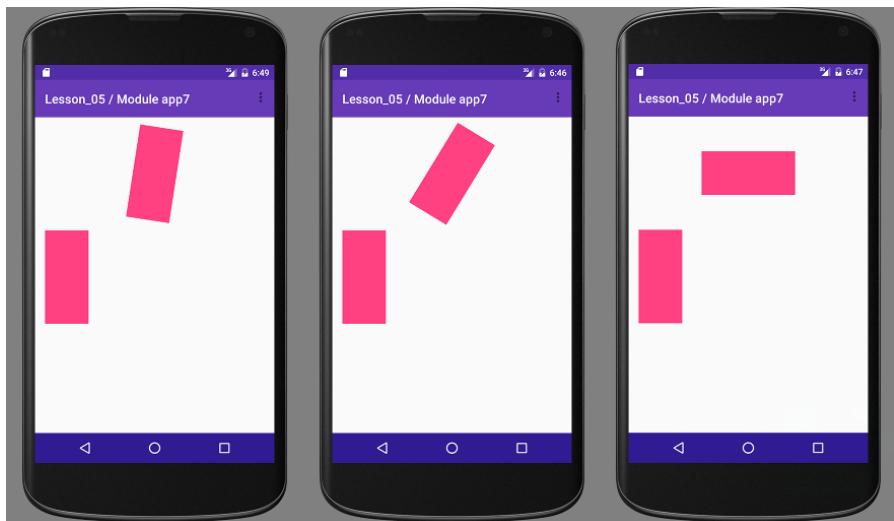


Рис. 3.7. Последовательность исполнения анимации *Value Animation* из Листингов 3.16 и 3.17

Рекомендуем исполнить этот пример в эмуляторе или реальном устройстве, так как Рис. 3.7 не может в полной степени отобразить исполнение этой анимации.

Вторым примером на применение *Value Animation* будет пример на комплексную анимацию, в которой будет изменяться сразу несколько значений.

Эти значения будут применяться к виджету `R.id.flTwo` для поворота виджета и для перемещения виджета одновременно. Для этой анимации создан файл ресурсов `/res/animator/complex_value_animator.xml`. Содержимое этого файла показано в Листинге 3.18.

Листинг 3.18. Содержимое файла
`/res/animator/complex_value_animator.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android=
      "http://schemas.android.com/apk/res/android">
    <animator
        android:valueType="floatType"
        android:duration="1000"
        android:valueFrom="0"
        android:valueTo="90" />

    <animator
        android:valueType="floatType"
        android:duration="1000"
        android:valueFrom="16dp"
        android:valueTo="256dp" />
</set>
```

Как видно из Листинга 3.18, анимация *Value Animation* является набором, который состоит из двух анимаций. Обе анимации имеют длительность 1000 миллисекунд (`android:duration`) и тип изменяемого значения **floatValue** (`android:valueType`). Первая анимация из набора предназначена для изменения значения от 0 до 90. Это значение

будет применяться для угла поворота виджета, следовательно — это величины в градусах. Вторая анимация предназначена для изменения значения от 16dp до 256dp. Это значение будет применяться для позиционирования виджета по горизонтали. Для запуска анимации из Листинга 3.18, необходимо выбрать пункт меню с надписью *Example Two: complex animation* (идентификатор этого пункта меню **R.id.action_animation_complex**). Программный код запуска анимации *Value Animation* для пункта меню **R.id.action_animation_complex** приведен в Листинге 3.19.

Листинг 3.19. Запуск на исполнение анимации Value Animation из файла ресурсов /res/animator/complex_value_animator.xml

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    switch (id)
    {
        ..
        case R.id.action_animation_complex :
        {
            final FrameLayout flTwo =
                (FrameLayout)
                this.findViewById
                (R.id.flTwo);
            AnimatorSet set =
                (AnimatorSet)
                AnimatorInflater.
                loadAnimator(
                this, R.animator.
                complex_value_animator);
            ArrayList<Animator> arrL =
                set.getChildAnimations();
    }
}
```

```
// ----- Add an event handler for the amount of
// ----- rotation -----
// ----- Добавляем обработчик события для величины
// ----- поворота -----
    ValueAnimator vaRotate =
        (ValueAnimator) arrL.get(0);
    vaRotate.addUpdateListener(
        new ValueAnimator.
        AnimatorUpdateListener()
    {
        @Override
        public void
        onAnimationUpdate(ValueAnimator animation)
        {
            float value = (float)animation.
                getAnimatedValue();
            flTwo.setRotation(value);
        }
    );
}

// ----- Add an event handler for the x position
// ----- value -----
// ----- Добавляем обработчик события для величины
// ----- x позиции -----
    ValueAnimator vaX = (ValueAnimator)
        arrL.get(1);
    vaX.addUpdateListener(new ValueAnimator.
        AnimatorUpdateListener()
    {
        @Override
        public void onAnimationUpdate(
            ValueAnimator animation)
        {
            float value = (float)animation.
                getAnimatedValue();
            flTwo.setX(value);
        }
    );
}
```

```
        set.start();
    }
    return true;
}
return super.onOptionsItemSelected(item);
}
```

Как видно из Листинга 3.19, на основе файла ресурсов /res/animator/complex_value_animator.xml, программа получает объект [android.animation.AnimatorSet](#), который представляет собой набор нескольких анимаций *Value Animator* (объекты [android.animation.ValueAnimator](#)). Для каждого объекта из набора необходимо назначить обработчик события «изменения величины значения», с помощью метода [addUpdateListener\(\)](#). Объект, который обрабатывает «события изменения значения» должен реализовать интерфейс [android.animation.ValueAnimator.AnimatorUpdateListener](#), в котором объявлен один метод [onAnimationUpdate\(\)](#). В этом методе необходимо извлечь текущую величину для изменяемого значения и применить ее к необходимому свойству виджета. В Листинге 3.19 программный код применения величин к свойствам виджета **R.id.flTwo** выделен жирным шрифтом. Обратите внимание, что одновременность изменения свойств виджетов осуществляется одновременной работой нескольких разных объектов [android.animation.ValueAnimator](#).

Внешний вид работы примера из Листингов 3.18 и 3.19 изображен на Рис. 3.8. Рекомендуем исполнить этот пример в эмуляторе или реальном устройстве, так как Рис. 3.8 не может в полной степени отобразить исполнение этой анимации.

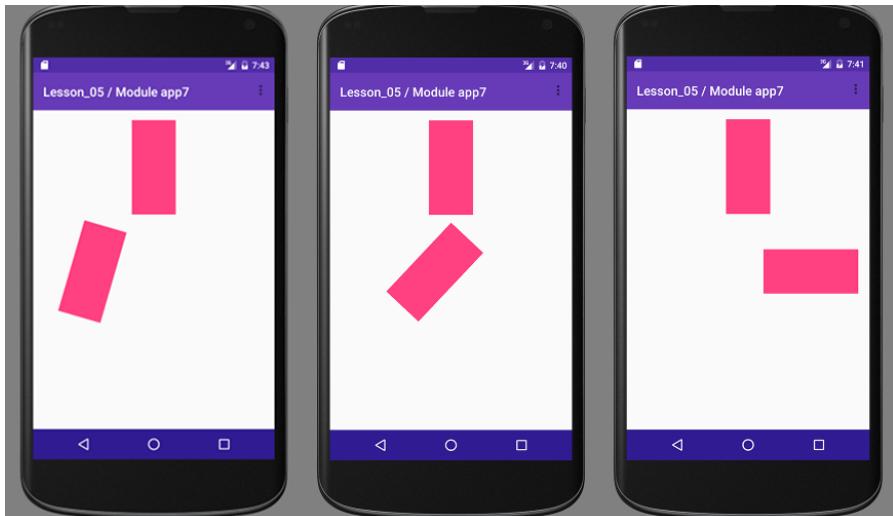


Рис. 3.8. Последовательность исполнения анимации
Value Animation из Листингов 3.18 и 3.19

Рассмотренные примеры на использование анимации *Value Animation* проводились над виджетами **android.widget.FrameLayout**. Конечно же, применять анимацию с тем же результатом можно и к другим виджетам.

Чтобы это продемонстрировать, добавим на Активность модуля *app7* виджет **android.widget.Button** с идентификатором **R.id.btn1**, для которого создадим обработчик события нажатия **btnClick()**. В обработчике события **btnClick()** нажатия на кнопку с идентификатором **R.id.btn1** напишем код, который приведен в Листинге 3.19 за одним исключением: анимация *Value Animation* из файла ресурсов */res/animator/complex_value_animator.xml* будет применяться не к виджету **android.widget.FrameLayout** с идентификатором **R.id.flTwo**, а к виджету **android.widget.Button** с идентификатором **R.id.btn1**. Последовательность

исполнения анимации из Листингов 3.18 и 3.19 для кнопки **android.widget.Button** показана на Рис. 3.9. Как видите, результат тот же.

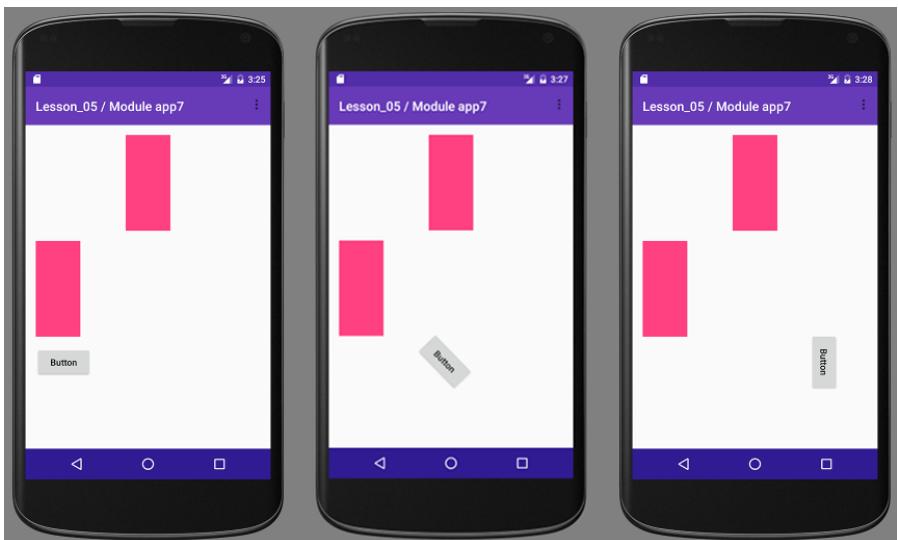


Рис. 3.9. Последовательность исполнения анимации
Value Animation из Листингов 3.18 и 3.19
для виджета **android.widget.Button**

Исходный код примера применения анимации из файла ресурсов `/res/animation/complex_value_animator.xml` для виджета **android.widget.Button** можно найти в модуле *app7*.

В рассмотренных примерах мы научились запускать на исполнение объекты **android.animation.ValueAnimator**, но может возникнуть необходимость приостановить или вообще прервать работу этих объектов. Для этих и других целей в классе **android.animation.ValueAnimator** существуют специальные методы, которые мы сейчас рассмотрим.

Иерархия классов *Value Animation* (`android.animation.ValueAnimator`) и *Property Animation* (`android.animation.ObjectAnimator`):

```
java.lang.Object
|
+--- android.animation.Animator
|
+--- android.animation.ValueAnimator
|
+--- android.animation.ObjectAnimator
```

Теперь подробней рассмотрим методы класса [android.animation.ValueAnimator](#):

- **void cancel()** — прерывает исполнение анимации. В отличии от метода `end()`, анимация, прерванная методом `cancel()`, остается в текущем положении;
- **void end()** — завершает анимацию, при этом значение, которое изменялось анимацией, принимает свою конечную величину;
- **Object getAnimatedValue()** — возвращает текущую величину значения, которое изменяется анимацией;
- **long getDuration()** — возвращает длительность анимации, которая назначена объекту Анимации (атрибут `android:duration`);
- **TimeInterpolator getInterpolator()** — возвращает ссылку на объект Интерполятора, который назначен объекту Анимации. Интерполяторы рассматриваются в одном из следующих разделов урока. По умолчанию, объекту

`android.animation.ValueAnimator` назначен Интерполятором `android.view.animation.AccelerateDecelerateInterpolator`;

- `int getRepeatCount()` — возвращает количество раз повтора анимации (атрибут `android:repeatCount`);
- `int getRepeatMode()` — возвращает режим повтора анимации (атрибут `android:repeatMode`);
- `long getStartDelay()` — возвращает время задержки перед началом исполнения анимации (свойство `android:startOffset`);
- `boolean isRunning()` — возвращает `true`, если анимация исполняется (при этом время задержки `android:startOffset` уже прошло);
- `boolean isStarted()` — возвращает `true`, если анимация исполняется (при этом время задержки `android:startOffset` не учитывается).
- `void pause()` — приостанавливает исполнение анимации. Метод должен быть вызван из того же потока исполнения, в котором анимация была запущена. Для возобновления работы анимации необходимо использовать метод `resume()`;
- `void resume()` — возобновляет ранее приостановленную с помощью вызова метода `pause()` анимацию;
- `void reverse()` — воспроизводит анимацию в обратном направлении. Если анимация в момент вызова этого метода исполняется, то воспроизведение анимации в обратном порядке начинается сразу же от текущей позиции анимации;

- **ValueAnimator setDuration(long duration)** — устанавливает длительность анимации (аналог атрибута `android:duration`) в миллисекундах;
- **void setInterpolator(TimeInterpolator value)** — устанавливает Интерполятор для анимации;
- **void setRepeatCount(int value)** — устанавливает количество раз повторения исполнения анимации (аналог атрибута `android:repeatCount`);
- **void setRepeatMode(int value)** — устанавливает режим повтора анимации (аналог атрибута `android:repeatMode`);
- **void setStartDelay(long startDelay)** — устанавливает время задержки перед началом исполнения анимации (аналог атрибута `android:startOffset`);
- **void start()** — запускает анимацию на исполнение.

3.3. Интерполяторы

Теперь пришло время рассказать о так называемых Интерполяторах. **Интерполятор** — это специальный объект (реализующий интерфейс [`android.animation.TimeInterpolator`](#)), который задает скорость изменения значения с течением времени. Значение может изменяться не только линейно (на одну и ту же величину через равные промежутки времени), оно может изменяться по специальному правилу (например, значение изменяется медленно вначале анимации, плавно переходя в быстрое изменение в конце анимации). Применение объекта Интерполятора «оживляет» анимацию, то есть делает ее более эффектной и интересной, более живой. Существует несколько классов

Интерполяторов, каждый из которых задает свое правило изменения значения с течением времени.

Вот некоторые классы Интерполяторов:

- **android.view.animation.**
AccelerateDecelerateInterpolator — интерполятор, в котором скорость изменения начинается и заканчивается медленно, но ускоряется в середине;
- **android.view.animation.AccelerateInterpolator** — интерполятор, в котором скорость изменения начинается медленно, а затем ускоряется;
- **android.view.animation.AnticipateInterpolator** — интерполятор, в котором изменение значения начинается в обратном от начального значения направлении, но затем значение начинает меняться в прямом направлении до конечного значения.
- **android.view.animation.**
AnticipateOvershootInterpolator — интерполятор, в котором изменение значения начинается в обратном от начального значения направлении, затем значение начинает меняться в прямом направлении до конечного значения, затем перевыполняет конечное значение и, наконец, возвращается к конечному значению;
- **android.view.animation.BaseInterpolator** — абстрактный класс, который является родительским классом для классов интерполяторов;
- **android.view.animation.BounceInterpolator** — интерполятор, в котором изменение «отскакивает» в конце;
- **android.view.animation.CycleInterpolator** — повторяет анимацию заданное количество раз (циклов). Количество

ство циклов передается в качестве параметра в конструктор этого интерполятора. Скорость изменения значения изменяется по синусоиде;

- **android.view.animation.DecelerateInterpolator** — интерполятор, в котором скорость изменения значения начинается быстро и затем замедляется;
- **android.view.animation.LinearInterpolator** — интерполятор, в котором скорость изменения значения постоянна (линейное изменение значения);
- **android.view.animation.OvershootInterpolator** — интерполятор, в котором изменение значения осуществляется от начального значения, достигает конечного значения и перевыполняет конечное значение, затем возвращается к конечному значению;
- **android.view.animation.PathInterpolator** — интерполятор, в котором можно самостоятельно задать правило изменения значения с течением времени с помощью объекта класса **android.graphics.Path**. Объект **android.graphics.Path** передается в конструктор интерполятора как показано в Листинге 3.20:

Листинг 3.20. Создание объекта
android.view.animation.PathInterpolator

```
Path path = new Path();  
path.lineTo(0.25f, 0.25f);  
path.moveTo(0.25f, 0.5f);  
path.lineTo(1f, 1f);  
PathInterpolator P = new PathInterpolator(path);  
...
```

Для того чтобы назначить объекту анимации Интерполятор, необходимо воспользоваться методом:

```
animation_object.setInterpolator(TimeInterpolator  
interpolator);
```

Метод **setInterpolator()** применяет ссылку на объект Интерполятора. Пример назначения Интерполятора для объекта анимации **android.animation.ValueAnimator** из модуля *app7* (см. Листинг 3.17) показан в Листинге 3.21 (код выделен жирным шрифтом).

Листинг 3.21. Назначение Интерполятора объекту анимации android.animation.ValueAnimator

```
@Override  
public boolean onOptionsItemSelected(MenuItem item)  
{  
    int id = item.getItemId();  
    switch (id)  
    {  
        case R.id.action_animation_rotation :  
            final FrameLayout flOne = (FrameLayout)  
                this.findViewById(  
                    R.id.flOne);  
            ValueAnimator anim = (ValueAnimator)  
                AnimatorInflater.  
                loadAnimator(this,  
                    R.animator.  
                    rotation_value_animator);  
anim.setInterpolator(  
    new AnticipateOvershootInterpolator());  
            anim.addUpdateListener(  
                new ValueAnimator.  
                AnimatorUpdateListener()  
                {
```

```
    @Override
    public void onAnimationUpdate(
        ValueAnimator animation)
    {
        float value = (float)animation.
            getAnimatedValue();
        flOne.setRotation(value);
    }
}
anim.start();
}
return true;
..
}
return super.onOptionsItemSelected(item);
}
```

Настоятельно рекомендуем попробовать на примере модуля *app7* (Листинг 3.21) все перечисленные в данном разделе урока объекты Интерполяторов. Эффекты, которые привносят в анимацию Интерполяторы, вам обязательно понравятся.

3.4. View Animation. Примеры

Рассмотрим еще один вид анимации — *View Animation*, которая позволяет выполнять переходы, такие как вращение (*rotating*), перемещение (*moving*), растяжение (*stretching*) и затухание (*fading*).

Файлы xml ресурсов для *View Animation* располагаются в каталоге */res/anim*. Название файла ресурсов */res/anim/filename.xml* используется в качестве идентификатора ресурсов: в Java коде как **R.anim.filename**, в xml коде как **@[package:]anim/filename**.

Файлы xml ресурсов *View Animation* компилируются в объекты **android.view.animation.Animation**.

Отличие этого вида анимации от *Value Animation* и *Property Animation* заключается в том, что этот тип анимации назначается непосредственно виджету при помощи метода **startAnimation()**, как это показано в Листинге 3.22.

Листинг 3.22. Запуск на исполнение анимации View Animation для любого типа виджета

```
SomeView view = (SomeView) findViewById(R.id.  
                  some_view_identifier);  
Animation animation = AnimationUtils.loadAnimation(  
                  this,  
                  R.anim.animation_identifier);  
view.startAnimation(animation);
```

Обратите внимание (см. Листинг 3.22), что для создания объекта анимации **android.view.animation.Animation**, используется статический метод **loadAnimation()** класса **android.view.animation.AnimationUtils**.

Синтаксис xml файла ресурсов View Animation приведен в Листинге 3.23.

Листинг 3.23. Синтаксис xml файла ресурсов View Animation

```
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android=  
      "http://schemas.android.com/apk/res/android"  
      android:interpolator=  
      "@[package:]anim/interpolator_resource"  
      android:shareInterpolator=["true" | "false"] >
```

```
<alpha
    android:fromAlpha="float"
    android:toAlpha="float" />

<scale
    android:fromXScale="float"
    android:toXScale="float"
    android:fromYScale="float"
    android:toYScale="float"
    android:pivotX="float"
    android:pivotY="float" />

<translate
    android:fromXDelta="float"
    android:toXDelta="float"
    android:fromYDelta="float"
    android:toYDelta="float" />

<rotate
    android:fromDegrees="float"
    android:toDegrees="float"
    android:pivotX="float"
    android:pivotY="float" />

<set>
    ...
</set>
</set>
```

Корневым элементом для xml файла ресурсов *View Animation* может быть один из следующих элементов: **<alpha>**, **<scale>**, **<translate>**, **<rotate>**, **<set>**.

Рассмотрим по порядку эти элементы:

- Элемент **<set>**. Этот элемент является контейнером, который предназначен для создания группы элементов

`<alpha>`, `<scale>`, `<translate>`, `<rotate>`, включая вложенные `<set>`. На основе этого элемента создается объект [android.view.animation.AnimationSet](#).

Атрибуты элемента `<set>`:

- **android:interpolator** — содержит идентификатор ресурса Интерполятора, который будет применяться для анимации. Как вы уже знаете, в платформе существуют несколько классов Интерполяторов, но можно создать свой собственный ресурс, описывающий Интерполятор;
- **android:shareInterpolator** — содержит значение типа Boolean. Содержит true, если Интерполятор применяется для всех дочерних элементов.
- Элемент `<alpha>`. Описывает анимации появления (fade-in) и затухания (fade-out). На основе этого элемента создается объект [android.view.animation.AlphaAnimation](#).

Атрибуты элемента `<alpha>`:

- **android:fromAlpha** — содержит значение типа float. Атрибут описывает начальное значение прозрачности. Значение 0.0 соответствует полной прозрачности, а значение 1.0 соответствует полной непрозрачности;
- **android:toAlpha** — содержит значение типа float. Атрибут описывает конечное значение прозрачности. Значение 0.0 соответствует полной прозрачности, а значение 1.0 соответствует полной непрозрачности.
- Элемент `<scale>`. Описывает анимацию изменения размеров. Можно указать центральную точку в виджете

(`android:pivotX` и `android:pivotY`), относительно которой будут меняться размеры виджета, то есть виджет будет как бы расти наружу или внутрь относительно этой точки. Например, если эти значения равны 0, 0 (верхний левый угол), то виджет будет вниз и вправо. На основе этого элемента создается объект [android.view.animation.ScaleAnimation](#).

Атрибуты элемента `<scale>`:

- **`android:fromXScale`** — содержит значение типа float. Начальное значение размера по оси X, где значение 1.0 означает что размер не изменяется;
 - **`android:toXScale`** — содержит значение типа float. Конечное значение размера по оси X, где значение 1.0 означает что размер не изменяется;
 - **`android:fromYScale`** — содержит значение типа float. Начальное значение размера по оси Y, где значение 1.0 означает что размер не изменяется;
 - **`android:toYScale`** — содержит значение типа float. Конечное значение размера по оси Y, где значение 1.0 означает что размер не изменяется;
 - **`android:pivotX`** — содержит значение типа float. X-координата, относительно которой происходит изменение размеров виджета;
 - **`android:pivotY`** — содержит значение типа float. Y-координата, относительно которой происходит изменение размеров виджета.
- Элемент `<translate>`. Описывает вертикальное и/или горизонтальное движение. Для атрибутов этого эле-

мента можно задавать значения в виде: значения от -100 до 100, заканчивающиеся суффиксом «%», указывают процентное значение для перемещения объекта относительно себя; значения от -100 до 100, заканчивающиеся суффиксом «%r», указывают процентное значение перемещения объекта относительно родителя; значение float без суффикса, указывает абсолютное значение. На основе этого элемента создается объект [android.view.animation.TranslateAnimation](#).

Атрибуты элемента <translate>:

- **android:fromXDelta** — начальное значение смещения по X. Если значение указано без суффикса, то оно означает величину в пикселях относительно текущей позиции объекта, если значение указано с суффиксом «%», то оно означает процентное отношение относительно ширины объекта, если значение указано с суффиксом «%r» то оно означает процентное отношение относительно ширины родительского объекта;
- **android:toXDelta** — конечное значение смещения по X. Типы суффиксов для значения атрибута означают то же, что и для атрибута **android:fromXDelta**;
- **android:fromYDelta** — начальное значение смещения по Y. Если значение указано без суффикса, то оно означает величину в пикселях относительно текущей позиции объекта, если значение указано с суффиксом «%», то оно означает процентное отношение относительно высоты объекта, если значение указано с суффиксом «%r», то оно означает

процентное отношение относительно высоты родительского объекта;

- **android:toYDelta** — конечное значение смещения по Y. Типы суффиксов для значения атрибута означают то же, что и для атрибута **android:fromYDelta**.
- Элемент **<rotate>**. Описывает анимацию вращения (поворота). На основе этого элемента создается объект [android.view.animation.RotateAnimation](#).

Атрибуты элемента **<rotate>**:

- **android:fromDegrees** — значение типа float. Начальное значение угла поворота в градусах;
- **android:toDegrees** — значение типа float. Конечное значение угла поворота в градусах;
- **android:pivotX** — Значение или типа float или в процентах. Содержит X-координату центра поворота. Если значение указано в пикселях, то это значение относительно левого края объекта. Если значение указано в процентах с суффиксом «%», то это значение относительно левого края объекта. Если значение указано в процентах с суффиксом «%р», то это значение относительно левого края родительского контейнера;
- **android:pivotY** — значение или типа float или в процентах. Содержит Y-координату центра поворота. Если значение указано в пикселях, то это значение относительно верхнего края объекта. Если значение указано в процентах с суффиксом «%», то это значение относительно верхнего края объекта. Если значение указано в процентах с суффиксом «%р»,

то это значение относительно верхнего края родительского контейнера.

Как уже говорилось выше, элементы `<alpha>`, `<scale>`, `<translate>`, `<rotate>`, `<set>` формируются в объекты производные от класса `android.view.animation.Animation`. Вот иерархия этих классов:

```
java.lang.Object
|
+--- android.view.animation.Animation
|
+--- android.view.animation.AlphaAnimation
+--- android.view.animation.AnimationSet
+--- android.view.animation.RotateAnimation
+--- android.view.animation.ScaleAnimation
+--- android.view.animation.TranslateAnimation
```

В классе `android.view.animation.Animation` объявлены следующие атрибуты, которые наследуются всеми перечисленными выше классами. Следовательно, эти атрибуты являются общими для всех перечисленных элементов.

Давайте рассмотрим эти атрибуты:

- **android:detachWallpaper** — специальный атрибут для анимации окон. Если это окно находится поверх обоев (wallpaper), то окно будет анимироваться без обоев;
- **android:duration** — длительность анимации в миллисекундах;
- **android:fillAfter** — если в атрибут установлено значение `true`, то значение, на котором завершилась

анимация View Animation, остается в виджете, в противном случае — значение принимает начальную величину, которая была у виджета до начала анимации, то есть виджет мгновенно возвращается в свое первоначальное состояние. Этому атрибуту соответствует метод `void setFillAfter(boolean fillAfter)`;

- **android:fillBefore** — когда в атрибут установлено значение true или когда значение атрибута **android:fillEnabled** установлено в false, то преобразование анимации применяется до начала анимации. То есть, виджет принимает начальное значение для анимации и после этого анимация исполняется к конечному значению. Значением по умолчанию для этого атрибута является значение true. Этому атрибуту соответствует метод `void setFillBefore(boolean fillBefore)`;
- **android:fillEnabled** — если значение true, учитывается значение **android:fillBefore**. Этому атрибуту соответствует метод `void setFillEnabled(boolean fillEnabled)`;
- **android:interpolator** — задает интерполятор, используемый для применения эффектов изменения скорости анимации. Этому атрибуту соответствует метод `void setInterpolator(TimeInterpolator interpolator)`;
- **android:repeatCount** — задает количество раз повторения анимации. Этому атрибуту соответствует метод `void setRepeatCount(int repeatCount)`;
- **android:repeatMode** — задает режим повторения анимации для случаев, когда значение атрибута **android:repeatCount** больше нуля. Этому атрибуту соответствует метод `void setRepeatMode(int repeatMode)`;

- **android:startOffset** — задает время задержки в миллисекундах перед началом исполнения анимации. Этому атрибуту соответствует метод **void setStartOffset(int startOffset);**
- **android:zAdjustment** — позволяет настроить Z-упорядочение анимационного контента на время анимации. Этому атрибуту соответствует метод **void setZAdjustment(int zAdjustment).**

Важно! Обратим ваше внимание на следующее: значения атрибутов *View Animation* (*<alpha>*, *<scale>*, *<translate>*, *<rotate>*) принимают не абсолютные, а относительные значения. Например, если размер виджета равен нулю, то увеличение в масштабе в *N* раз даст в результате значение 0 (то есть $0 * N = 0$). Или еще один пример, если значение атрибута *alpha* (прозрачность) для виджета равно 0 (полностью прозрачный), то изменение прозрачности при помощи *View Animation* от *android:fromAlpha="0"* до *android:toAlpha="1"* не приведет к полной непрозрачности виджета, по причине опять же относительности значений (виджет так и останется полностью прозрачным). Возможно, это недоработка разработчиков компании Google, и в более поздних версиях *Android API* это будет изменено, но на текущий момент этот факт необходимо учитывать при построении анимаций *View Animation* над свойствами виджетов, которые имеют нулевые значения.

Теперь перейдем к рассмотрению примеров на использование анимации View Animation. Для этого в проекте, который прилагается к данному уроку создан модуль app8. Для модуля app8 создан **Toolbar**, которому назначено меню приложения. В меню приложения находятся четыре пункта с надписями: Example One: Alpha (для анимации прозрачности), Example Two: Rotate (для анимации поворота), Example Three: Translate (для анимации перемещения), Example Four: Complex (для комплексной анимации). В макете Активности добавлены четыре кнопки **android.widget.Button** с идентификаторами **R.id.btnOne**, **R.id.btnTwo**, **R.id.btnThree**, **R.id.btnFour** и надписями Alpha, Rotate, Translate, Complex. Как вы уже

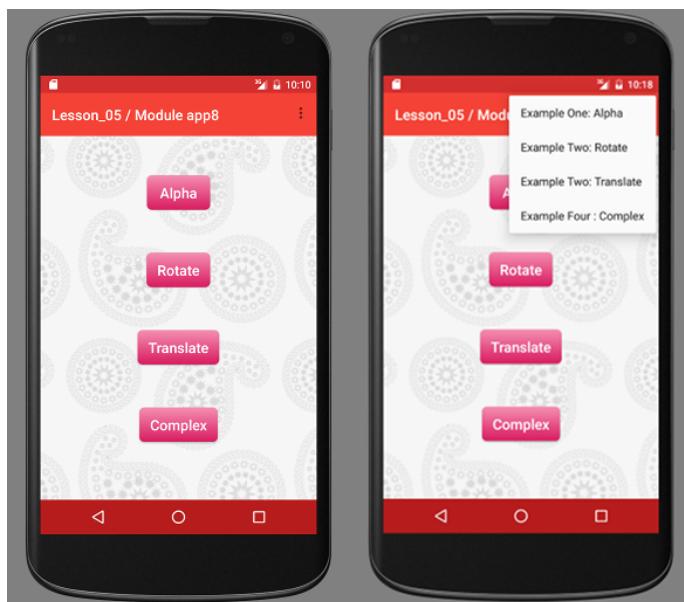


Рис. 3.10. Внешний вид Активности и меню приложения для рассматриваемых примеров модуля app8

догадались — будет четыре примера и каждый виджет **android.widget.Button** предназначен для своего примера.

Кнопкам текущего примера назначен стиль, который описан в Листингах 1.50–1.55 текущего примера (Модуль app4). Для главного контейнера Активности назначен Xml Bitmap Drawable из Листинга 3.3 (модуль app6) для фонового изображения.

Внешний вид Активности из модуля app8, вместе с меню приложения, изображен на Рис. 3.10.

Перед тем как создавать файлы ресурсов View Animation в модуле app8 создан каталог ресурсов /res/anim.

Для первого примера View Animation для прозрачности виджета созданы два файла ресурсов View Animation: /res/anim/view_animation_alpha_forward.xml (для анимации в одном направлении) и /res/anim/view_animation_alpha_backward.xml (для анимации в обратном направлении). Содержимое этих файлов приведено в Листинге 3.24.

Листинг 3.24. Содержимое файлов ресурсов View Animation /res/anim/view_animation_alpha_forward.xml и /res/anim/view_animation_alpha_backward.xml

```
/res/anim/view_animation_alpha_forward.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<alpha
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:fromAlpha="1.0"
    android:toAlpha="0.3"
    android:duration="1000"
/>
```

```
/res/anim/view_animation_alpha_backward.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<alpha
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:fromAlpha="0.3"
    android:toAlpha="1.0"
    android:duration="1000"
/>
```

Два файла для примера анимации прозрачности сделаны с целью, чтобы чередовать изменение alpha-значения от непрозрачности к прозрачности при первом выборе пункта меню *Example One: Alpha* (идентификатор пункта меню **R.id.action_alpha**) и изменение от прозрачности к непрозрачности при втором выборе пункта меню *Example One: Alpha* и так далее. Для того чтобы приложение знало, какой файл ресурсов *View Animation* использовать, в классе Активности (класс **MainActivity**) объявлено поле **isAlphaForward** (см. Листинг 3.25).

Листинг 3.25. Объявление полей в классе Активности модуля app8, которые являются индикаторами направления исполнения анимаций View Animation

```
public class MainActivity extends AppCompatActivity
{
    /**
     * Indicator – in which direction to produce the
     * animation: true – forward, false – backward
     *
     * Индикатор – в какую сторону производить анимацию:
     * true – вперед, false – назад
     */
    private boolean isAlphaForward = true;
```

```

    /**
     * Indicator – in which direction to produce the
     * animation: true – forward, false – backward
     *
     * Индикатор – в какую сторону производить анимацию:
     * true – вперед, false – назад
     */
    private boolean isRotateForward = true;

    /**
     * Indicator – in which direction to produce the
     * animation: true – forward, false – backward
     *
     * Индикатор – в какую сторону производить анимацию
     * true – вперед, false – назад
     */
    private boolean isComplexForward = true;
    ..
}

```

Запуск анимации для примера из пункта меню *Example One: Alpha* осуществляется в методе **onOptionsItemSelected()** класса Активности, который является методом обработчиком событий выбора пунктов меню приложения. Код запуска анимаций из Листинга 3.24 приведен в Листинге 3.26.

Листинг 3.26. Запуск анимаций View Animation из Листинга 3.24.

```

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    switch (id)
    {

```

```
// ----- View Animation for alpha -----
    case R.id.action_alpha :
    {
        Button btnOne = (Button) this.
            findViewById(R.id.btnOne);

// ----- Selecting the direction of the animation:
// ----- Forward / Backward -----
// ----- Выбор направления анимации -
// ----- Вперед / Назад -----
        Animation anim = (this.isAlphaForward) ?
            AnimationUtils.
                loadAnimation(this,
                    R.anim.
                        view_animation_alpha_forward) :
            AnimationUtils.
                loadAnimation(this,
                    R.anim.
                        view_animation_alpha_backward);
        this.isAlphaForward = !this. isAlphaForward;
        anim.setFillAfter(true);
        btnOne.startAnimation(anim);
    }
    return true;
..
}
return super.onOptionsItemSelected(item);
}
```

Обратите внимание, что в Листинге 3.26 для объекта **android.view.animation.Animation** (переменная **anim**) вызывается метод **setFillAfter(true)**. Это делается с целью, чтобы после завершения работы анимации для виджета (кнопка с идентификатором **R.id.btnOne**), который применяет данную анимацию, не вернулось первоначальное значение прозрачности.

Конечно же, можно было не вызывать метод `setFillAfter()`, а воспользоваться атрибутом `android:fillAfter` в xml файле ресурсов, и результат был бы одинаков. Однако, для того чтобы показать разные возможности, которые предоставляются разработчику, в данном примере вместо атрибута `android:fillAfter` в xml коде вызывается метод `setFillAfter()` в Java коде.

Мы уверены, что вы без наших разъяснений смогли понять значения в файлах ресурсов `View Animation /res/anim/view_animation_alpha_forward.xml` и `/res/anim/view_animation_alpha_backward.xml` из Листинга 3.24. Последовательность скриншотов работы примера из Листингов 3.24, 3.26 (для случая анимации в прямом направлении) изображен на Рис. 3.11.

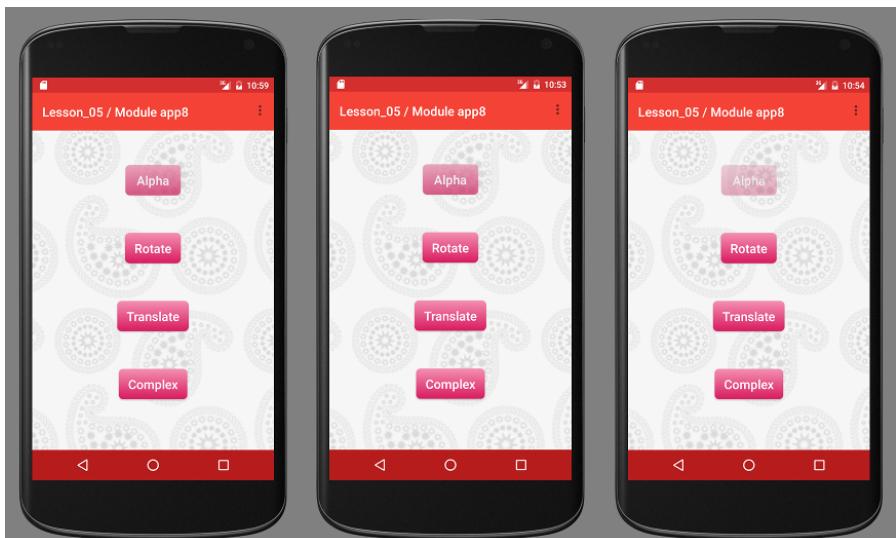


Рис. 3.11. Последовательность скриншотов работы примера из Листингов 3.24 и 3.26

Анимация в обратном направлении работает при повторном выборе пункта меню *Example One: Alpha* и имеет такой же внешний вид, который показан на Рис. 3.11, только в обратном направлении.

Для второго примера View Animation для поворота виджета созданы два файла ресурсов View Animation: /res/anim/view_animation_rotate_forward.xml (для анимации в одном направлении) и /res/anim/view_animation_rotate_backward.xml (для анимации в обратном направлении). Содержимое этих файлов приведено в Листинге 3.27.

Листинг 3.27. Содержимое файлов ресурсов View Animation /res/anim/view_animation_rotate_forward.xml и /res/anim/view_animation_rotate_backward.xml для примера анимации вращения

```
/res/anim/view_animation_rotate_forward.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<rotate
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:duration="2000"
    android:fromDegrees="0"
    android:toDegrees="135"
    android:pivotX="0%"
    android:pivotY="0%" />

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

```
    android:duration="2000"
    android:fromDegrees="135"
    android:toDegrees="0"
    android:pivotX="0%"
    android:pivotY="0%" />
```

Два файла для примера анимации поворота сделаны с целью, чтобы чередовать изменение угла поворота виджета при последовательных выборах пункта меню *Example Two: Rotate* (идентификатор пункта меню `R.id.action_rotate`). Для того, чтобы приложение знало, какой файл ресурсов *View Animation* использовать, в классе Активности (класс `MainActivity`) объявлено поле `isRotateForward` (см. Листинг 3.25).

Запуск примера на анимацию поворота (Листинг 3.27) для пункта меню *Example Two: Rotate* осуществляется в методе `onOptionsItemSelected()` и приведен в Листинге 3.28.

Листинг 3.28. Запуск анимаций View Animation из Листинга 3.27

```
// ----- View Animation for rotation -----
case R.id.action_rotate :
{
    Button btnTwo = (Button) this.
        findViewById(
            R.id.btnTwo);

// ----- Selecting the direction of the animation :
// ----- Forward / Backward -----
// ----- Выбор направления анимации -
// ----- Вперед / Назад -----
```

```
Animation anim = (this.isRotateForward) ?  
    AnimationUtils.  
    loadAnimation(this, R.anim.  
    view_animation_rotate_forward) :  
    AnimationUtils.  
    loadAnimation(this, R.anim.  
    view_animation_rotate_backward);  
this.isRotateForward = !this.  
    isRotateForward;  
anim.setFillAfter(true);  
btnTwo.startAnimation(anim);  
}  
return true;
```

Последовательность скриншотов для примера из Листингов 3.27 и 3.28 (в одном направлении) показана на Рис. 3.12.

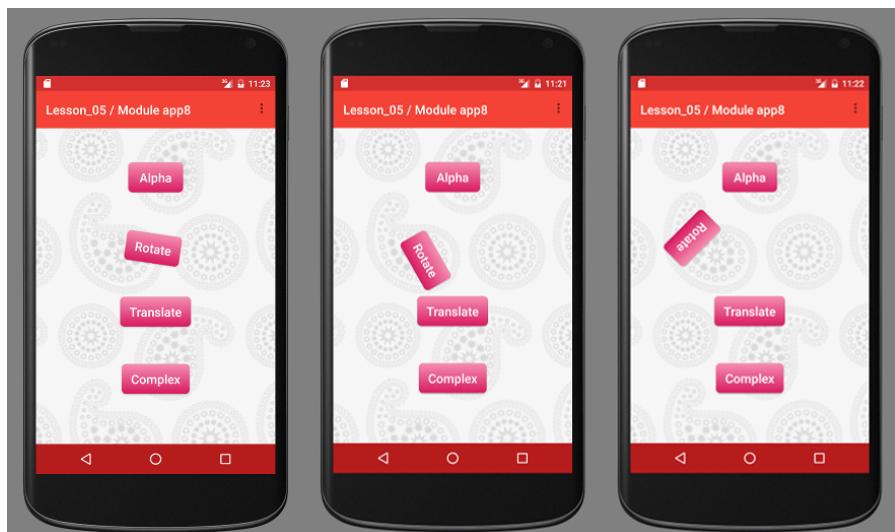


Рис. 3.12. Последовательность скриншотов для примера из Листингов 3.27 и 3.28

Для третьего примера *View Animation* для перемещения виджета создан файл ресурсов *View Animation*: /res/anim/view_animation_translate.xml. В этом примере анимация будет также работать в двух направлениях, только реализация такой работы осуществляется при помощи xml атрибутов **android:repeatMode** и **android:repeatCount**. Содержимое этих файлов приведено в Листинге 3.29.

Листинг 3.29. Содержимое файла ресурсов View Animation
/res/anim/view_animation_translate.xml

```
<?xml version="1.0" encoding="utf-8"?>
<translate
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fillBefore="false"
    android:fromXDelta="0%"
    android:toXDelta="100%p"
    android:repeatCount="1"
    android:repeatMode="reverse" />
```

Обратите внимание (см. Листинг 3.29), что перемещение по горизонтали начинается от текущего положения центра виджета (значение атрибута **android:fromXDelta="0%"**) и происходит до правого края родительского контейнера (значение атрибута **android:toXDelta="100%p"**). При этом перемещаемый виджет полностью «выезжает» за границы родительского контейнера.

Для того, чтобы запустить анимацию перемещения из Листинга 3.29, необходимо в приложении из модуля *app* выбрать пункт меню *Example Three: Translate* (идентификатор

пункта меню R.id.action_translate). Запуск анимации осуществляется в методе `onOptionsItemSelected()` и показан в Листинге 3.30.

Листинг 3.30. Запуск анимации View Animation из Листинга 3.29

```
// ----- View Animation for translate -----
case R.id.action_translate :
{
    Button btnThree = (Button)
        this.findViewById(R.id.btnThree);
    Animation anim =
        AnimationUtils.loadAnimation(this,
            R.anim.view_animation_translate);
    btnThree.startAnimation(anim);
}
return true;
```

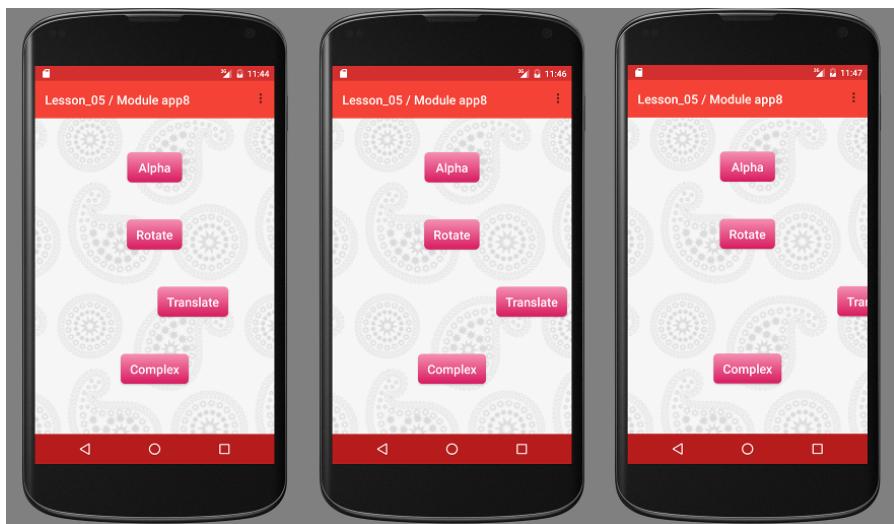


Рис. 3.13. Последовательность скриншотов для примера из Листингов 3.29 и 3.30 (в одном направлении)

Последовательность скриншотов для примера из Листингов 3.29 и 3.30 (в одном направлении) показана на Рис. 3.13.

Последний пример на использование *View Animation* будет пример, применяющий одновременно набор разных анимаций (изменение размеров, поворот, перемещение). Для этого примера созданы два файла ресурсов *View Animation*: /res/anim/view_animation_complex_forward.xml (для анимации в одном направлении) и /res/anim/view_animation_complex_backward.xml (для анимации в обратном направлении). Содержимое этих файлов приведено в Листингах 3.31 и 3.32 соответственно.

Листинг 3.31. Содержимое файла ресурсов View Animation
/res/anim/view_animation_complex_forward.xml
(для анимации в одном направлении)

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:interpolator=
        "@android:anim/accelerate_interpolator"
    android:shareInterpolator="true"
    android:startOffset="200"
    android:duration="1000">

    <rotate
        android:fromDegrees="0"
        android:toDegrees="-765"
        android:pivotX="50%"
        android:pivotY="50%" />
```

```

<translate
    android:fromXDelta="0%"
    android:toXDelta="100%p" />

<scale
    android:fromXScale="1.0"
    android:toXScale="0.0"
    android:fromYScale="1.0"
    android:toYScale="0.0"
    android:pivotX="50%"
    android:pivotY="50%" />

```

</set>

Обратите внимание (см. Листинг 3.31), что поворот виджета будет осуществляться от значения 0 градусов (значение атрибута **android:fromDegrees**) до значения -765 градусов (значение атрибута **android:toDegrees**). Это позволит повернуть виджет на два полных оборота и еще на 45 градусов ($360 \times 2 + 45 = 765$) против часовой стрелки.

Листинг 3.32. Содержимое файла ресурсов View Animation
/res/anim/view_animation_complex_backward.xml
(для анимации в обратном направлении)

```

<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:interpolator=
        "@android:anim/accelerate_interpolator"
    android:shareInterpolator="true"
    android:startOffset="200"
    android:duration="1000">

```

```
<rotate  
    android:fromDegrees="-765"  
    android:toDegrees="0"  
    android:pivotX="50%"  
    android:pivotY="50%" />  
  
<translate  
    android:fromXDelta="100%p"  
    android:toXDelta="0%" />  
  
<scale  
    android:fromXScale="0.0"  
    android:toXScale="1.0"  
    android:fromYScale="0.0"  
    android:toYScale="1.0"  
    android:pivotX="50%"  
    android:pivotY="50%" />  
</set>
```

Сразу сообщим, что комплексная анимация из Листингов 3.31 и 3.32 имеет достаточно интересный вид — с помощью последовательности скриншотов этот вид сложно передать, поэтому мы рекомендуем вам обязательно исполнить эту анимацию в эмуляторе или на реальном устройстве.

Набор анимаций из Листинга 3.31 (прямое направление) дает в результате совместного применения следующий эффект: виджет, врачаясь против часовой стрелки уменьшается в размерах и движется по спирали до полного исчезновения. Набор анимаций из Листинга 3.32 (обратное направление) имеет обратный эффект: виджет появляется и врачаясь по часовой стрелке движется по

спирали увеличивая свой размер до тех пор, пока не вернется в свой начальное положение.

Чтобы приложение знало, какой файл ресурсов использовать (для прямой или для обратной анимации), в классе Активности объявлено специальное поле **isComplexForward** (см. Листинг 3.25). Чтобы запустить пример на комплексную анимацию *View Animation* из Листингов 3.31 и 3.32, необходимо выбрать пункт меню приложения *Example Four: Complex* (идентификатор пункта меню **R.id.action_complex**). Запуск анимации на исполнение осуществляется в методе **onOptionsItemSelected()** и показан в Листинге 3.33.

Листинг 3.33. Запуск анимации из Листингов 3.31 и 3.32

```
// ---- complex View Animation -----
case R.id.action_complex :
{
    Button btnFour = (Button) this.
        findViewById(R.id.btnFour);
    Animation anim = (this.isComplexForward) ?
        AnimationUtils.loadAnimation(this,
            R.anim.view_animation_complex_forward) :
        AnimationUtils.loadAnimation(this,
            R.anim.view_animation_complex_backward);
    this.isComplexForward = !this.isComplexForward;
    anim.setFillAfter(true);
    btnFour.startAnimation(anim);
}
return true;
```

Последовательность скриншотов из Листингов 3.31, 3.32, 3.33 (в одном направлении) изображена на Рис. 3.14.

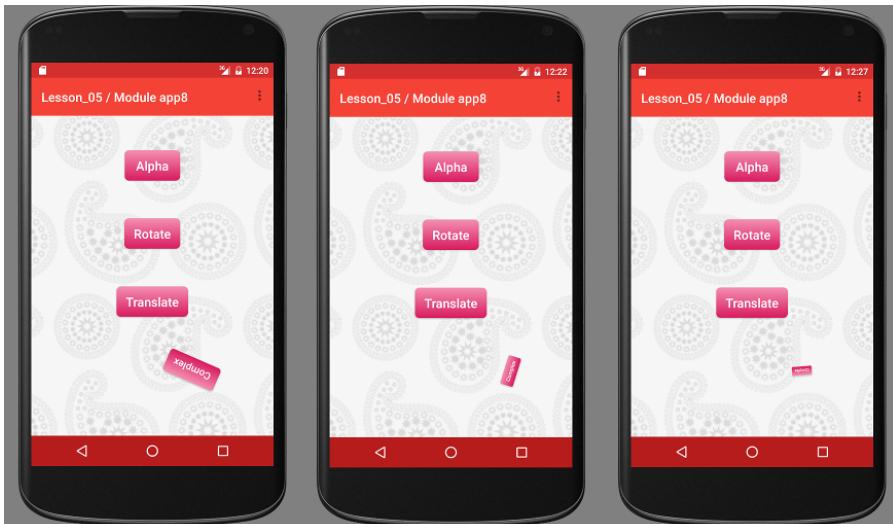


Рис. 3.15. Скриншоты для примеров комплексной анимации из Листингов 3.31, 3.32, 3.33 (в одном направлении)

Исходный код примеров из данного раздела находится в модуле *app8* проекта, среди файлов с исходными кодами, которые прилагаются к данному уроку.

Поскольку рисунки с последовательностью скриншотов не могут полноценно передать работу исполнения анимации, мы рекомендуем вам исполнить примеры текущего раздела в эмуляторе или на реальном устройстве.

3.5. Frame Animation. Анимация путем последовательной смены изображений через указанные промежутки времени. Примеры

Frame Animation предназначена для показа последовательности изображений которые сменяются (чертедуются) одно за другим через определенные промежутки времени (аналогично смене кадров в кинофильме).

Файлы ресурсов Frame Animation располагаются в каталоге /res/drawable. Имя файла ресурсов /res/drawable/filename.xml впоследствии используется в качестве идентификатора при ссылке на ресурс : в Java коде как **R.drawable.filename**, в xml коде как **@[package:]drawable.filename**. На основе файлов ресурсов Frame Animation создаются объекты **android.graphics.drawable.AnimationDrawable**.

Синтаксис файла ресурсов *Frame Animation* приведен в Листинге 3.34.

Листинг 3.34. Синтаксис файла ресурсов Frame Animation

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:oneshot=["true" | "false"] >
    <item
        android:drawable=
            "@[package:]drawable/drawable_resource_name"
        android:duration="integer" />
</animation-list>
```

Как видно из Листинга 3.34, корневым элементом для файла ресурсов *Frame Animation* является элемент **<animation-list>**. Этот элемент должен содержать один или несколько элементов **<item>**. У элемента **<animation-list>** есть атрибут **android:oneshot**, который содержит значение типа **Boolean**. Если в этом атрибуте находится значение *true*, то анимация *Frame Animation* исполняется один раз. Если в этом атрибуте находится значение *false*, то анимация исполняется бесконечно.

Теперь рассмотрим элемент `<item>` и его атрибуты. Элемент `<item>` описывает один «кадр» (фрейм, frame) анимации *Frame Animation*.

Атрибуты элемента `<item>`:

- `android:drawable` — идентификатор изображения из *Drawable Resources*, которое будет отображено в качестве кадра, который описывается элементом `<item>`.
- `android:duration` — длительность показа кадра в миллисекундах.

Как видите, анимация *Frame Animation* несложная в ознакомлении.

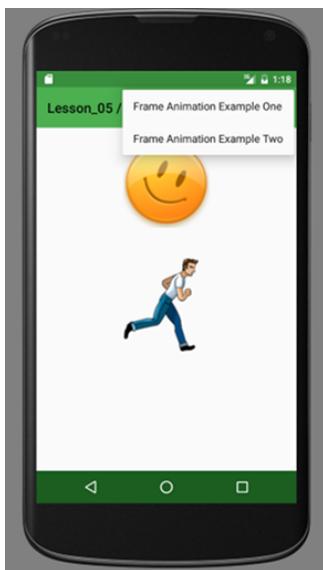


Рис. 3.15. Внешний вид приложения для рассматриваемых в данном разделе примеров

Теперь перейдем к рассмотрению примеров, для которых создан модуль `app9` в проекте, который прилагается

к данному уроку. В модуле app9 создан **Toolbar**, которому назначено меню приложения, состоящее из двух пунктов: Frame Animation Example One и Frame Animation Example Two. На главной Активности приложения размещены два виджета **android.widget.ImageView** с идентификаторами **R.id.imgOne** и **R.id.imgTwo** соответственно. Внешний вид приложения из модуля app9 изображен на Рис. 3.15.

Виджетам **R.id.imgOne** и **R.id.imgTwo** в макете Активности (файл `/res/layout/activity_main.xml`) не назначается никакого изображения (см. Листинг 3.35). Анимация Frame Animation будет назначена этим виджетам программно. Конечно же, можно назначить анимацию Frame Animation этим виджетам и с помощью xml (указав значение атрибута **android:background="frame_animation_filename"**), но анимация как в случае xml, так и в случае Java, не запустится — необходимо программно выполнять специальные команды по ее запуску. А в виджетах, после назначения ресурса анимации, будет неподвижно отображен первый кадр анимации Frame Animation.

Листинг 3.35. «Пустые» виджеты **android.widget.ImageView** в файле с макетом внешнего вида Активности `/res/layout/activity_main.xml` рассматриваемого примера

```
<ImageView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_horizontal"  
    android:id="@+id/imgOne" />
```

```

<Space
    android:layout_width="20dp"
    android:layout_height="20dp"      />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:id="@+id/imgTwo" />

```

В Листинге 3.36 показан фрагмент кода из метода `onCreate()` класса Активности, в котором происходит назначение виджетам `android.widget.ImageView` (идентификаторы `R.id.imgOne` и `R.id.imgTwo`) ресурсов *Frame Animation*.

Листинг 3.36. Программное назначение виджетам `android.widget.ImageView` ресурсов *Frame Animation*

```

@Override
protected void onCreate(Bundle savedInstanceState)
{
    ...
// --- Assigning widgets android.widget.ImageView
// --- background images -----
// --- Назначение виджетам
// --- android.widget.ImageView фоновых изображений
    ImageView imgOne = (ImageView)
                    findViewById(R.id.imgOne);
    imgOne.setBackgroundDrawable(
                    frame_animation_smile);
    ImageView imgTwo = (ImageView)
                    findViewById(R.id.imgTwo);
    imgTwo.setBackgroundDrawable(
                    frame_animation_running_man);
}

```

Еще раз обратим ваше внимание, что после исполнения кода Листинга 3.36, анимация Frame Animation не запустится — в виджетах будет всего лишь отображен первый кадр из последовательности кадров.

Для первого примера были скачаны из Интернета (<http://www.iconsearch.ru/>) два файла с изображениями смайликов, которым даны названия smile_01.png и smile_02.png. Эти файлы помещены в каталог ресурсов /res/drawable (/res/drawable/smile_01.png и /res/drawable/smile_02.png соответственно).

Эти файлы изображений будут представлять представлять кадры анимации Frame Animation для первого примера. Файл ресурсов анимации Frame Animation для первого примера называется /res/drawable/frame_animation_smile.xml.

Содержимое этого файла приведено в Листинге 3.37.

Листинг 3.37. Содержимое файла ресурсов Frame Animation
/res/drawable/frame_animation_smile.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:oneshot="false">

    <item
        android:drawable="@drawable/smile_01"
        android:duration="350" />

    <item
        android:drawable="@drawable/smile_02"
        android:duration="350" />
</animation-list>
```

Как видно из Листинга 3.37, в анимации участвуют два кадра, длительность показа каждого кадра составляет 350 миллисекунд. Кадрами для анимации являются растровые изображения /res/drawable/smile_01.png и /res/drawable/smile_02.png. Чтобы запустить анимацию *Frame Animation* из Листинга 3.36, необходимо выбрать пункт меню приложения *Frame Animation Example One* (идентификатор пункта меню **R.id.action_frame_animation_one**).

Программный код запуска анимации Frame Animation из Листинга 3.37, приведен в Листинге 3.38 и находится в методе класса Активности **onOptionsItemSelected()**.

Листинг 3.38. Запуск анимации Frame Animation из Листинга 3.37 для виджета android.widget.ImageView с идентификатором R.id.imgOne

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    int id = item.getItemId();
    switch (id)
    {
        case R.id.action_frame_animation_one :
        {
            ImageView imgOne = (ImageView)
                findViewById(R.id.imgOne);
            AnimationDrawable smileAnimation =
                (AnimationDrawable)
                    imgOne.getBackground();

            if (smileAnimation.isRunning())
            {
                smileAnimation.stop();
            }
        }
    }
}
```

```
        else
        {
            smileAnimation.start();
        }
    }
    return true;
...
}
return super.onOptionsItemSelected(item);
}
```

Как видно из Листинга 3.38, вначале необходимо получить ссылку на объект [android.graphics.drawable.AnimationDrawable](#), при помощи вызова метода `getBackground()` виджета `android.widget.ImageView`.

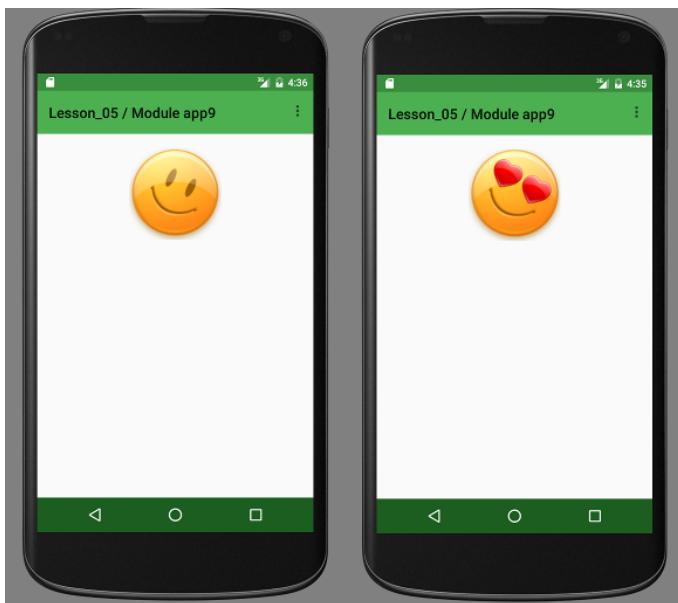


Рис. 3.16. Последовательность скриншотов работы примера из Листингов 3.37 и 3.38

Далее происходит проверка — исполняется ли в данный момент анимация Frame Animation или нет, с помощью вызова метода `isRunning()`. Если анимация исполняется, то она останавливается при помощи вызова метода `stop()`, если анимация не исполняется — то она запускается при помощи вызова метода `start()`. Последовательность скриншотов работы примера из Листингов 3.37 и 3.38 показана на Рис. 3.16.

Получилась легкая и незатейливая анимация, и это привело к мысли сделать более живую анимацию, в которой бы некоторый персонаж двигался, совершая большое количество движений. Такую идею реализовать при помощи анимации *Frame Animation* также не сложно, как и анимацию из первого примера. В помощь пригодятся так называемые спрайты (*sprites*). О том, что это такое *sprite* можно прочитать на сайте <http://wikipedia.org>.



Рис. 3.17. Спрайт с движущимся персонажем, из которого будут вырезаны отдельные изображения для кадров анимации Frame Animation для второго примера

Конечно же, мы не будем использовать спрайты так же, как их используют например, при web-разработке. В нашем следующем примере на основе спрайта, загруженного из интернета, мы сделаем несколько кадров, которые и будем использовать для анимации *Frame Animation*.

На Рис. 3.17 изображен спрайт, который послужил источником для кадров анимации *Frame Animation* нашего следующего примера. Нам пришлось немного потрудиться, чтобы с помощью графического редактора вырезать из этого спрайта необходимые фрагменты изображения. Для нашего второго примера взяты первые восемь изображений (из верхней строки спрайта). Размер каждого изображения 128×170 пикселей. Полученные файлы с отдельными рисунками получили названия man_01.png, man_02.png, ... man_08.png. Эти файлы были помещены в каталог /res/drawable модуля app9. И затем был создан файл ресурсов *Frame Animation* /res/drawable/frame_animation_running_man.xml. Содержимое этого файла показано в Листинге 3.39.

Листинг 3.39. Содержимое файла ресурсов *Frame Animation* /res/drawable/frame_animation_running_man.xml

```
<?xml version="1.0" encoding="utf-8"?>
<animation-list
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:oneshot="false">

    <item
        android:drawable="@drawable/man_01"
        android:duration="100" />
```

```
<item
    android:drawable="@drawable/man_02"
    android:duration="100" />

<item
    android:drawable="@drawable/man_03"
    android:duration="100" />

<item
    android:drawable="@drawable/man_04"
    android:duration="100" />

<item
    android:drawable="@drawable/man_05"
    android:duration="100" />

<item
    android:drawable="@drawable/man_06"
    android:duration="100" />

<item
    android:drawable="@drawable/man_07"
    android:duration="100"
    />

<item
    android:drawable="@drawable/man_08"
    android:duration="100" />

</animation-list>
```

Как видно из Листинга 3.39, принцип создания анимации для движущегося персонажа точно такой же, как и в Листинге 3.37, только изображений (фигурок) больше и время показа каждого изображения составляет 100 миллисекунд. Для того чтобы запустить анимацию

Frame Animation из Листинга 3.39 на исполнение, необходимо в приложении выбрать пункт меню *Frame Animation Example Two* (идентификатор пункта меню **R.id.action_frame_animation_two**). Java код запуска этой анимации аналогичен коду запуска анимации из Листинга 3.38 и приведен в Листинге 3.40.

Листинг 3.40. Запуск и остановка анимации Frame Animation из Листинга 3.39

```
case R.id.action_frame_animation_two :  
{  
    ImageView imgTwo = (ImageView)  
        findViewById(R.id.imgTwo);  
    AnimationDrawable manAnimation = (  
        AnimationDrawable) imgTwo.  
        getBackground();  
  
    if (manAnimation.isRunning())  
    {  
        manAnimation.stop();  
    }  
    else  
    {  
        manAnimation.start();  
    }  
}  
return true;
```

Анимация *Frame Animation* из Листинга 3.39 выглядит интересной и мотивирует к созданию какого-нибудь игрового приложения, которое можно было бы выложить в *Google Play* (магазин приложений, игр, книг,

музыки и фильмов компании *Google*). Об этом магазине можно прочитать в Википедии на [страничке](#). Адрес магазина приложений *Google Play*: <https://play.google.com/store?hl=ru>. Мы уверены, что у вас все получится и ждем вам творческих успехов!

Последовательность скриншотов работы примера из Листингов 3.39 и 3.40 изображена на Рис. 3.18.

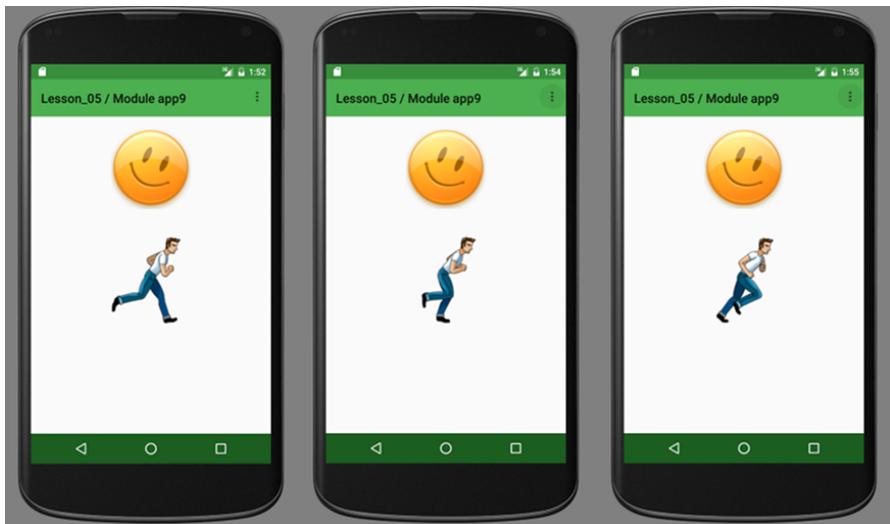


Рис. 3.18. Последовательность скриншотов работы примера анимации Frame Animation из Листингов 3.39 и 3.40

Рекомендуем вам обязательно исполнить примеры из данного раздела урока в эмуляторе или на реальном устройстве, так как последовательности скриншотов не могут передать всей полноты работы примеров.

Исходные коды примеров из данного раздела можно найти в модуле *app9* проекта, среди файлов с исходными кодами, которые прилагаются к данному уроку.

4. Практический пример приложения с использованием анимации

В этом уроке были рассмотрены разные виды анимации, которые позволяют разработчикам Android-приложений создавать динамические, отзывчивые и привлекательные приложения. Также были рассмотрены множество ознакомительных примеров для каждого вида анимации. Теперь, когда проделаны первые шаги в освоении анимации, пришло время увидеть, как можно применять анимацию в реальных приложениях с графическим пользовательским интерфейсом.

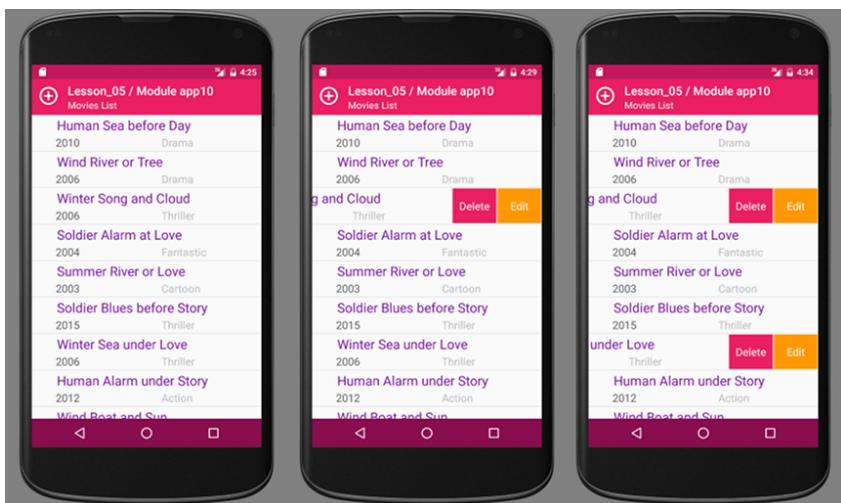


Рис. 4.1. Выдвигающаяся панель с опциями для операций удаления и редактирования элемента списка android.widget.ListView

Для рассматриваемого в данном разделе примера было выбрано приложение со списком `android.widget.ListView`. Как вы знаете, многие приложения, базирующиеся на работе со списками, предоставляют пользователю функциональность для добавления новых элементов списка, удалению или редактированию существующих элементов списка. В предыдущих уроках этого курса мы рассматривали с вами такие примеры. И применение анимации в подобных приложениях выводит их на новый, более высокий уровень. Суть рассматриваемого в данном разделе примера заключается в том, что при выборе любого элемента списка `android.widget.ListView` для этого элемента будет появляться специальная панель (набор виджетов) с опциями *Delete* («Удалить») и *Edit* («Редактировать»). Эта специальная панель, при нажатии пользователем на элемент списка, будет «выезжать», сдвигая при этом панель, представляющую элемент списка. Идея появления панели изображена на Рис. 4.1.

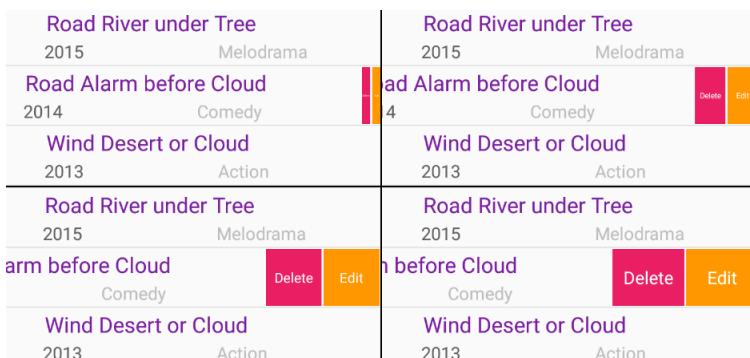


Рис. 4.2. Последовательность скриншотов, показывающая анимацию появления панели операций рассматриваемого в данном разделе примера

Как видно из Рис. 4.1, в примере можно одновременно для нескольких элементов списка активизировать панели с опциями для операций удаления и редактирования. Последовательность скриншотов, показывающую анимацию появления панели операций, изображена на Рис. 4.2. Конечно же, для динамики, панель операций должна появляться достаточно быстро, чтобы пользователь не почувствовал, что ему приходится ждать. В нашем примере время появления панели операций составляет 300 миллисекунд. Хотим напомнить, что компания *Google* настойчиво продвигает концепцию по созданию и поведению внешнего вида пользовательского интерфейса, которая называется *Material Design*. Описание этой концепции можно найти по адресу <https://material.io/guidelines>. Мы рекомендуем вам методично и регулярно выделять немного времени на изучение этой концепции. Так вот, величину времени появления панели операций в рассматриваемом примере (Рис. 4.1) мы подобрали исходя из рекомендаций *Material Design*.

Далее, если пользователь выберет опцию *Delete* или *Edit* или нажмет на элемент списка, то панель операций плавно исчезнет в обратном показанному на Рис. 4.2 порядке. Функциональность добавления, удаления и редактирования элементов списка в рассматриваемом примере не реализована потому, что мы хотим сконцентрироваться только на применении анимации.

Отдельно хотим акцентировать ваше внимание на том, что для анимации выбран вариант, при котором панель элемента списка сдвигается при появлении панели операций (см. Рис 4.2). Конечно же, нами рассматривался

вариант, при котором панель элемента списка остается на месте при появлении панели операций, как это показано на Рис. 4.3.

Movie 3 2002	Genre 3	
Movie 4 2003	Genre 4	
Movie 5 2004	Genre 5	

Рис. 4.3. Вариант появления панели операций, который был отвергнут

Вариант, показанный на Рис. 4.3, был отвергнут по следующим причинам: во-первых, он достаточно прост для реализации, а в данном уроке и так достаточно много простых примеров. Во-вторых, такой вариант не выглядит красиво для случая, когда элемент списка имеет длинное название (см. Рис. 4.4), так как содержимое панели элемента списка съезжает вниз и нарушает внешний вид панели элемента списка.

Winter Song and Sea 3 2002	Love story 3	
Winter Song and Sea 4 2003	Love story 4	
Winter Song and Sea 5 2004	Love story 5	

Рис. 4.4. Визуальный недостаток отвергнутого варианта появления панели операций – нарушение внешнего вида элемента списка при появлении панели операций

Мы не спроста рассказали вам о варианте из Рис. 4.3 и Рис. 4.4. Выбранный нами вариант из Рис. 4.2 требует

нестандартного решения при создании внешнего вида панели элемента списка, о чём будет рассказано ниже.

Также хотим сообщить вам, что в примере данного раздела для реализации анимации из Рис. 4.2 мы по отдельности рассмотрим применение таких анимаций: *Value Animation*, *View Animation*, *Property Animation*. Это делается с целью увидеть различия в реализации примера между этими видами анимаций, а также сравнить достоинства и недостатки каждого из этих видов анимаций.

Все три вида анимаций будут рассмотрены в одном приложении. Для этого, в проекте Android Studio, который прилагается к данному уроку, создан модуль *app10*.

4.1. Вводное описание общих частей практического примера

Перед тем, как начать рассмотрение применения каждого вида анимации, давайте рассмотрим общую часть реализации приложения из модуля *app10*. Как вы уже увидели, в качестве списка для *android.widget.ListView* выбран список фильмов. Для того чтобы представлять каждый элемент списка в нашем модуле создан класс *MyMovie*. Содержимое этого класса приведено в Листинге 4.1.

Листинг 4.1. Класс *MyMovie*, который представляет элементы списка *android.widget.ListView*

```
/**  
 * Class MyMovie  
 * - - - - -  
 */  
class MyMovie  
{
```

```
// ----- Class members -----
/** 
 * Movie title
 *
 * Название фильма
 */
public String title;

/** 
 * Movie genre
 *
 * Жанр фильма
 */
public String genre;

/** 
 * Year of movie release
 *
 * Год выпуска фильма
 */
public int year;

// ----- Class methods -----
public MyMovie(String title, String genre, int year)
{
    this.title = title;
    this.genre = genre;
    this.year = year;
}

@Override
public String toString()
{
    return "Title : " + this.title + ", Genre : " +
           this.genre + ", Year : " + this.year;
}
}
```

Как видно из Листинга 4.1, класс **MyMovie** состоит из трех полей: название фильма, жанр фильма и год выпуска фильма. То есть в каждом элементе списка **android.widget.ListView** будет три поля для отображения.

Наверное вы также обратили внимание на абсурдные названия фильмов на Рис. 4.1–4.4. Разумеется, эти названия не придумывались, а были получены с помощью функции, которая специально была написана для данного примера. Задача этой функции — генерировать случайные названия фильмов таким образом, чтобы названия фильмов были приближены к реальным названиям. Код этой функции представлен в Листинге 4.2. Функция называется **makeMovieTitle()** и принадлежит классу Активности **MainActivity** примера из модуля *app10*.

Листинг 4.2. Функция, которая генерирует случайные названия для фильмов из рассматриваемого примера

```
private String makeMovieTitle()
{
    String[] first =
    {
        "Winter", "House", "Summer", "Road",
        "Human", "Wind", "Soldier"
    };

    String[] second =
    {
        "Blues", "Alarm", "Song", "Sea",
        "River", "Boat", "Desert"
    };
    String[] third =
    {
        "Tree", "Cloud", "Sun", "Day",
        "Year", "Story", "Love"
    };
}
```

```

String[] z =
{
    " and ", " or ", " at ", " under ",
    " before ", " after "
};

String str = first[(int)
    (Math.random() * first.length)] + " " +
second[(int)(Math.random() *
second.length)] + z[(int)
(Math.random() * z.length)] +
third[(int)(Math.random() *
third.length)];
return str;
}

```

Как видно из Листинга 4.2, случайные названия для фильмов получаются на основе нескольких массивов, содержащих слова. Из этих слов, которые выбираются из массивов случайно, и скрепляется название фильма.

Далее, xml верстка макета Активности показана в Листинге 4.3.

Листинг 4.3. Xml верстка макета Активности рассматриваемого примера

```

<?xml version="1.0" encoding="utf-8"?>
<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:orientation="vertical"
    tools:context="itstep.com.myapp8.MainActivity">

```

```
<android.support.v7.widget.Toolbar  
    android:id="@+id/toolbar"  
    android:layout_width="match_parent"  
    android:layout_height="60dp"  
    android:background="#E91E63"  
    android:subtitleTextColor="#FFFFFF"  
    android:titleTextColor="#FFFFFF" />  
  
<FrameLayout  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:background="#F48FB1"  
    android:id="@+id/flAddPanel" >  
</FrameLayout>  
  
<ListView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/lvMovies">  
</ListView>  
  
</LinearLayout>
```

Как видно из Листинга 4.3, в макете Активности присутствуют: тулбар `android.support.v7.widget.Toolbar` с идентификатором `R.id.toolbar`, список `android.widget.ListView` с идентификатором `R.id.lvMovies` и виджет `android.widget.FrameLayout` с идентификатором `R.id.flAddPanel`. Сразу сообщим, что виджет `R.id.flAddPanel` в данном примере не используется, а является заготовкой для вашего домашнего задания. Размер по высоте виджета `R.id.flAddPanel` равен `0dp`, то есть виджет не виден на Активности. Виджет `R.id.lvMovies` предназначен для отображения списка фильмов (объектов `MyMovie`). Это

именно тот виджет, к элементам которого будет применяться анимация. Виджет тулбар `R.id.toolbar` так же в нашем примере не используется (за исключением текстов заголовка и подзаголовка), однако, на нем отображается кнопка навигации (см. Рис. 4.1), к которой прикреплен обработчик события нажатия и эта кнопка, так же как и виджет `R.id.flAddPanel`, будет использоваться вами при выполнении домашнего задания.

Как уже сообщалось выше, приложение из модуля `app10` будет демонстрировать появление панели операций для элементов списка с помощью трех видов анимаций: Value Animation, View Animation, Property Animation. Необходимо будет переключать работу примера на применение нужного вида анимации перед сборкой и запуском приложения в эмуляторе или на реальном устройстве. Такая возможность реализована с помощью специального статического поля класса Активности `MainActivity`. Это поле называется `animationType` и имеет тип `int`. Для этого поля в классе `MainActivity` созданы статические константы, содержащие код для каждого вида анимации: `TYPE_VALUE_ANIMATION` (для Value Animation), `TYPE_VIEW_ANIMATION` (для View Animation), `TYPE_PROPERTY_ANIMATION` (для Property Animation). Объявление статического поля `animationType` и перечисленных выше констант показано в Листинге 4.4. Для того чтобы запустить приложение с демонстрацией применения требуемого вида анимации, необходимо в статическое поле `animationType` записать значение соответствующей константы, как это показано в Листинге 4.4.

Листинг 4.4. Объявление статического поля animationType и статических констант для переключения работы примера на применение соответствующего вида анимации

```
public class MainActivity extends AppCompatActivity
{
    // ----- Class constants -----
    /**
     * The type of animation in the example:
     * "Value Animation"
     * Тип рассматриваемой анимации в примере:
     * "Value Animation"
     */
    private final static int TYPE_VALUE_ANIMATION = 0;

    /**
     * The type of animation in the example:
     * "View Animation"
     * Тип рассматриваемой анимации в примере :
     * "View Animation"
     */
    private final static int TYPE_VIEW_ANIMATION = 1;

    /**
     * The type of animation in the example:
     * "Property Animation"
     * Тип рассматриваемой анимации в примере:
     * "Property Animation"
     */
    private final static int TYPE_PROPERTY_ANIMATION = 2;

    // ----- Class static members -----
    /**
     * A static constant that contains the type of
     * animation that will be applied to
     * the demonstration in the application.
     * The value of this constant must be manually
     */
```

```

* changed before compiling the application.
* Статическая константа, которая содержит тип
* анимации, которая будет применяться для
* демонстрации в приложении. Значение этой
* константы нужно менять перед компиляцией
* приложения.
*/
private final static int animationType =
        MainActivity.TYPE_VALUE_ANIMATION;

// ----- Class members -----
/***
 * Movies list
 * Список фильмов
 */
private ListView lvMovies;
...
}

```

Инициализация тулбара R.id.toolbar, списка R.id.lvMovies вместе с Адаптером данных android.widget.ArrayAdapter<MyMovie> осуществляется в методе onCreate() класса Активности и показана в Листинге 4.5.

Листинг 4.5. Инициализация тулбара R.id.toolbar и списка R.id.lvMovies вместе с Адаптером данных android.widget.ArrayAdapter<MyMovie>

```

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

```

```
// ----- Toolbar -----
Toolbar toolbar = (Toolbar) findViewById
    (R.id.toolbar);
setSupportActionBar(toolbar);

toolbar.setTitleTextColor(Color.WHITE);
toolbar.setSubtitleTextColor(Color.WHITE);
toolbar.setSubtitle("Movies List");
toolbar.setNavigationIcon(R.mipmap.
    ic_add_circle_outline_white_36dp);

toolbar.setNavigationOnClickListener(new
    View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        Log.d("#####", "Navigation Icon Click");
    }
});

// ----- Initializing the lvMovies field -----
// ----- Инициализация поля lvMovies -----
this.lvMovies = (ListView) this.
    findViewById(R.id.lvMovies);

// ----- Create a movie collection -----
// ----- Создание коллекции фильмов -----
String[] genres =
{
    "Action", "Fantastic", "Drama",
    "Melodrama",
    "Comedy", "Adventure", "Cartoon",
    "Thriller", "LoveStory"
};
```

4. Практический пример приложения с использованием анимации

```
ArrayList<MyMovie> movies = new ArrayList<>();
for (int i = 0; i < 50; i++)
{
    movies.add(new MyMovie(this.makeMovieTitle(),
        genres[(int) (Math.random() *
        genres.length)],
        2000 + (int) (Math.random() * 17)));
}

// ----- Selecting the Layout Resource File for the
// ----- ListView Item -----
// ----- Выбор файла макета раскладки виджетов для
// ----- элемента списка -----
int movieListItemLayoutId;
switch (MainActivity.animationType)
{

// ----- List item for "Value Animation" -----
    case MainActivity.TYPE_VALUE_ANIMATION :
        default :
            movieListItemLayoutId =
                R.layout.list_item_value_animation;
        break;

// ----- List item for "View Animation" -----
    case MainActivity.TYPE_VIEW_ANIMATION :
        movieListItemLayoutId =
            R.layout.list_item_view_animation;
        break;

// ----- List item for "Property Animation" -----
    case MainActivity.TYPE_PROPERTY_ANIMATION :
        movieListItemLayoutId =
            R.layout.list_item_value_animation;
        break;
}
```

```
// ----- Create a Data Adapter and assign it to the
// ----- lvMovies list -----
// ----- Создание адаптера данных и назначение его
// ----- списку lvMovies -----
ArrayAdapter<MyMovie> adapter =
    new ArrayAdapter<MyMovie>(this,
        movieListLayoutId,
        R.id.tvTitle, movies)
{
    @Override
    public View getView(int position,
        View convertView, ViewGroup parent)
    {
        // ----- Receiving a widget that represents the
        // ----- current list item -----
        // ----- Получение виджета, который представляет
        // ----- текущий элемент списка -----
        View view = super.getView(position,
            convertView, parent);

        // ----- Obtaining a link to the MyMovie object that
        // ----- is displayed in this widget -----
        // ----- Получение ссылки на объект MyMovie, который
        // ----- отображается в этом виджете -----
        MyMovie F = this.getItem(position);

    /*
     * Getting links to TextView widgets to display
     * information about the movie in them
     * Получение ссылок на виджеты TextView для
     * отображения в них информации о фильме
     */
        TextView tvTitle = (TextView) view.
            findViewById(R.id.tvTitle);
        TextView tvGenre = (TextView) view.
            findViewById(R.id.tvGenre);
```

```
        TextView tvYear = (TextView) view.  
            findViewById(R.id.tvYear);  
  
        // ----- Writing to widgets TextView information  
        // ----- about the current movie -----  
        // ----- Запись в виджеты TextView информации  
        // ----- о текущем фильме -----  
            tvTitle.setText(F.title);  
            tvGenre.setText(F.genre);  
            tvYear.setText(String.  
                valueOf(F.year));  
  
        // ----- Hide operations panel if necessary -----  
        // ----- Спрятать панель операций, если нужно -----  
  
        // ----- Return the list item view -----  
            return view;  
        }  
    };  
  
    // ----- Assigning the Data Adapter to the  
    // ----- android.widget.ListView -----  
    // ----- Назначение Адаптера Данных списку  
    // ----- android.widget.ListView -----  
    this.lvMovies.setAdapter(adapter);  
  
    // ----- Assigning a click event to a ListView item  
    // ----- Назначение события нажатия по элементу  
    // ----- списка ListView -----  
    this.lvMovies.setOnItemClickListener(  
        new AdapterView.OnItemClickListener()  
    {  
        @Override  
        public void onItemClick(AdapterView<?>  
            adapterView, View view, int i, long l)  
    {
```

```
        switch (MainActivity.animationType)
        {
// ----- Demonstrates the use of "Value Animation"
// ----- Демонстрируется применение "Value Animation"
            case MainActivity.TYPE_VALUE_ANIMATION :
                default :
                    MainActivity.this.
                        toggleValueAnimation(view);
                    break;

// ----- Demonstrates the use of "View Animation"
// ----- Демонстрируется применение "View Animation"
            case MainActivity.TYPE_VIEW_ANIMATION :
                MainActivity.this.
                    toggleViewAnimation(view);
                break;

// ----- Demonstrates the use of "Property
// ----- Animation" -----
// ----- Демонстрируется применение "Property
// ----- Animation" -----
            case MainActivity.
                TYPE_PROPERTY_ANIMATION :
                MainActivity.this.
                    togglePropertyAnimation(view);
                break;
        }
    });
}
```

Как видно из Листинга 4.5, инициализация тулбара **R.id.toolbar** затрагивает назначение тулбару цвета текста для заголовка и подзаголовка, назначение тулбару текста подзаголовка (*Movies List*), назначение тулбару иконки

навигации (идентификатор ресурса `R.mipmap.ic_add_circle_outline_white_36dp`), а также назначение иконке навигации обработчика события нажатия `android.view.View.OnClickListener`. В обработчике события нажатия на иконку навигации (см. Листинг 4.6) ничего не делается, кроме вывода информационного сообщения. Напомним, что этот обработчик события вам нужно будет использовать при выполнении домашнего задания.

Листинг 4.6. Обработчик события нажатия на иконку навигации виджета Toolbar рассматриваемого примера

```
toolbar.setNavigationOnClickListener(new View.  
    OnClickListener()  
{  
    @Override  
  
    public void onClick(View v)  
    {  
        Log.d("#####", "Navigation Icon Click");  
    }  
});
```

Далее (см. Листинг 4.5) происходит создание коллекции фильмов `ArrayList<MyMovie>` (локальная переменная `movies`). Как уже упоминалось выше, названия фильмов генерируются случайным образом. Также случайным образом генерируются и названия жанров и год выпуска фильмов.

После заполнения коллекции `movies` происходит создание объекта Адаптера Данных `android.widget.ArrayAdapter<MyMovie>` (локальная переменная `adapter`).

Поскольку при создании Адаптера Данных [android.widget.ArrayAdapter](#) необходимо указать идентификатор файла ресурсов с макетом раскладки для панели элемента списка, а макеты раскладки в рассматриваемом примере для разных видов анимаций имеют отличия в верстке (об этом будет рассказано ниже), то происходит выбор подходящего для выбранного вида анимации макета с помощью кода, который отдельно показан в Листинге 4.7.

Листинг 4.7. Выбор файла ресурсов с макетом раскладки для элементов списка android.widget.ListView в зависимости от выбранного вида анимации

```
// ----- Selecting the Layout Resource File for the
// ----- ListView Item -----
// ----- Выбор файла макета раскладки виджетов для
// ----- элемента списка -----
int movieListItemLayoutId;
switch (MainActivity.animationType)
{
    // ----- List item for "Value Animation" -----
    case MainActivity.TYPE_VALUE_ANIMATION :
        default :
            movieListItemLayoutId =
                R.layout.list_item_value_animation;
            break;

    // ----- List item for "View Animation" -----
    case MainActivity.TYPE_VIEW_ANIMATION :
        movieListItemLayoutId =
            R.layout.list_item_view_animation;
        break;
```

```
// ----- List item for "Property Animation" -----
case MainActivity.TYPE_PROPERTY_ANIMATION :
    movieListItemId =
        R.layout.list_item_value_animation;
    break;
}
```

Листинг 4.7 показывает, как в приложении осуществляется настройка в зависимости от выбранного вида анимации.

При создании Адаптера Данных `android.widget.ArrayAdapter` (локальная переменная `adapter`) происходит переопределение метода `getView()` (см. Листинг 4.5). Это делается с целью создания нестандартного набора виджетов для каждого элемента списка. Подобный способ описывался в предыдущих уроках данного курса, поэтому здесь описываться не будет. Обратите внимание, что в переопределенном методе `getView()` Адаптера Данных в Листинге 4.5 есть строка, содержащая набор символов: `***`. Запомните это место в коде — позже мы вернемся к этому месту. В этом месте (ниже в уроке) будет дописан программный код.

Далее, после назначения Адаптера Данных `adapter` списку `android.widget.ListView` (поле объекта Активности `lvMovies`), происходит назначение события нажатия на элемент списка `android.widget.ListView` (см. Листинг 4.5 в конце). Напомним, что суть рассматриваемого примера заключается в том, что при нажатии пользователем на элемент списка `lvMovies` будет происходить плавный (с применением анимации) выезд панели операций (при

этом панель элемента списка будет плавно сдвигаться влево). Так вот, в обработчике **onClick()** события нажатия на элемент списка и осуществляется запуск анимации на плавный выезд (или плавный заезд) панели операций в зависимости от типа применяемой анимации. Это делается при помощи вызова соответствующего метода: **toggleValueAnimation()** для запуска анимации Value Animation, **toggleViewAnimation()** для запуска анимации View Animation и **togglePropertyAnimation()** для запуска анимации Property Animation. В названии каждого из перечисленных методов присутствует префикс **toggle** («переключатель»), что означает, что метод вызывается как для показа панели операций, так и для ее скрытия.

Теперь можно переходить к рассмотрению применения перечисленных выше видов анимации.

4.2. Использование Value Animation

Первым шагом для рассмотрения использования в рассматриваемом примере анимации *Value Animation* для появления/скрытия панели операций будет создание файла ресурсов с макетом внешнего вида виджетов, который будет представлять каждый элемент списка (панель элемента). Исходя из необходимости делать адаптивный дизайн для элементов списка **android.widget.ListView**, создавать макет нужно таким образом, чтобы он не был строго привязан к определенному размеру по ширине. Это необходимо чтобы каждый элемент списка растягивался или сжимался в зависимости от размеров экрана устройства или от ориентации устройства, и занимал

при этом всю ширину списка `android.widget.ListView` (см. Рис. 4.5 и Рис. 4.1).

Такая адаптивная по ширине верстка достигается за счет использования атрибута `android:layout_width="match_parent"` (также такой же результат может дать применение атрибута `android:layout_width="1"`). Однако, если виджет занимает всю ширину области, предоставляемой ему родителем, то при сдвиге по оси X этого виджета он по прежнему будет занимать всю ширину области, которую дает ему родитель!

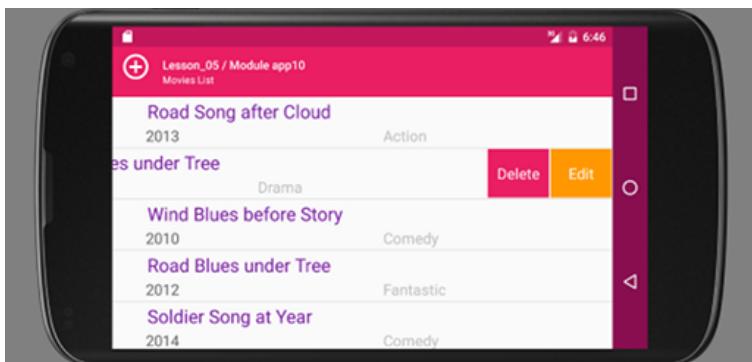


Рис. 4.5. При повороте устройства элементы списка `android.widget.ListView` также должны занимать всю ширину области виджета `android.widget.ListView`

Это означает, что сдвиг виджета по горизонтали не освободит место для другого виджета (в нашем случае — для виджета панели операций). А ведь нам необходимо, чтобы при сдвиге панели элемента, в освободившемся пространстве появилась панель операций. И так же напомним, что от решения, которое изображено на Рис. 4.4 мы принципиально отказались. При этом тут же

сообщим, что делать адаптивной высоту элемента списка `android.widget.ListView` нет необходимости — список `android.widget.ListView` имеет вертикальную полосу прокрутки и это позволяет сделать высоту элементов списка такой, какой она нам нравится (потому что все равно будет скроллинг). В рассматриваемом примере высота каждого элемента списка составляет 60dp.

Чтобы решить проблему адаптивной ширины панели элемента и необходимости появления рядом с этой панелью панели операций, был выбрано следующее решение: в качестве контейнера для элемента списка `android.widget.ListView` был выбран контейнер `android.widget.FrameLayout`. В него помещается два дочерних виджета `android.widget.LinearLayout`.

Да, да! Именно два виджета, хотя вы знаете, что виджет `android.widget.FrameLayout` предназначен для размещения одного дочернего виджета. Это так. Но есть одно уточнение — в виджет `android.widget.FrameLayout` можно поместить несколько дочерних виджетов, и эти виджеты будут расположены один над другим. Причем виджет, который помещен позже всех будет находиться выше всех и будет закрывать собой остальные виджеты!

А что если при этом самый верхний виджет сделать невидимым или сделать его нулевой ширины? Верно — он будет находиться в контейнере `android.widget.FrameLayout`, но при этом не будет закрывать собой другой виджет, который находится «под ним». И когда настанет необходимый момент, этот невидимый виджет делается видимым, и он появляется «сверху».

Если это появление сделать плавным с применением анимации, то появление ранее не видимого виджета будет эффектным, с точки зрения пользовательского интерфейса. Идея верстки макета внешнего для элемента списка `android.widget.ListView` показана на Рис. 4.6, где черная рамка — это корневой элемент макета контейнер `android.widget.FrameLayout`.

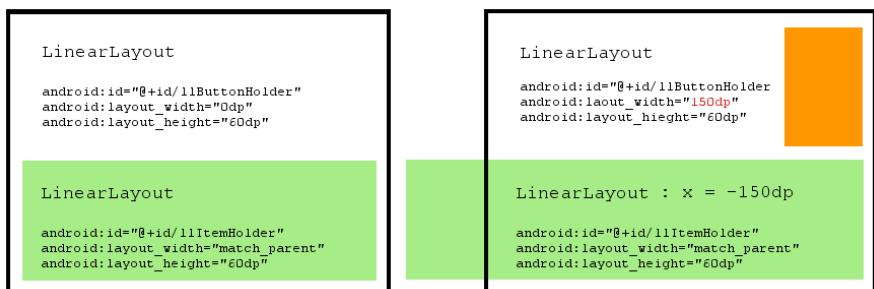


Рис. 4.6. Идея для верстки макета внешнего вида для элементов списка `android.widget.ListView` рассматриваемого примера

На Рис 4.6 слева показано состояние виджетов `android.widget.LinearLayout`, при котором панели операций нет (состояние до начала анимации), а справа (состояние после завершения анимации) показано состояние виджетов, когда панель элемента (виджет с идентификатором `R.id.l1ItemHolder`, на рисунке показан салатовым цветом) сдвинута по горизонтали влево на 150dp, а панель операций (виджет с идентификатором `R.id.l1ButtonHolder`, на рисунке показан оранжевым цветом) появилась над панелью элемента и имеет ширину 150dp.

При этом, сдвиг на фиксированную (не адаптивную) величину в 150dp не нарушает принципа адаптивности

по ширине элемента списка `android.widget.ListView` — вы видите, что панель элемента по прежнему занимает всю ширину родительской области.

Для реализации верстки макета внешнего вида элемента списка `android.widget.ListView` рассматриваемого примера (список фильмов) согласно изображению на Рис. 4.6, создан файл ресурсов `/res/layout/list_item_value_animation.xml`. Содержимое этого файла ресурсов приведено в Листинге 4.8.

Листинг 4.8. Содержимое файла ресурсов
`/res/layout/list_item_value_animation.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- First widget: Element panel -->

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:paddingLeft="40dp"
        android:id="@+id/l1ItemHolder">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="10pt"
            android:layout_gravity="left"
            android:layout_margin="2dp"
```

```
    android:id="@+id/tvTitle"
    android:textColor="#7B1FA2" />

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

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="8pt"
        android:id="@+id/tvYear"
        android:gravity="left"
        android:layout_weight="1"
        android:textColor="#616161" />

    <Space
        android:layout_width="10dp"
        android:layout_height="0dp" />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="8pt"
        android:id="@+id/tvGenre"
        android:gravity="left"
        android:layout_weight="1"
        android:textColor="#BDBDBD" />

```

</LinearLayout>

</LinearLayout>

```
<!-- Second widget: Operations panel -->
<LinearLayout
    android:orientation="horizontal"
    android:layout_gravity="right"
```

```
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:id="@+id/l1ButtonHolder">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:background="#E91E63"
        android:textColor="#FFFFFF"
        android:text="Delete"
        android:textSize="0sp"
        android:layout_margin="1dp"
        android:onClick="txtClick"
        android:id="@+id/tvDelete"
    />

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:background="#FF9800"
        android:textColor="#FFFFFF"
        android:text="Edit"
        android:textSize="0sp"
        android:layout_margin="1dp"
        android:onClick="txtClick"
        android:id="@+id/tvEdit"
    />
</LinearLayout>
</FrameLayout>
```

Обратите внимание (см. Листинг 4.8), что в качестве «кнопок» используются виджеты **android.widget.TextView** (так тоже можно) и размер шрифта для текста этих «кнопок» равен 0sp. То есть (см. Рис. 4.2), размер шрифта при

появлении и исчезновении панели операций так же будет меняться вместе с изменением размеров панели операций. И для изменения размеров шрифта так же будет использоваться анимация *Value Animation*. Также, как видно из Листинга 4.8, «кнопкам» с идентификаторами **R.id.tvDelete** и **R.id.tvEdit** назначен обработчик события нажатия **txtClick()**. Этот метод находится в классе Активности (класс **MainActivity**) и его функциональность в данном примере заключается в том, чтобы с применением анимации убирать панель операций. Код этого метода будет приведен в данном уроке ниже.

Для анимации Value Animation появления панели операций (идентификатор **R.id.llButtonHolder**), сдвига панели элемента (идентификатор **R.id.llItemHolder**) и изменения размеров шрифта «кнопок» панели операций (виджеты с идентификаторами **R.id.tvDelete**, **R.id.tvEdit**) были созданы два файла ресурсов Value Animation: */res/animator/value_animation_item_rise.xml* (анимация появления панели операций) и */res/animator/value_animation_item_fall.xml* (анимация исчезновения панели операций). Содержимое этих xml файлов ресурсов показано в Листинге 4.9.

Листинг 4.9. Содержимое файла ресурсов
/res/layout/list_item_value_animation.xml

```
/res/animator/value_animation_item_rise.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:ordering="together">
```

```
<animator
    android:valueType="intType"
    android:duration="300"
    android:valueFrom="0"
    android:valueTo="150dp" />

<animator
    android:valueType="floatType"
    android:duration="300"
    android:valueFrom="0"
    android:valueTo="18" />

<animator
    android:valueType="intType"
    android:duration="300"
    android:valueFrom="0"
    android:valueTo="-150dp" />
</set>



---

/res/animator/value_animation_item_fall.xml
-----
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:ordering="together">
    <animator
        android:valueType="intType"
        android:duration="300"
        android:valueFrom="150dp"
        android:valueTo="0" />

    <animator
        android:valueType="floatType"
        android:duration="300"
        android:valueFrom="18"
        android:valueTo="0" />
```

```
<animator
    android:valueType="intType"
    android:duration="300"
    android:valueFrom="-150dp"
    android:valueTo="0" />
</set>
```

Как видно из Листинга 4.9, каждый из файлов ресурсов, представленных в листинге, содержит набор анимаций. В каждом наборе по три анимации: первая анимация описывает изменение размера панели операций (от 0 до 150dp при появлении и от 150dp до 0 при исчезновении), вторая анимация описывает изменение размера шрифта для кнопок *Delete* и *Edit*, находящихся на панели операций (от 0sp до 18sp при появлении и от 18sp до 0sp при исчезновении) и третья анимация описывает сдвиг по горизонтали (от 0 до -150dp при появлении и от -150dp до 0 при исчезновении).

Программный код запуска анимаций из Листинга 4.9 находится в методе **toggleValueAnimation()** класса Активности, который вызывается в обработчике события нажатия на элемент списка фильмов (виджет **android.widget.ListView**). Содержимое метода **toggleValueAnimation()** приведено в Листинге 4.10.

Листинг 4.10. Метод **toggleValueAnimation()** — запуск анимации Value Animation на исполнение

```
/***
 * Run animation "Value Animation" for execution.
 * @param view      - the widget of the list item panel
 * above which it is necessary to apply the
 * animation.
```

```
* Запуск анимации "Value Animation" на
* исполнение.
* @param view - Ссылка на виджет панели
* элемента списка, над которым необходимо
* применять анимацию.
*/
private void toggleValueAnimation(View view)
{
// ----- Getting links to widgets to which
// ----- animation is applied -----
// ----- Получение ссылок на виджеты, к которым
// ----- применяется анимация -----
final LinearLayout llButtonHolder = (LinearLayout)
    view.findViewById(R.id.llButtonHolder);
final LinearLayout llItemHolder = (LinearLayout)
    view.findViewById(R.id.llItemHolder);
final TextView tvDelete = (TextView)
    view.findViewById(R.id.tvDelete);
final TextView tvEdit = (TextView)
    view.findViewById(R.id.tvEdit);

// ----- Select the required resource file for the
// ----- animation -----
// ----- Выбор необходимого файла ресурсов для
// ----- анимации -----
AnimatorSet asRise;

if (llButtonHolder.getWidth() == 0)
{
// ----- The operations panel must be shown -----
// ----- Панель операций необходимо показать -----
    asRise = (AnimatorSet) AnimatorInflater.
        loadAnimator(this,
            R.animator.value_animation_item_rise);
}
else
{
```

4. Практический пример приложения с использованием анимации

```
// ----- The operation panel must be hidden -----
// ----- Панель операций необходимо спрятать -----
    asRise = (AnimatorSet)
        AnimatorInflater.loadAnimator(this,
            R.animator.value_animation_item_fall);
}

// ----- Interpolator assignment -----
// ----- Назначение Интерполятора -----
    asRise.setInterpolator(new
        AccelerateDecelerateInterpolator());

// ----- Getting the list of animations "Value
// ----- Animation" -----
// ----- Получение списка анимаций "Value Animation"
ArrayList<Animator> arrL =
    asRise.getChildAnimations();

// ----- Handling events for changing the width of
// ----- the operation panel -----
// ----- Обработка событий изменения ширины панели
// ----- операций -----
ValueAnimator vaRise = (ValueAnimator)arrL.get(0);
vaRise.addUpdateListener(new ValueAnimator.
    AnimatorUpdateListener()
{
    @Override
    public void onAnimationUpdate(ValueAnimator
        animation)
    {
        int curWidth = (int) animation.
            getAnimatedValue();

        FrameLayout.LayoutParams LP =
            new FrameLayout.LayoutParams(
                curWidth, FrameLayout.
                    LayoutParams.MATCH_PARENT);
    }
});
```

```
    LP.gravity = Gravity.RIGHT;
    llButtonHolder.setLayoutParams(LP);
}
});

// ----- Handling events changing the font size of
// ----- the buttons "Delete", "Edit" -----
// ----- Обработка событий изменения размеров
// ----- шрифта кнопок "Delete", "Edit" -----
ValueAnimator vaRiseTextSize = (ValueAnimator)
    arrL.get(1);
vaRiseTextSize.addUpdateListener(new ValueAnimator.
    AnimatorUpdateListener()
{
    @Override
    public void onAnimationUpdate(ValueAnimator
        animation)
    {
        float curTextSize = (float) animation.
            getAnimatedValue();
        tvDelete.setTextSize(TypedValue.
            COMPLEX_UNIT_SP, curTextSize);
        tvEdit.setTextSize(TypedValue.
            COMPLEX_UNIT_SP, curTextSize);
    }
});

// ----- Handling Horizontal Shift Events for an
// ----- Element Panel -----
// ----- Обработка событий изменения сдвига по
// ----- горизонтали панели элемента -----
ValueAnimator vaRiseShift = (ValueAnimator)
    arrL.get(2);
vaRiseShift.addUpdateListener(new ValueAnimator.
    AnimatorUpdateListener()
{
    @Override
```

```

public void onAnimationUpdate(ValueAnimator animation)
{
    int curX = (int) animation.
        getAnimatedValue();
    llItemHolder.setX(curX);
}
});

// ----- Run animation "View Animation" -----
// ----- Запуск анимации "View Animation" -----
asRise.start();
}

```

Как видно из Листинга 4.10, запуск анимаций на появление панели операций и на исчезновение панели операций отличается друг от друга только выбором файла ресурса с анимацией.

Также обратите внимание (см. Листинг 4.7), что в методе **toggleValueAnimation()** происходит определение, какой именно файл ресурсов запускать при помощи сравнения ширины виджета **R.id.llButtonHolder**: если ширина равна нулю, то запускается анимация на показ панели операций (файл `/res/animator/value_animation_item_rise.xml`), в противном случае — запускается анимация на исчезновение панели операций (файл `/res/animator/value_animation_item_fall.xml`).

Внешний вид работы примера из Листингов 4.8, 4.9, 4.10 изображен на Рис. 4.1 и Рис. 4.2.

Подведем итог: использование Value Animation легко позволило реализовать поставленную задачу плавного появления панели операций.

Исходные коды примера из данного раздела находятся в модуле *app10*, среди файлов с исходными кодами, которые прилагаются к данному уроку.

4.3. Использование View Animation

Перейдем к рассмотрению применения анимации *View Animation* в нашем примере, для решения задачи плавного показа панели операций для выбранного элемента списка. Использование *View Animation* для этой задачи оказалось затруднительным. Причина заключается в том, что панель операций в нормальном состоянии элемента списка не видна. Если сделать размер для панели операций равным 0dp (то есть панель будет не видима), то анимация *View Animation* для изменения размеров виджетов предоставляет нам только механизм масштабирования (элемент `<scale>` файла ресурсов).

Казалось бы, можно воспользоваться этим механизмом для показа панели операций. Однако, учитывая тот факт, что все значения атрибутов *View Animation* (`<alpha>`, `<scale>`, `<translate>`, `<rotate>`) принимают не абсолютные, а относительные значения, получается, что если размер виджета равен нулю, то увеличение в масштабе в N раз даст в результате значение 0 (то есть $0 * N = 0$). Поэтому, для *View Animation* вариант с версткой макета элемента списка, как для *Value Animation* (см. Рис. 4.6 и Листинг 4.8) не подойдет.

Если попробовать панель элементов сделать полностью прозрачной (значение свойства виджета `alpha` равно 0) и при этом дать ей нормальный размер,

и попробовать настроить анимацию на изменение значения прозрачности (`<alpha>`) от `android:fromAlpha="0.0"` до `android:toAlpha="1.0"`, то в этом случае, опять же, из-за относительности значений, анимация *View Animation* не сделает панель операций видимой (возможно, это недоработка разработчиков компании *Google* и в более поздних версиях Android API это будет изменено).

Также есть еще один недостаток данного варианта: если виджету установить значение свойства `alpha` равное 0, то виджет все равно будет принимать события нажатия. То есть, допустим, пользователь нажимает на элемент списка с целью активизировать панель операций, но точка нажатия попадает на невидимую кнопку панели операций — в этом случае событие нажатия получит не список `android.widget.ListView`, а кнопка панели операций, следовательно, анимация на показ панели операций не запустится. Поэтому вариант с анимацией `<alpha>` отвергаем.

В результате проб нескольких вариантов решения задачи при помощи *View Animation* был выбран вариант, который показан на Рис. 4.7. На рисунке Рис. 4.7 изображено три состояния панели элемента: 1) нормальное состояние, 2) состояние перед началом показа панели операций (перед началом анимации), 3) состояние после завершения анимации — панель операций показана.

Для каждого состояния на Рис. 4.7 черная рамка — это корневой контейнер макета элемента списка `android.widget.FrameLayout`. Внутри черной рамки изображены два дочерних виджета, которые располагаются один над другим. Сверху находится панель операций (идентификатор

R.id.llButtonHolder), снизу — панель элемента (идентификатор **R.id.llItemHolder**). Для каждого из этих двух виджетов на Рис. 4.7 показаны значения свойств, которые играют важную роль в решении задачи.

Рассмотрим подробнее эти три состояния.

- 1) **Нормальное состояние.** Панель операций в нормальном состоянии панели элемента имеет свой оригинальный размер, но при этом невидима, при помощи установления свойства виджета **visibility** в значение *invisible* (**android:visibility="invisible"**). Событий нажатия кнопки панели операции в таком случае не получают. Начальное состояние панелей можно увидеть в Листинге 4.11, который показывает xml код верстки макета для элементов списка **android.widget.ListView**.
- 2) **Состояние перед началом анимации.** Это состояние возникает, когда пользователь нажал на элемент списка для показа панели операций. В этом состоянии, анимация *View Animation* начинает перемещение панели со значения атрибута **android:fromXDelta="100%"** элемента **<translate>** (см. Листинг 4.12 для /res/anim/view_animation_operations_panel.rise.xml). Это приводит к тому, что панель операций «выскакивает» за границы панели элементов и начинает движение из-за этой границы, стремясь занять свое первоначальное положение. Одновременно с этим, программно меняется значение свойства **visibility** на значение *visible*, поэтому для пользователя это выглядит, как плавное появление («выезд») панели операций, что и требуется для реализации поставленной задачи.

- 3) **Состояние после завершения анимации.** В этом состоянии панель операций находится на своем месте, только видима для пользователя и ее кнопки способны обрабатывать события нажатия. Панель элемента сдвинута вправо.

При повторном нажатии на панель элемента происходит последовательность действий, соответствующая обратному порядку, изображенному на Рис. 4.7.

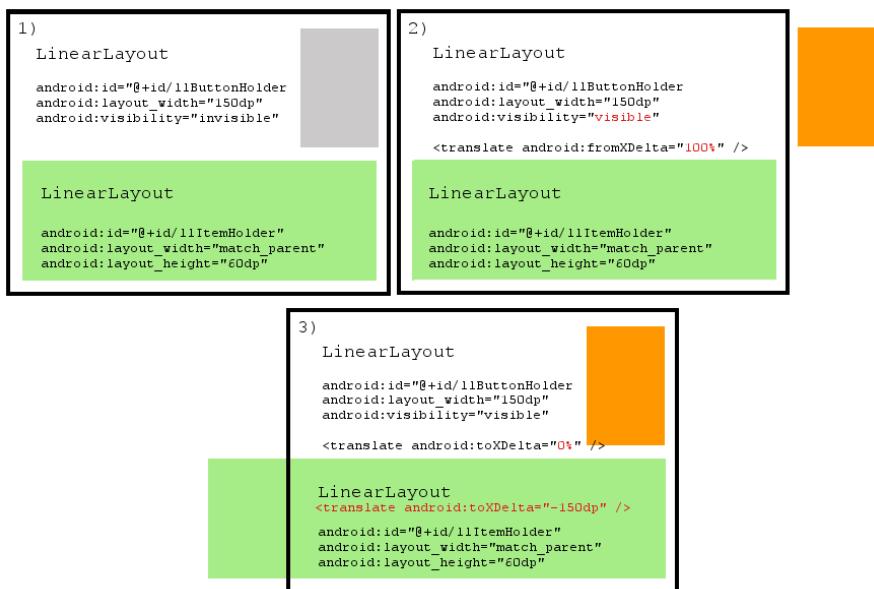


Рис. 4.7. Принцип применения View Animation для задачи показа панели операций

Теперь перейдем к рассмотрению программного кода, который реализует состояния, изображенные на Рис. 4.7.

Для макета внешнего вида каждого элемента списка фильмов, с учетом описанных выше состояний виджетов

(см. Рис. 4.7 и пояснение к нему), создан файл ресурсов /res/layout/list_item_view_animation.xml. Содержимое этого файла показано в Листинге 4.11. Жирным шрифтом в Листинге 4.11 показаны отличия от макета из Листинга 4.8.

Листинг 4.11. Содержимое файла ресурсов
/res/layout/list_item_view_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- First widget: Element panel -->

    <LinearLayout
        xmlns:android=
            "http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:id="@+id/l1ItemHolder">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="10pt"
            android:layout_gravity="center_horizontal"
            android:layout_margin="2dp"
            android:id="@+id/tvTitle"
            android:textColor="#0F5119" />

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
```

```
    android:orientation="horizontal">
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="8pt"
            android:id="@+id/tvGenre"
            android:gravity="center_horizontal"
            android:layout_weight="1"
            android:textColor="#0F5119" />
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:textSize="8pt"
            android:id="@+id/tvYear"
            android:gravity="center_horizontal"
            android:layout_weight="1"
            android:textColor="#0F5119" />
    </LinearLayout>

</LinearLayout>

<!-- Second widget: Operations panel -->

<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_gravity="right"
    android:layout_width="150dp"
    android:visibility="invisible"
    android:layout_height="60dp"
    android:id="@+id/l1ButtonHolder">

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
```

```
        android:gravity="center"
        android:background="#E91E63"
        android:textColor="#FFFFFF"
        android:text="Delete"
        android:textSize="18sp"
        android:onClick="txtClick"
        android:id="@+id/tvDelete" />

    <Space
        android:layout_width="2dp"
        android:layout_height="2dp"/>

    <TextView
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:background="#FF9800"
        android:textColor="#FFFFFF"
        android:text="Edit"
        android:textSize="18sp"
        android:onClick="txtClick"
        android:id="@+id/tvEdit" />

```

</LinearLayout>

```
</FrameLayout>
```

Для *View Animation* в модуле *app10* создан каталог ресурсов */res/anim*. В котором созданы четыре файла ресурсов *View Animation* (см. Листинг 4.12):

- */res/anim/view_animation_item_panel_rise.xml* — описывает анимацию *View Animation* для сдвига влево панели элемента (**R.id.llItemHolder**), при появлении панели операций;
- */res/anim/view_animation_item_panel.fall.xml* — описывает анимацию *View Animation* для сдвига вправо

панели элемента (**R.id.llItemHolder**), при исчезновении панели операций;

- /res/anim/ view_animation_operations_panel.rise.xml — описывает анимацию *View Animation* для появления панели операций (**R.id.llButtonHolder**). Этой анимации для появления панели недостаточно — необходимо еще программно выполнить подготовительные действия, которые показаны в Листинге 4.13;
- /res/anim/ view_animation_operations_panel.fall.xml — описывает анимацию *View Animation* для исчезновения панели операций (**R.id.llButtonHolder**). После завершения анимации необходимо выполнить дополнительные действия, чтобы вернуть панель элемента в состояние 1 (см. Рис. 4.7 и пояснение к нему). Эти дополнительные действия можно увидеть в Листинге 4.12 для /res/anim/ view_animation_operations_panel.fall.xml и в Листинге 4.13.

Перечисленные выше файлы ресурсов приведены в Листинге 4.12.

Листинг 4.12. Содержимое файлов ресурсов View Animation:

```
/res/anim/view_animation_item_panel_rise.xml,  
/res/anim/view_animation_item_panel_fall.xml,  
/res/anim/view_animation_operations_panel_rise.xml,  
/res/anim/view_animation_operations_panel_fall.xml
```

```
/res/anim/view_animation_item_panel_rise.xml
```

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

```
<translate  
    android:fromXDelta="0"  
    android:fromYDelta="0"  
    android:toYDelta="0"  
    android:toXDelta="-300"  
    android:duration="300" />  
</set>
```

/res/anim/view_animation_item_panel_fall.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android=  
      "http://schemas.android.com/apk/res/android">  
    <translate  
        android:fromXDelta="-300"  
        android:fromYDelta="0"  
        android:toYDelta="0"  
        android:toXDelta="0"  
        android:duration="300" />  
</set>
```

/res/anim/view_animation_operations_panel_rise.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<translate  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    android:duration="300"  
    android:fromXDelta="100%"  
    android:toXDelta="0%"  
    android:fillAfter="true"  
    android:fillEnabled="true"  
    android:fillBefore="true"  
/>
```

```
/res/anim/view_animation_operations_panel_fall.xml  
-----  
<?xml version="1.0" encoding="utf-8"?>  
<set xmlns:android=  
      "http://schemas.android.com/apk/res/android">  
    <translate  
        xmlns:android=  
            "http://schemas.android.com/apk/res/android"  
        android:duration="300"  
        android:fromXDelta="0%"  
        android:toXDelta="100%"  
        android:fillAfter="false"  
        android:fillEnabled="true"  
        android:fillBefore="true"  
    />  
  
    <alpha  
        android:duration="10"  
        android:startOffset="290"  
        android:fromAlpha="1.0"  
        android:toAlpha="0.0"  
        android:fillAfter="true" />  
</set>
```

Код, показанный в Листинге 4.12, предварительно комментировался при описании состояний, изображенных на Рис. 4.7. Здесь только будет добавлено, что в файле `/res/anim/view_animation_operations_panel_fall.xml` добавлена анимация на прозрачность панели операций (элемент `<alpha>`). Длительность анимации составляет 10 миллисекунд и начинается эта анимация практически в самом конце анимации перемещения (значение атрибута `android:startOffset="290"`). Пользователь не увидит

этой анимации, но по ее завершению, панель операций `R.id.llButtonHolder` станет невидимой, и это даст ей возможность вернуться назад в состояние 1 (см. Рис. 4.7) не заметно для пользователя.

Запуск анимаций из Листинга 4.12 осуществляется в методе `toggleViewAnimation()` класса Активности (класс `MainActivity`). Содержимое этого метода приведено в Листинге 4.13. Сам метод `toggleViewAnimation()` запускается в обработчике события нажатия на элемент списка `android.widget.ListView` (метод `onItemClick()`), код которого приведен в Листинге 4.5.

Листинг 4.13. Метод `toggleViewAnimation()` — запуск анимации View Animation на исполнение

```
/**
 * Run animation "View Animation" for execution.
 * @param view - the widget of the list item panel
 * above which it is necessary to apply the
 * animation.
 *
 * Запуск анимации "View Animation" на исполнение.
 * @param view - Ссылка на виджет панели элемента
 * списка, над которым необходимо применять
 * анимацию.
 */
private void toggleViewAnimation(View view)
{
    // ----- Start animation for shift of the
    // ----- element panel -----
    // ----- Запуск анимации сдвига панели
    // ----- элемента -----
    final LinearLayout llItemHolder = (LinearLayout)
        view.findViewById(R.id.llItemHolder);
```

4. Практический пример приложения с использованием анимации

```
Animation animItem = AnimationUtils.  
    loadAnimation(this,  
        (llItemHolder.getTag() == null) ?  
            R.anim.view_animation_item_panel_rise :  
            R.anim.view_animation_item_panel_fall);  
animItem.setFillAfter(true);  
llItemHolder.startAnimation(animItem);  
  
// ----- Start animation for operations panel  
// ----- Запуск анимации показа панели  
// ----- операций -----  
final LinearLayout llButtonHolder = LinearLayout)  
    view.findViewById(R.id.llButtonHolder);  
Animation animOperations = AnimationUtils.  
    loadAnimation(this,  
        (llItemHolder.getTag() == null) ?  
            R.anim.view_animation_operations_panel_rise :  
            R.anim.view_animation_operations_panel_fall);  
llButtonHolder.startAnimation(animOperations);  
  
// ----- Do preparatory or final actions  
// ----- Делаем подготовительные или  
// ----- завершающие действия -----  
if (llItemHolder.getTag() == null)  
{  
    llButtonHolder.setVisibility(View.VISIBLE);  
    llItemHolder.setTag(1);  
}  
else  
{  
    llItemHolder.setTag(null);  
    llButtonHolder.setVisibility(View.INVISIBLE);  
}  
}
```

Как видно из Листинга 4.13, метод `toggleViewAnimation()` используется как для запуска анимации на появления панели операций, так и для запуска анимации на исчезновение панели операций. В качестве критерия — какую анимацию запускать — используется назначение панели элемента тега со значением равным «1» или «null» (`llItemHolder.setTag(1)` или `llItemHolder.setTag(null)`). Если значение тега равно «null», то нужно запускать анимацию на показ панели операций, в противном случае — нужно запускать анимацию на исчезновение.

Human Alarm under Story Melodrama 2008	Human Alarm under Story Melodrama 2008
Summer River and Tree Comedy 2003	Summer River and Tree Comedy 2003
House Sea after Story Action 2013	House Sea after Story Action 2013
Human Alarm under Story Melodrama 2008	Human Alarm under Story Melodrama 2008
Summer River and Tree Comedy 2003	Summer River and Tree Comedy 2003

Рис. 4.8. Последовательность скриншотов, показывающая анимацию View Animation из Листингов 4.11, 4.12, 4.13

Внешний вид получившейся анимации *View Animation* отличается от внешнего вида, изображенного на Рис. 4.2. Так как использование *View Animation* не позволило реализовать поведение виджетов, показанное на Рис. 4.2, из-за ограничений, описанных выше. Последовательность скриншотов, показывающая появление панели

операций при использовании *Value Animation* изображена на Рис. 4.8. Найдите отличия. :)

Для того, чтобы исполнить в модуле *app10* анимацию *View Animation*, необходимо в статическое поле **animationType** класса **MainActivity** установить значение **MainActivity.TYPE_VIEW_ANIMATION**, как показано в Листинге 4.14.

Листинг 4.14. Конфигурирование приложения модуля *app10* для запуска анимации *View Animation*

```
class MainActivity
{
    ..
    private final static int animationType =
        MainActivity.TYPE_VIEW_ANIMATION;
    ..
}
```

Подведем итог. Использование анимации View Animation для решения задачи плавного появления панели операций, хотя и позволило добиться желаемого, оказалось достаточно неудобным. Можно рекомендовать использование View Animation для простых, не сложных, анимаций над виджетами. Применение View Animation для комплексных анимаций не дает простых решений, что, в свою очередь, требует от разработчика большие времени для решения задач.

Исходные коды примера из данного раздела можно найти в модуле *app10*, среди файлов с исходными кодами которые прилагаются к данному уроку.

4.4. Использование Property Animation

Рассмотрим применение анимации *Property Animation* для решения задачи плавного появления панели операций, для выбранного элемента списка фильмов в виджете `android.widget.ListView`. Сразу отметим, что применение анимации *Property Animation* легко позволяет решить задачу появления панели операций по варианту, описанному в Рис. 4.6 и показанному на Рис. 4.1 и 4.2. Неизвестным вопросом на данный момент является вопрос, как применить анимацию *Property Animation* для изменения ширины панели операций, ведь *Property Animation* не позволяет явно модифицировать значение этого свойства. Но решение данного вопроса имеется и будет описано чуть ниже.

Для макета внешнего вида элементов списка, с учетом пути решения, изображенного на Рис. 4.6, для *Property Animation* берется тот же макет, который использовался для рассмотрения *Value Animation*, то есть файл `/res/layout/list_item_value_animation.xml`, содержимое которого приведено в Листинге 4.8.

Для изменения размеров шрифта надписей виджетов `R.id.tvDelete` («кнопка» с надписью *Delete*) и `R.id.tvEdit` («кнопка» с надписью *Edit*) созданы файлы ресурсов *Property Animation* `/res/animator/property_animation_text_size_rise.xml` (увеличение размера шрифта от 0sp до 10sp) и `/res/animator/property_animation_text_size_fall.xml` (уменьшение размера шрифта от 10sp до 0sp). Содержимое этих файлов ресурсов приведено в Листинге 4.15.

Листинг 4.15. Содержимое файлов ресурсов
/res/animator/property_animation_text_size_rise.xml
и /res/animator/property_animation_text_size_fall.xml

res/animator/property_animation_text_size_rise.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="textSize"
    android:duration="300"
    android:valueFrom="0sp"
    android:valueTo="10sp"
/>
```

res/animator/property_animation_text_size_fall.xml

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="textSize"
    android:duration="300"
    android:valueFrom="10sp"
    android:valueTo="0sp"
/>
```

Для плавного сдвига панели элемента созданы файлы ресурсов *Property Animation*: /res/animator/property_animation_list_item_rise.xml (сдвигает панель элемента влево при появлении панели операций) и /res/animator/property_animation_list_item_fall.xml (возвращает панель элемента при исчезновении панели операций). Содержимое этих файлов показано в Листинге 4.16.

Листинг 4.16. Содержимое файлов ресурсов Property Animation

/res/Animator/property_animation_list_item_rise.xml
и /res/Animator/property_animation_list_item_fall.xml

```
/res/Animator/property_animation_list_item_rise.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="x"
    android:duration="300"
    android:valueFrom="0"
    android:valueTo="-150dp"
    android:valueType="floatType" />
```

```
/res/Animator/property_animation_list_item_fall.xml
```

```
<?xml version="1.0" encoding="utf-8"?>
<objectAnimator
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:propertyName="x"
    android:duration="300"
    android:valueFrom="-150dp"
    android:valueTo="0"
    android:valueType="floatType" />
```

Содержимое Листингов 4.15 и 4.16 достаточно очевидно и поэтому подробно не объясняется. А вот для изменения размеров (а точнее ширины) панели операций познакомимся со специальными возможностями класса [android.animation.ObjectAnimator](#). В классе [android.animation.ObjectAnimator](#) существуют статические методы,

которые обобщенно назовем **ofType()** (**ofInt()**, **ofFloat()**, **ofArgb()** и так далее). Эти методы предназначены для создания объектов **android.animation.ObjectAnimator**, которые будут модифицировать свойства других объектов, причем эти другие объекты вовсе не обязательно должны быть виджетами. Познакомимся с возможностями этих методов класса **android.animation.ObjectAnimator** на примере всего одного метода:

```
static ObjectAnimator ofFloat(Object target, String  
                               propertyName, float... values);
```

Этот метод принимает ссылку на объект (параметр **target**), к которому будет применяться анимация. Параметр **propertyName** содержит название свойства, которое необходимо изменять для применения анимации. При этом (внимание!!!) в объекте **target** должен быть метод **setPropertyName()**. То есть, например, если в параметре **propertyName** передать значение **Volume**, то в объекте, на который ссылается параметр **target** должен быть метод **setVolume()**, который и будет вызываться объектом **android.animation.ObjectAnimator** для изменения значения свойства **Volume**. При этом, поскольку метод называется **ofFloat()**, тип свойства **Volume** должен быть **float**.

Параметры **values** (их может быть сколько угодно) содержат значения, через которые должно пройти свойство объекта **target**, указанное в параметре **propertyName**.

Метод **ofFloat()** возвращает созданный объект **android.animation.ObjectAnimator**, который будет применять анимацию к свойству **propertyName** объекта **target**.

Для изменения ширины панели операций и будет использован рассмотренный метод `ofFloat()`. Для объекта, ссылка на который будет передаваться в метод `ofFloat()`, создается специальный класс, который в нашем примере получил название **WidthEvaluator**. Содержимое этого класса приведено в Листинге 4.17.

Листинг 4.17. Класс `WidthEvaluator`, при помощи которого в рассматриваемом примере будет изменяться ширина панели операций, при помощи `Property Animation`

```
/**  
 * Class WidthEvaluator. Auxiliary class for  
 * Property Animation.  
 * It helps to apply animation to the width property  
 * of widgets.  
 *  
 * Класс WidthEvaluator. Вспомогательный класс для  
 * "Property Animation", помогающий применить  
 * анимацию к свойству width виджетов.  
 */  
class WidthEvaluator  
{  
    /**  
     * A reference to a widget whose width will be  
     * changed with the help of  
     * android.animation.ObjectAnimator object.  
     *  
     * Ссылка на виджет, ширина которого будет  
     * изменяться при помощи объекта  
     * android.animation.ObjectAnimator  
     */  
    private View v;  
    // ----- Class methods -----  
    public WidthEvaluator(View v)  
    {
```

```
        this.v = v;
    }

    /**
     * The method that will be called by the
     * android.animation.ObjectAnimator
     * to change the width of the widget v.
     *
     * @param w - The current value for the width of
     * the widget that is set by the
     * android.animation.ObjectAnimator
     *
     * Метод, который будет вызываться объектом
     * android.animation.ObjectAnimator
     * для изменения ширины виджета v.
     *
     * @param w - Текущее значение для ширины
     * виджета, которое устанавливается объектом
     * android.animation.ObjectAnimator
     */
    public void    setWidth(float w)
    {
        ViewGroup.LayoutParams params =
            v.getLayoutParams();
        params.width      = (int) w;
        v.setLayoutParams(params);
    }
}
```

Как видно из Листинга 4.17, в классе **WidthEvaluator** есть поле **v**, которое может ссылаться на любой виджет. Это поле инициализируется при помощи конструктора. Также в классе **WidthEvaluator** объявлен метод **setWidth()**, который предназначен для вызова объектом **android.animation.ObjectAnimator**. Метод **setWidth()** использует

получаемое через параметр **w** значение ширины для виджета, на который ссылается поле **v**. Все очень просто.

В Листинге 4.18 показано, как запустить анимацию *Property Animation* с использованием объекта **WidthEvaluator** для изменения ширины виджета панели операций.

Листинг 4.18. Запуск анимации *Property Animation* с использованием объекта класса **WidthEvaluator** для изменения ширины виджета панели операций

```
LinearLayout llButtonHolder =  
    (LinearLayout) this.findViewById  
    (R.id.llButtonHolder);  
  
ObjectAnimator.ofFloat  
    (new WidthEvaluator(llButtonHolder),  
    "Width", 0f, 300f).start();
```

Как видно из Листинга 4.18, ширина виджета **R.id.llButtonHolder** будет меняться от начального значения 0 до конечного значения 300.

Теперь осталось рассмотреть запуск анимаций из Листингов 4.16, 4.17 и 4.18 для решения задачи плавного появления панели операций. Запуск вышеуказанных анимаций *Property Animation* осуществляется в методе **togglePropertyAnimation()** класса **MainAcitivity**. Содержимое этого метода приведено в Листинге 4.19. Сам метод **togglePropertyAnimation()** запускается в обработчике события нажатия на элемент списка **android.widget.ListView** (метод **onItemClick()**), код которого приведен в Листинге 4.5.

Листинг 4.19. Содержимое метода togglePropertyAnimation() для запуска Property Animation

```
/**  
 * Run animation "Property Animation" for execution.  
 * @param view - the widget of the list item panel  
 * above which it is necessary to apply the animation  
 *  
 * Запуск анимации "Property Animation" на  
 * исполнение.  
 * @param view - Ссылка на виджет панели элемента  
 * списка, над которым необходимо применять анимацию  
 */  
  
private void togglePropertyAnimation(View view)  
{  
    // ----- Start animation for shift of the element  
    // ----- panel -----  
    // ----- Запуск анимации сдвига панели элемента  
  
    final LinearLayout llItemHolder =  
        (LinearLayout)  
            view.findViewById(R.id.llItemHolder);  
  
    ObjectAnimator objectAnimator = (ObjectAnimator)  
        AnimatorInflater.loadAnimator(  
            this,  
            (llItemHolder.getX() == 0) ?  
                R.animator.  
                    property_animation_list_item_rise :  
                R.animator.  
                    property_animation_list_item_fall);  
    objectAnimator.setTarget(llItemHolder);  
    objectAnimator.start();  
    final TextView tvDelete =  
        (TextView) view.findViewById(R.id.tvDelete);  
    final TextView tvEdit = (TextView)  
        view.findViewById(R.id.tvEdit);
```

```
// ----- Run the font size change animation for
// ----- the widget R.id.tvDelete -----
// ----- Запуск анимации изменения размера шрифта
// ----- для виджета R.id.tvDelete
ObjectAnimator objectAnimator2 =
    (ObjectAnimator)
    AnimatorInflater.loadAnimator(
        this,
        (l1ItemHolder.getX() == 0) ?
            R.animator.
                property_animation_text_size_rise :
            R.animator.
                property_animation_text_size_fall);
objectAnimator2.setTarget(tvDelete);
objectAnimator2.start();

// ----- Run the font size change animation for
// ----- the widget R.id.tvEdit -----
// ----- Запуск анимации изменения размера шрифта
// ----- для виджета R.id.tvEdit -----
ObjectAnimator objectAnimator3 =
    (ObjectAnimator)
    AnimatorInflater.loadAnimator(
        this,
        (l1ItemHolder.getX() == 0) ?
            R.animator.
                property_animation_text_size_rise :
            R.animator.
                property_animation_text_size_fall);

objectAnimator3.setTarget(tvEdit);
objectAnimator3.start();

// ----- Run the animation to change the width of
// ----- the operation panel -----
// ----- Запуск анимации изменения ширины панели
// ----- операций -----
```

```
final LinearLayout llButtonHolder = (LinearLayout)
    view.findViewById(R.id.llButtonHolder);

if (llItemHolder.getX() == 0)
{
    ObjectAnimator.ofFloat(
        new WidthEvaluator(llButtonHolder),
        "Width", 0f, 300f).start();
}
else
{
    ObjectAnimator.ofFloat(
        new WidthEvaluator(llButtonHolder),
        "Width", 300f, 0f).start();
}
}
```

Напомним, что для запуска приложения из модуля *app10* для демонстрации применения анимации *Property Animation*, необходимо установить значение статического поля **animationType** класса **MainActivity** в значение **MainActivity.TYPE_PROPERTY_ANIMATION**, как показано в Листинге 4.20.

Листинг 4.20. Конфигурирование приложения модуля *app10* для запуска анимации *Property Animation*

```
class MainActivity
{
    ...
    private final static int animationType =
        MainActivity.TYPE_PROPERTY_ANIMATION;
    ...
}
```

Последовательность скриншотов, демонстрирующая применение *Property Animation* из Листингов 4.16–4.20, изображена на Рис. 4.2.

Подведем итог. Применение анимации Property Animation для решения задачи плавного показа панели операций для элементов списка фильмов, позволило полноценно решить поставленную задачу. Решение является несложным. Анимация Property Animation является удобным и гибким инструментом при создании анимационных эффектов в приложениях как с графическим пользовательским интерфейсом, так и любых других (например игровых) приложений.

Исходный код применения анимации *Property Animation* находится в модуле *app10* среди файлов с исходными кодами, которые прилагаются к данному уроку.

4.5. Завершающее описание общих частей рассматриваемого примера

Нам осталось рассмотреть еще две детали (части) в рассматриваемом примере на использование анимации при появлении панели операций. Во-первых, это обработчик события клика по кнопкам панели операций *txtClick()*. И, во-вторых, пропущенный в Листинге 4.5 код, обозначенный тремя звездочками (метод *getView()* Адаптера Данных *android.widget.ArrayAdapter<T>*).

В обработчике клика на кнопки *R.id.tvDelete* и *R.id.tvEdit* панели операций (метод *txtClick()*) происходит запуск анимации на исчезновение панели операций

(ведь пользователь сделал свой выбор и панель операций должна исчезнуть). Исчезновение панели операций осуществляется при помощи того вида анимации, под которое сконфигурировано приложение при помощи статического поля **animationType** класса **MainActivity** (см. Листинги 4.14 и 4.20). Содержимое метода **txtClick()** приведено в Листинге 4.21.

Листинг 4.21. Метод **txtClick()** — обработчик события нажатия на виджеты **R.id.tvDelete** и **R.id.tvEdit**

```
/**  
 * Event handler for clicking on the R.id.tvDelete  
 * and R.id.tvEdit buttons.  
 * @param v - Reference to the button on which  
 * clicked.  
 *  
 * Метод является обработчиком события нажатия на  
 * кнопки R.id.tvDelete и R.id.tvEdit панели операций.  
 * @param v - ссылка на кнопку, по которой  
 * осуществлено нажатие.  
 */  
  
public void txtClick (View v)  
{  
// ----- Get a link to the root container of the  
// ----- list item -----  
// ----- Получаем ссылку на корневой контейнер  
// ----- элемента списка -----  
View view = (View) v.getParent().getParent();  
switch (MainActivity.animationType)  
{  
// ----- "Value Animation" -----  
case MainActivity.TYPE_VALUE_ANIMATION:  
default:  
    MainActivity.this.toggleValueAnimation(view);  
    break;  
}
```

```

// ----- "View Animation" -----
    case MainActivity.TYPE_VIEW_ANIMATION:
        MainActivity.this.toggleViewAnimation(view);
        break;

// ----- "Property Animation" -----
    case MainActivity.TYPE_PROPERTY_ANIMATION:
        MainActivity.this.togglePropertyAnimation(view);
        break;
}

// ----- Here it is necessary to place the code
// ----- for the operations "Delete" and "Edit"
// ----- Здесь необходимо разместить код для
// ----- операций "Delete" и "Edit"
Log.d("#####", "txtClick");
}

```

Как видно из Листинга 4.21, исчезновение панели операций при нажатии на виджеты `R.id.tvDelete` и `R.id.tvEdit` осуществляется при помощи все тех же методов `toggleТипАнимацииAnimation()`, которые вызываются в методе обработчике события нажатия на элемент списка `onItemClick()` (см. Листинг 4.5).

Напомним, что методы `toggleValueAnimation()`, `toggleViewAnimation()`, `togglePropertyAnimation()` вызываются в зависимости от конфигурации приложения (статическое поле `animationType` класса `MainActivity`).

Теперь вернемся к коду (см. Листинг 4.5), который помечен символами "***" (метод `getView()` Адаптера Данных `android.widget.ArrayAdapter<T>`). В этом коде осуществляется скрытие панели операций без применения анимации.

Это необходимо сделать для случая, когда панель операций показана, но пользователь проигнорировал ее и осуществляет скроллинг списка `android.widget.ListView` (поле `lvMovies`). В этом случае панель элемента и панель операций необходимо вернуть в первоначальное состояние. Это делается без применения анимации, так как этот элемент списка `android.widget.ListView` (поле `lvMovies`) не виден, так как находится за пределами экрана. Содержимое этого кода (то, что должно быть вместо "****", в Листинге 4.5) приведено в Листинге 4.22.

Листинг 4.22. Фрагмент кода метода `getView()` из Листинга 4.5, в котором панель операций и панель элемента приводятся в начальное состояние, так как элемент списка исчез с экрана по причине прокрутки списка

```
// ----- Hide operations panel if necessary -----
// ----- Спрятать панель операций, если нужно -----
final LinearLayout llButtonHolder =
    (LinearLayout)view.findViewById(R.id.llButtonHolder);
final LinearLayout llItemHolder =
    (LinearLayout) view.findViewById(R.id.llItemHolder);
final TextView tvDelete =
    (TextView) view.findViewById(R.id.tvDelete);
final TextView tvEdit =
    (TextView) view.findViewById(R.id.tvEdit);

switch (MainActivity.animationType)
{
// ----- "Value Animation" and "Property Animation"
    case MainActivity.TYPE_VALUE_ANIMATION:
    case MainActivity.TYPE_PROPERTY_ANIMATION :
    default:
{
```

```
        if (llButtonHolder.getWidth() != 0)
        {
// ----- The width of the operation panel becomes 0
// ----- Ширина панели операций становится = 0 ---
        FrameLayout.LayoutParams LP =
            new FrameLayout.LayoutParams(0,
                FrameLayout.LayoutParams.
                MATCH_PARENT);
        LP.gravity = Gravity.RIGHT;
        llButtonHolder.setLayoutParams(LP);

// ----- The font size = 0 for the widgets R.id.
// ----- tvDelete and R.id.tvEdit -----
// ----- Размер шрифта = 0 для виджетов R.id.
// ----- tvDelete и R.id.tvEdit -----
        tvDelete.setTextSize(TypedValue.
            COMPLEX_UNIT_SP, 0);
        tvEdit.setTextSize(TypedValue.
            COMPLEX_UNIT_SP, 0);

// ----- The horizontal position for the element
// ----- panel = 0 -----
// ----- The horizontal position for the element
// ----- panel = 0 -----
        llItemHolder.setX(0);
    }
}
break;

// ----- "View Animation" -----
case MainActivity.TYPE_VIEW_ANIMATION:
    if (llItemHolder.getTag() != null)
    {

// ----- Run an instant animation to shift the
// ----- element panel backward -----
// ----- Запуск мгновенной анимации на сдвиг
// ----- панели элемента назад -----
    }
```

```
Animation animItem =
    AnimationUtils.loadAnimation(
        MainActivity.this,
        R.anim.
            view_animation_item_panel_fall);
animItem.setFillAfter(true);
animItem.setDuration(0);
llItemHolder.startAnimation(animItem);

// ----- Running instant animation to disappear
// ----- the operation panel -----
// ----- Запуск мгновенной анимации на
// ----- исчезновение панели операций -----
Animation animOperations =
    AnimationUtils.loadAnimation(
        MainActivity.this,
        R.anim.
            view_animation_operations_panel_fall);
animOperations.setDuration(0);
llButtonHolder.startAnimation(animOperations);

// ----- The operations panel becomes invisible
// ----- Панель операций делается невидимой -----
llButtonHolder.setVisibility(View.
    INVISIBLE);

// ----- Set the value of the tag field to null ---
// ----- Устанавливаем значение поля tag в null --
    llItemHolder.setTag(null);
}
break;
}
```

Из Листинга 4.22 видно для анимаций *Value Animation* и *Property Animation* понадобился одинаковый код, для приведения панелей операций и элемента в первоначальное

состояние. Для *View Animation* опять же пришлось прибегать к крайностям — без запуска анимации привести виджеты панели элемента и панели операций не получалось. В этом случае время исполнения анимации *View Animation*, при помощи метода `setDuration()`, устанавливалось равным 0 миллисекунд (т. е. мгновенно).

Исходные коды примеров текущего раздела можно найти в модуле *app10* среди файлов с исходными кодами, которые прилагаются к данному уроку.

5. Домашнее задание

1. Создайте в модуле проекта по два текстовых файла в каталогах `/res/raw` и `/assets`. Напишите приложение, которое прочитает содержимое этих файлов и выведет это содержимое в виджеты `android.widget.TextView`.
2. Задание на использование *Frame Animation*. Найдите где-нибудь в сети Интернет спрайт с движущимся персонажем (человечком, животным, птицей и т. д.). Например, как показано на Рис. 5.1.



Рис. 5.1. Пример спрайта для домашнего задания № 2

Напишите макет игрового приложения, в котором персонаж двигался бы, в ответ на команды пользователя, как показано на Рис. 5.2. Командами пользователя являются нажатия на области экрана, которые показаны на Рис. 5.3 голубым цветом.

Обращаем ваше внимание, что игровой персонаж всегда находится в центра экрана (см. Рис. 5.2). При нажатии на правую голубую область экрана (см. Рис. 5.3), игровой персонаж двигается вправо до тех пор, пока пользователь не прекратит удерживать палец на экране или пользователь не нажмет на левую область экрана.

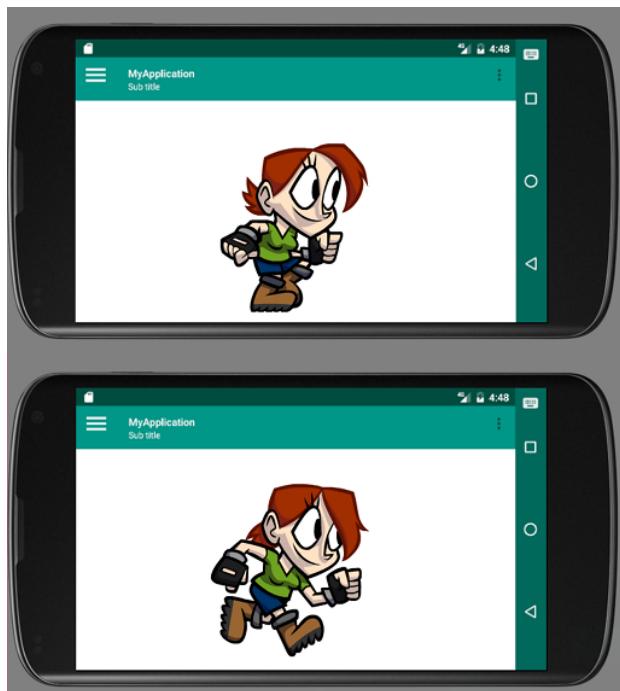


Рис. 5.2. Движение игрового персонажа в ответ на события нажатия пользователя

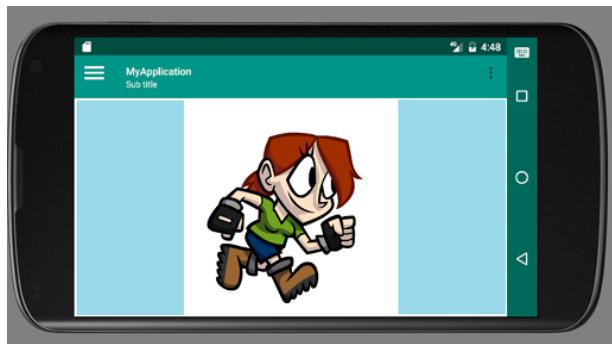


Рис. 5.3. Голубым цветом обозначены области экрана (правая и левая), по событиям нажатия на которые игровой персонаж осуществляет движение

При нажатии на левую голубую область экрана (см. Рис. 5.3), игровой персонаждвигается влево до тех пор, пока пользователь не прекратит удерживать палец на экране или пользователь не нажмет на правую область экрана. При прекращении нажатий со стороны пользователя, игровой персонаж останавливается.

Приложение должно быть зафиксировано в альбомной ориентации (программно). Как это сделать, рассказывалось в одном из наших предыдущих уроков этого курса.

Подобрать изображения для пиктограммы приложения с различной детализацией и разместить их в каталоге /res/mipmap. Назначить в Манифесте приложения пиктограмму для этого приложения.

3. Необходимо доработать пример со списком фильмов из раздела 4 данного урока. В Листинге 4.3 показана верстка макета Активности для примера применения анимации к элементам списка фильмов. В макете Активности размещен виджет `android.widget.FrameLayout` с идентификатором `R.id.flAddPanel` (см. Листинг 5.1).

Листинг 5.1. Виджет `android.widget.FrameLayout` из макета Активности модуля app10

```
<FrameLayout  
    android:layout_width="match_parent"  
    android:layout_height="0dp"  
    android:background="#F48FB1"  
    android:id="@+id/flAddPanel">  
</FrameLayout>
```

Этот виджет в примере модуля *app10* не используется, однако он предназначен для текущего домашнего задания. Суть задания заключается в следующем. Виджет **R.id.flAddPanel** (см. Листинг 5.1 и 4.3) предназначен быть панелью для добавления/редактирования фильма. Чтобы добавить фильм, пользователь приложения должен нажать на иконку навигации, расположенную на тулбаре (см. Рис. 5.4 в правом верхнем углу). Разумеется, внешний вид кнопки навигации должен быть как на Рис. 4.1 (символ плюс в кружочке).

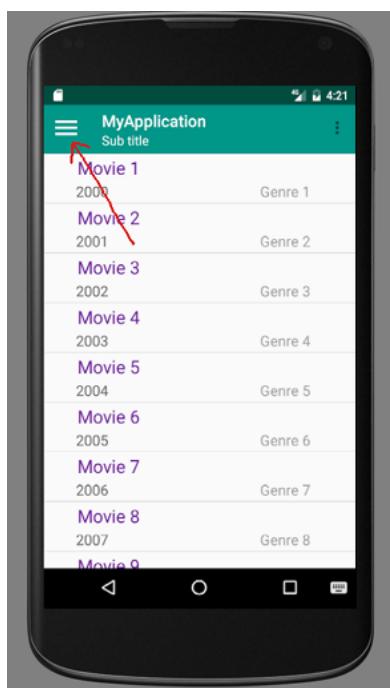


Рис. 5.4. Кнопка навигации, при нажатии на которую должна появиться панель добавления фильма

Панель добавления/редактирования должна появиться с использованием анимации, как показано на Рис. 5.5. Время плавного появления панели добавления/редактирования должно составлять 300 миллисекунд. Высота панели добавления/редактирования должна стать равной где-то 390dp. Далее, после появления панели добавления/редактирования, на ней должны плавно появится виджеты, как показано на Рис. 5.6. Время плавного появления виджетов (см. Рис. 5.6) составляет 300 миллисекунд. Используйте *Value Animation*. Реализуйте появление панели добавления/редактирования и последующее появление виджетов на ней при помощи одного набора (`<set>`) анимаций *Value Animation*. При повторном нажатии на кнопку навигации (см. Рис. 5.4) или при нажатии на кнопки *Apply* или *Cancel* (см. Рис. 5.6) панель добавления/редактирования должна плавно исчезнуть в обратном порядке, чем порядок, изображенный на Рис. 5.5 и 5.6. То есть, вначале исчезают виджеты на панели добавления/редактирования, а затем исчезает и сама панель. Время исчезновения виджетов также составляет 300 миллисекунд, время исчезновения панели добавления/редактирования также составляет 300 миллисекунд. Анимацию исчезновения панели добавления/редактирования также необходимо реализовать одним набором (`<set>`).

Разумеется, не нужно повторять внешний вид панели добавления/редактирования, который изображен на Рис. 5.5 и 5.6. Воспользуйтесь знаниями из данного урока, чтобы придать панели добавления/редактирования и виджетам, которые на ней размещены, более стильный внешний вид.

Урок №5

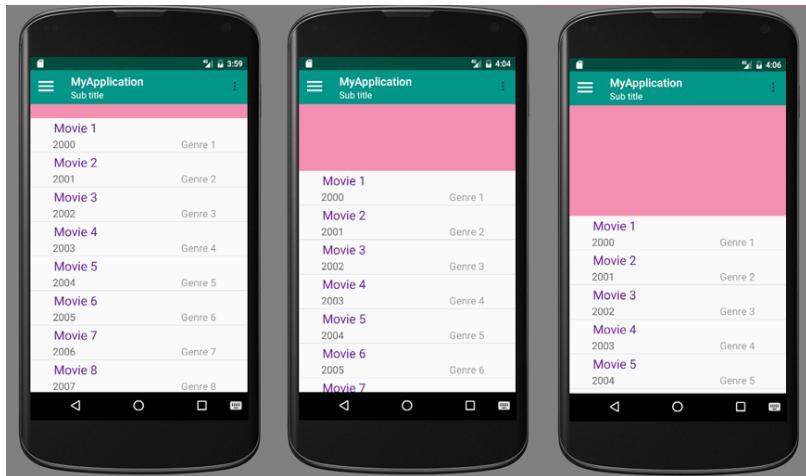


Рис. 5.5. Последовательность скриншотов, показывающая анимацию плавного появления панели добавления/редактирования фильма из Домашнего задания №3

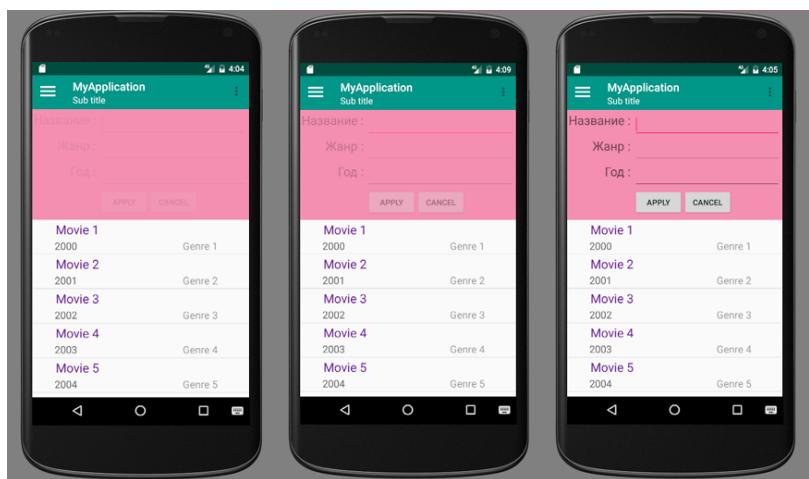


Рис. 5.6. Последовательность скриншотов (продолжение), показывающая анимацию плавного появления виджетов на панели добавления/редактирования фильма из Домашнего задания №3

Реализуйте функциональность внесения изменений в список фильмов. То есть, чтобы пользователь приложения мог добавлять, удалять, редактировать фильмы из списка фильмов. Сохранение списка фильмов в базу данных не является обязательным — реализация этой функциональности зависит от вашего желания и времени, и добавит вам дополнительные баллы к оценке за домашнее задание.

Также необходимо подобрать изображения для пиктограммы приложения, с различной детализацией, и разместить их в каталоге /res/mipmap. Назначить в Манифесте приложения пиктограмму для этого приложения.

Выполните локализацию данного приложения для 2-3 языков, на ваш выбор.

P.S. Рекомендуем для выполнения этого домашнего задания взять из модуля app10 только функциональность, относящуюся к анимации Value Animation, так как остальная функциональность не нужна, и будет только нагружать ваше решение.



Урок №5

Ресурсы приложения Android

© Бояршинов Юрий

© Компьютерная Академия «Шаг»

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.