

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ  
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

# ANDROID



# Урок №2

Структура  
Android-проекта.  
Пользовательский  
интерфейс  
приложения

## Содержание

<b>1. Программное создание объектов LayoutParams ..5</b>
1.1. Создание объектов LayoutParams для LinearLayout .....9
1.2. Создание объектов LayoutParams для TableLayout .....14
<b>2. Менеджер раскладки GridLayout .....20</b>
<b>3. Менеджер раскладки FrameLayout .....32</b>
<b>4. Элементы управления CheckBox, RadioButton..35</b>
4.1. CheckBox .....35
4.2. RadioButton, RadioGroup .....45
<b>5. Получение информации об ориентации устройства. Установка ориентации.....54</b>

<b>6. Меню приложения.....</b>	<b>58</b>
6.1. Создание меню .....	58
6.2. Обработка события выбора пункта меню .....	72
6.3. Контекстное меню .....	77
<b>7. Менеджер раскладки ScrollView.</b>	
<b>Создание прокручиваемых элементов.....</b>	<b>85</b>
<b>8. Создание виджетов с помощью LayoutInflater.....</b>	<b>95</b>
<b>9. Пример формирования списка с помощью LayoutInflater и размещение его в ScrollView .....</b>	<b>104</b>
<b>10. Понятие Адаптера данных (Adapter).</b>	
<b>Виды адаптеров данных.....</b>	<b>125</b>
10.1. Интерфейсы Adapter, ListAdapter, SpinnerAdapter. Абстрактный класс BaseAdapter .....	126
10.2. Класс ArrayAdapter<T> .....	129
10.3. Классы SimpleAdapter, CursorAdapter, SimpleCursorAdapter .....	135
<b>11. Виджет Spinner — выпадающий список .....</b>	<b>137</b>
11.1. Применение ArrayAdapter<T> для Spinner ..	139
11.2. Применение SimpleAdapter для Spinner ..	147
<b>12. Необходимость сохранения состояния Активности при повороте устройства.....</b>	<b>156</b>
12.1. Жизненный цикл Активности при повороте устройства.....	156

12.2. Пример, демонстрирующий необходимость сохранения состояний .....	161
12.3. Сохранение состояний Активности с помощью Bundle. Событие onSaveInstanceState.....	176
12.4. Использование статических полей для сохранения состояний .....	185
<b>13. Домашнее задание .....</b>	<b>188</b>

# 1. Программное создание объектов LayoutParams

В прошлом уроке мы с вами рассмотрели как создавать программно виджеты и менеджеры раскладки. Эта глава так же посвящена программной «верстке», а точнее программному созданию и настройке объектов **LayoutParams**. Если вы обратили внимание, то при верстке с помощью xml для виджетов и для менеджеров компоновки практически всегда необходимо указывать так называемые «layout parameters», т.е. «параметры расположения» (см. Листинг 1.1).

## Листинг 1.1. Использование Layout Parameters (см. Рис. 1.2)

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

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

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button One"
    />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="Button Two"
    />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:orientation="horizontal"
        android:background="#7DD0CD"
        android:gravity="center"
    >

        <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Button Three"
        />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:text="Button Four"
        />

    </LinearLayout>
</LinearLayout>
```

Эти параметры управляют расположением виджета (или менеджера раскладки) относительно других виджетов (чаще всего родительских). В Листинге 1.1 эти layout-параметры выделены жирным шрифтом: `android:layout_width`, `android:layout_height`, `android: layout_gravity`, `android: layout_margin`. Более подробное описание этих атрибутов находится в предыдущем уроке. Наша задача, научиться программно задавать виджетам эти атрибуты.

Так вот, для того чтобы управлять layout-параметрами в классах контейнерах (или в менеджерах раскладки) для этих целей существуют вложенные классы с названием `LayoutParams`. Если, к примеру, виджет `Button` добавляется в менеджер раскладки `TipLayout`, то необходимо создать объект экземпляр класса `TipLayout.LayoutParams` (см. Листинг 1.2).

### Листинг 1.2. Создание объекта LayoutParams

```
TipLayout.LayoutParams LP =  
    new TipLayout.LayoutParams(  
        ViewGroup.LayoutParams.MATCH_PARENT,  
        ViewGroup.LayoutParams.WRAP_CONTENT);
```

Как видно из Листинга 1.2, уже при создании объекта указываются значения для атрибутов `android:layout_width` и `android:layout_height`. При необходимости указать значение для атрибута `android:layout_marginTop`, можно воспользоваться соответствующим полем из объекта `LayoutParams`:

```
LP.marginTop = 20;
```



**Рис. 1.1.** Внешний вид верстки макета из Листинга 1.1

При добавлении виджета в объект контейнера (менеджера раскладки) используется один из двух вариантов:

```
Виджет.setLayoutParams (LP) ;  
МенеджерРаскладки.addView(Виджет) ;
```

ИЛИ

```
МенеджерРаскладки.addView(Виджет, LP) ;
```

Если виджет уже добавлен в контейнер, и необходимо поменять значения layout-параметров, то получить ссылку на объект **LayoutParams** можно следующим образом:

```
ТипLayout.LayoutParams LP = (ТипLayout.LayoutParams)  
    Виджет.getLayoutParams () ;  
LP.width = ViewGroup.LayoutParams.MATCH_PARENT;  
// Например меняем ширину
```

Объекты **LayoutParams** для разных менеджеров раскладки в целом имеют много общих свойств и методов,

но и отличия у них тоже имеются. Поэтому в этом уроке рассматриваются объекты для двух разных менеджеров раскладки: **LinearLayout** и **TableLayout**.

## 1.1. Создание объектов LayoutParams для LinearLayout

Layout-параметры для менеджера раскладки **LinearLayout** представлены классом **LinearLayout.LayoutParams**. Иерархия классов для этого класса следующая:

```
java.lang.Object
|
+-- android.view.ViewGroup.LayoutParams
|
+-- android.view.ViewGroup.MarginLayoutParams
|
+-- android.widget.LinearLayout.LayoutParams
```

Полное описание этого класса можно прочитать по ссылке <http://developer.android.com/reference/android/widget/LinearLayout.LayoutParams.html>.

Программная верстка макета (внешний вид которого изображен на Рис. 1.1) с использованием layout-параметров показана в Листинге 1.3. Исходный код этого примера можно найти в модуле «app» в файлах исходных кодов, которые прилагаются к этому уроку.

### Листинг 1.3. Программное создание и настройка LayoutParams для LinearLayout

```
public class MainActivity extends ActionBarActivity
{
    //---Class members-----
    /**
     * Менеджер раскладки LinearLayout,
```

```
* который будет назначен Активности
* для демонстрации применения LayoutParams
*/
private LinearLayout mainLL;

//--Class methods-----
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
// setContentView(R.layout.activity_main);
// !!! Закомментировано !!!

//--Создание главного LinearLayout (mainLL)---
    this.mainLL = new LinearLayout(this);
    this.mainLL.setOrientation(LinearLayout.VERTICAL);

//--Внутренние отступы для главного контейнера---
    this.mainLL.setPadding(
        (int)this.getResources().getDimension(R.dimen.
            activity_horizontal_margin),
        (int)this.getResources().getDimension(R.dimen.
            activity_vertical_margin),
        (int)this.getResources().getDimension(R.dimen.
            activity_horizontal_margin),
        (int)this.getResources().getDimension(R.dimen.
            activity_vertical_margin));

//--Создание LayoutParams для главного LinearLayout--
    LinearLayout.LayoutParams mainLinLP =
        new LinearLayout.LayoutParams(
            ViewGroup.LayoutParams.MATCH_PARENT,
            ViewGroup.LayoutParams.MATCH_PARENT);
//--Назначение LayoutParams менеджеру раскладки mainLL-
    this.mainLL.setLayoutParams(mainLinLP);

//--Добавление кнопки "Button One"-----
    Button B1 = new Button(this);
```

## 1. Программное создание объектов LayoutParams

```
B1.setText("Button One");

//--LayoutParams для кнопки "Button One"-----
LinearLayout.LayoutParams lp1 =
    new LinearLayout.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        // Ширина на всю родительскую область
        ViewGroup.LayoutParams.WRAP_CONTENT);
    // Высота равна высоте своего контента

this.mainLL.addView(B1, lp1);

//--Добавление кнопки "Button Two"-----
Button B2 = new Button(this);
B2.setText("Button Two");

//--LayoutParams для кнопки "Button Two"-----
LinearLayout.LayoutParams lp2 =
    new LinearLayout.LayoutParams(
        ViewGroup.LayoutParams.WRAP_CONTENT,
        // Ширина равна ширине своего контента
        ViewGroup.LayoutParams.WRAP_CONTENT);
    // Высота равно высоте своего контента

lp2.gravity= Gravity.RIGHT;
// "Причаливание" справа

B2.setLayoutParams(lp2);

this.mainLL.addView(B2);

//--Добавление вложенного контейнера LinearLayout
// с горизональной ориентацией
LinearLayout LL1 = new LinearLayout(this);
```

```
LL1.setOrientation(LinearLayout.HORIZONTAL);
    // Горизонтальная ориентация

LL1.setBackgroundColor(Color.rgb(0x7D, 0xD0, 0xCD));
    // Цвет фона, чтобы увидеть контейнер

//--LayoutParams для вложенного контейнера LL1---
LinearLayout.LayoutParams lpLL1 =
    new LinearLayout.LayoutParams(
        ViewGroup.LayoutParams.MATCH_PARENT,
        // Ширина на всю ширину родителя
        ViewGroup.LayoutParams.WRAP_CONTENT);
    // Высота равна высоте своего контента

lpLL1.topMargin =
    (int)this.getResources().
    getDimension(R.dimen.top_margin_20dp);
    // Верхний отступ 20dp

LL1.setLayoutParams(lpLL1);

LL1.setGravity(Gravity.CENTER_HORIZONTAL);
    // Дочерние виджеты выровнять по центру

this.mainLL.addView(LL1);

//--Добавление кнопки "Button Three" в контейнер LL1--
Button      B3      = new Button(this);
B3.setText("Button Three");

LL1.addView(B3, lp2);
// Воспользуемся уже существующим LayoutParams

//--Добавление кнопки "Button Four" в контейнер LL1-
```

```
Button      B4      = new Button(this);
B4.setText("Button Four");

LL1.addView(B4, lp2);
// Воспользуемся уже существующим LayoutParams

//--Назначаем Активности главный контейнер (mainLL) -
this.setContentView(this.mainLL);
}

...
}
```

Результат исполнения примера из Листинга 1.3 соответствует изображению, показанному на Рис. 1.1.

Отдельно необходимо обратить внимание на способ задания величины размера для отступа:

```
ТипLayout.LayoutParams LP =
    new ТипLayout.LayoutParams(...);
LP.marginTop = 20;           // Это не будет 20dp !!!
```

Задание значений для величин внешних и внутренних отступов (и других размеров) требует указания единиц измерения («dp», «sp», «px» и так далее). Просто число не содержит информацию о единице измерения и поэтому трактуется как размер в пикселях. Использование размеров в пикселях это плохая практика для Android программирования, так как размеры пикселей на разных устройствах разные. О единицах измерения для величин размеров можно прочесть по ссылке <http://developer.android.com/guide/topics/resources/more-resources.html#Dimension>. Простым решением это проблемы является добавление констант для величин размеров в файл ресурсов приложения имя\_модуля/res/values/dimens.xml.

Пример добавления константы для вертикального отступа нашего примера (Листинг 1.3) показан в Листинге 1.4.

#### Листинг 1.4. Добавление константы для вертикального отступа

```
<resources>
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="top_margin_20dp">20dp</dimen>
</resources>
```

Чтобы извлечь значение этой константы, используется код, пример которого показан в Листинге 1.5.

#### Листинг 1.5. Извлечение значения размера из файла ресурсов

```
lpLL1.topMargin = (int)this.getResources().
    getDimension(R.dimen.top_margin_20dp);
```

Как видно из Листинга 1.5. для извлечения значений из размеров используется метод **getDimension** объекта ресурсов приложения, которому в качестве параметра необходимо передать идентификатор ресурса, содержащего требуемое значение. Получить ссылку на объект ресурсов приложения можно с помощью вызова метода **getResources()** объекта Активности (этот метод класса Активности **android.app.Activity** наследует от класса Контекста приложения **android.content.Context**).

## 1.2. Создание объектов LayoutParams для TableLayout

Очень похоже программное создание и применение Layout-параметров для **TableLayout**. Особенность состоит в том, что менеджер раскладки **TableLayout** является

контейнером для объектов **TableRow**. Соответственно, Layout-параметры существуют и для менеджера раскладки **TableRow**. В этом разделе мы познакомимся с созданием Layout-параметров как для **TableLayout**, так и для **TableRow**.

Иерархия классов для класса **TableLayout.LayoutParams** выглядит следующим образом:

```
java.lang.Object
  |
  +--android.view.ViewGroup.LayoutParams
    |
    +--android.view.ViewGroup.MarginLayoutParams
      |
      +--- android.widget.LinearLayout.
          LayoutParams
            |
            +--- android.widget.TableLayout.
                LayoutParams
```

Полное описание класса **TableLayout.LayoutParams** можно прочитать по ссылке <http://developer.android.com/reference/android/widget/TableLayout.LayoutParams.html>.

Иерархия классов для класса **TableRow.LayoutParams**:

```
java.lang.Object
  |
  +--android.view.ViewGroup.LayoutParams
    |
    +--android.view.ViewGroup.MarginLayoutParams
      |
      +--android.widget.LinearLayout.LayoutParams
        |
        +--android.widget.TableRow.LayoutParams
```

Полное описание класса `TableRow.LayoutParams` можно прочитать по ссылке <http://developer.android.com/reference/android/widget/TableRow.LayoutParams.html>.

Давайте рассмотрим пример программного создания Layout-параметров для этих менеджеров. Код примера представлен в Листинге 1.6. Необходимо напомнить, что для дочерних виджетов, помещаемых в менеджер раскладки `TableRow` отсутствует необходимость обязательно указывать такие атрибуты как `android:layout_width` и `android:layout_height`. Поэтому в примере из Листинга 1.6 используются специфические настройки Layout-параметров, характерные для `TableLayout` и `TableRow` (Например, объединение ячеек `android:layout_span`, позиционирование в конкретный столбец `android:layout_column` и т.д.).

**Листинг 1.6.** Пример программного создания `TableLayout`, `LayoutParams` и `TableRow.LayoutParams`

```
public class MainActivity extends ActionBarActivity
{
    //--Class members-----
    ...
    /**
     * Менеджер раскладки TableLayout, который будет
     * назначен Активности для демонстрации применения
     * LayoutParams
     */
    private           TableLayout          mainTL;

    //--Class methods-----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

```
// setContentView(R.layout.activity_main);
// !!! Закомментировано !!!
...
/*
* Пример создания Layout-параметров для TableLayout
* и TableRow
* -----
*/
this.mainTL = new TableLayout(this);

//--Цвет фона, чтобы увидеть сам контейнер-----
this.mainTL.setBackgroundColor(Color.rgb(0x7D,
    0xD0, 0xCD));

//--Layout-параметры для TableLayout-----
TableLayout.LayoutParams mainTabLP =
        new TableLayout.LayoutParams();
mainTabLP.width =
        ViewGroup.LayoutParams.WRAP_CONTENT;
mainTabLP.height =
        ViewGroup.LayoutParams.WRAP_CONTENT;
this.mainTL.setLayoutParams(mainTabLP);

//--Первая строка (Layout-параметры не используются)-
TableRow          TR1      = new TableRow(this);
mainTL.addView(TR1);

Button            B11      = new Button(this);
B11.setText("One");
TR1.addView(B11);

Button            B12      = new Button(this);
B12.setText("Two");
TR1.addView(B12);

Button            B13      = new Button(this);
```

```
B13.setText("Three");
TR1.addView(B13);

//--Вторая строка-----
TableRow          TR2      = new TableRow(this);
mainTL.addView(TR2);

TableRow.LayoutParams L21      =
                    new TableRow.LayoutParams();
L21.column           = 1;
// Номер столбца, куда поместим виджет

Button            B21      = new Button(this);
B21.setText("One");
TR2.addView(B21, L21);

Button            B22      = new Button(this);
B22.setText("Two");
TR2.addView(B22);

//--Третья строка-----
TableRow          TR3      = new TableRow(this);
mainTL.addView(TR3);

Button            B31      = new Button(this);
B31.setText("One");
TR3.addView(B31);

TableRow.LayoutParams L32      =
                    new TableRow.LayoutParams();
L32.span           = 2;
// Объединяем две ячейки

Button            B32      = new Button(this);
B32.setText("Two");
TR3.addView(B32, L32);
}
```

Результат работы примера приложения из Листинга 1.6 изображен на Рис. 1.2.



**Рис. 1.2.** Результат работы примера из Листинга 1.6

Полную версию примера, показанного в Листингах 1.3, 1.4, 1.5, 1.6 можно найти в файлах с исходными кодами примеров, которые прилагаются к данному уроку, в модуле «app». В этом модуле для демонстрации работы Листингов 1.3 и 1.6 необходимо выбрать соответствующие опции меню приложения.

## 2. Менеджер раскладки GridLayout

Наше знакомство с менеджерами раскладки продолжается. В этом разделе рассматривается менеджер **android.widget.GridLayout**, который располагает дочерние виджеты по ячейкам в прямоугольной «сетке» (Сама «сетка» является всего лишь абстрактным понятием).

Иерархия классов для класса **android.widget.GridLayout**:

```
java.lang.Object
  |
  +--- android.view.View
      |
      +--- android.view.ViewGroup
          |
          +--- android.widget.GridLayout
```

Полное описание полей и методов класса **android.widget.GridLayout** можно прочитать по ссылке <http://developer.android.com/reference/android/widget/GridLayout.html>.

Пример xml верстки макета Активности (имя\_модуля/res/layout/activity\_main.xml) с использованием менеджера раскладки **android.widget.GridLayout** приведен в Листинге 2.1.

**Листинг 2.1.** Простейшая верстка с использованием **android.widget.GridLayout**

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

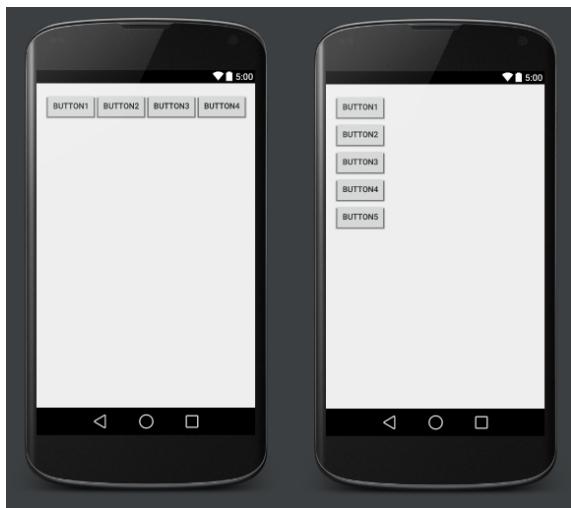
```
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"

android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"

android:orientation="horizontal"

tools:context=".MainActivity">

<Button android:text="Button1" />
<Button android:text="Button2" />
<Button android:text="Button3" />
<Button android:text="Button4" />
<Button android:text="Button5" />
</GridLayout>
```



**Рис. 2.1.** Внешний вид примера из Листинга 2.1 с горизонтальной  
ориентацией (слева) и вертикальной ориентации (справа)

Внешний вид примера Листинга 2.1 изображен на Рис. 2.1.

Как можно увидеть в Листинге 2.1 (выделено жирным шрифтом) у менеджера раскладки `android.widget.GridLayout` есть ориентация (вертикальная и горизонтальная, по умолчанию — горизонтальная). Ориентация задается атрибутом `android:orientation`, который может принимать значения `vertical` или `horizontal`. При горизонтальной ориентации дочерние виджеты заполняются построчно, а при вертикальной ориентации — по столбцам. В примере Листинга 2.1 не указывалось сколько строк и столбцов должно быть у `GridView`. Поэтому, по умолчанию при горизонтальной ориентации `GridView` содержит одну строку, в которой будет столбцов столько, сколько дочерних виджетов будет размещено в `GridView`. Если виджетов будет много — то они не поместятся на экране, как видно из левого изображения макета Активности на Рис. 2.1.

При вертикальной ориентации `GridView` содержит один столбец, и дочерние виджеты располагаются в строках этого столбца с последующим выходом за границы экрана если виджетов будет много. Поэтому для менеджера раскладки `android.widget.GridView` нужно задавать, сколько у него будет строк и столбцов. Это делается с помощью атрибутов `android:rowCount` (количество строк) и `android:columnCount` (количество столбцов). Добавим в пример эти атрибуты и укажем количество строк равным 4, а количество столбцов равным 3. И еще добавим побольше дочерних виджетов. Изменения указаны на Листинге 2.2.

**Листинг 2.2.** Добавление атрибутов  
android:rowCount и android:columnCount в GridLayout

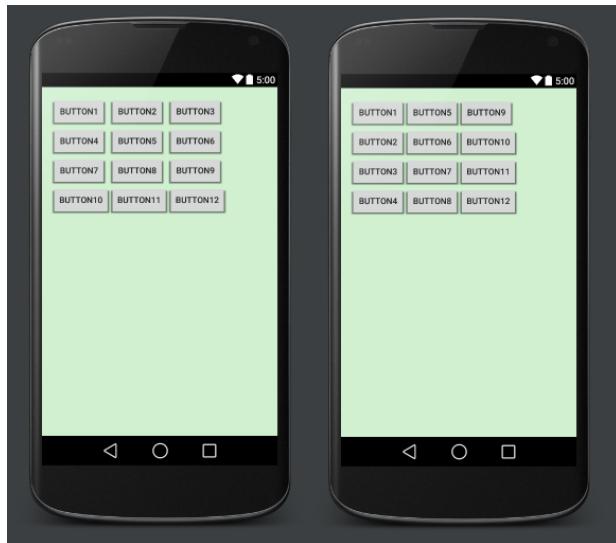
```
<GridLayout
    ...
    android:orientation="horizontal"
    android:rowCount="4"
    android:columnCount="3"
    android:background="#D0F0D0"
    tools:context=".MainActivity">

    <Button android:text="Button1" />
    <Button android:text="Button2" />
    <Button android:text="Button3" />
    <Button android:text="Button4" />
    <Button android:text="Button5" />
    <Button android:text="Button6" />
    <Button android:text="Button7" />
    <Button android:text="Button8" />
    <Button android:text="Button9" />
    <Button android:text="Button10" />
    <Button android:text="Button11" />
    <Button android:text="Button12" />
</GridLayout>
```

На Рис. 2.2 можно увидеть применение атрибутов **android:rowCount** и **android:columnCount**. Обратите еще раз внимание, какую роль играет атрибут **android:orientation** для **GridView** — это способ заполнения дочерних виджетов — по строкам или по столбцам.

Так же у **GridLayout** существует возможность явно указывать при размещении в нем дочернего виджета в какую строку и столбец его разместить. Для этого предназначены атрибуты **android:layout\_column** и **android:layout\_row**, которым присваиваются числовые значения номера столбца

и строки соответственно. Эти атрибуты указываются для дочерних виджетов. В Листинге 2.3 приводится пример использования таких атрибутов.



**Рис. 2.2.** Внешний вид примера из Листинга 2.2.  
Слева горизонтальная ориентация, справа — вертикальная

**Листинг 2.3.** Использование атрибутов  
android:layout\_row и android:layout\_column

```
<GridLayout
    ...
    android:orientation="horizontal"
    android:rowCount="4"
    android:columnCount="3"
    tools:context=".MainActivity">

    <Button android:text="Button1" />
    <Button android:text="Button2" />
    <Button android:text="Button3" />
```

```
<Button android:text="Button4" />
<Button android:text="Button5" />
<Button android:text="Button6" />
<Button android:text="Button7" />
<Button android:text="Button8" android:
    layout_row="3" android:layout_column="0"/>
<Button android:text="Button9" />
<Button android:text="Button10" />

</GridLayout>
```

Внешний вид результата примера из Листинга 2.3 изображен на Рис. 2.3.



**Рис. 2.3.** Внешний вид примера из Листинга 2.3

В примерах из Листингов 2.2 и 2.3 менеджеру раскладки **android.widget.GridLayout** задается цвет фона:

```
    android:background="#D0F0D0"
```

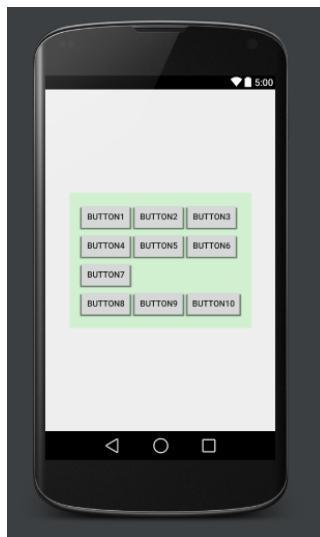
Это делается с целью показать размеры, который занимает менеджер **GridLayout** в наших примерах. Как видите (см. Рис. 2.2 и 2.3) менеджер раскладки **GridLayout** растягивается на всю область экрана, что и задавалось с помощью атрибутов:

```
android:layout_width="match_parent"  
android:layout_height="match_parent"
```

Как видите, при значениях атрибутов **android:layout: width** и **android:layout:height** равными значению **match\_parent**, дочерние виджеты притягиваются к левому дочернему углу менеджера **GridLayout**. Если необходимо разместить менеджер **GridLayout** со своими дочерними виджетами например по центру экрана, то это можно сделать следующим образом, как показано в Листинге 2.4.

#### Листинг 2.4. Выравнивание по центру экрана менеджера GridLayout

```
<GridLayout  
    ...  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center"  
  
    android:orientation="horizontal"  
    android:rowCount="4"  
    android:columnCount="3"  
    android:background="#D0F0D0"  
  
    tools:context=".MainActivity">  
    ...  
</GridLayout>
```



**Рис. 2.4.** Внешний вид примера из Листинга 2.4

Как видно из Рис. 2.4, чтобы позиционировать **GridLayout** с помощью атрибута **android:layout\_gravity**, рекомендуется присвоить атрибутам **android\_layout\_width** и **android\_layout\_height** значение **wrap\_content**.

В Листинге 2.5 приведен пример программного создания менеджера **android.widget.GridLayout**, соответствующий изображению макета на Рис. 2.4.

**Листинг 2.5.** Программная верстка GridLayout  
соответствующая изображению на Рис. 2.4

```
public class MainActivity extends ActionBarActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

```
// setContentView(R.layout.activity_main);
// Закомментировано !!!
```

```
/*
 * Программная верстка менеджера раскладки
 * android.widget.GridLayout
 * -----
 */
GridLayout GL      = new GridLayout(this);
```

```
//--Горизонтальная ориентация-----
GL.setOrientation(GridLayout.HORIZONTAL);
```

```
//--Цвет фона-----
GL.setBackgroundColor(Color.rgb(0xD0, 0xF0, 0xD0));
```

```
//--Количество строк и столбцов-----
GL.setColumnCount(3);
GL.setRowCount(4);
```

```
//--Layout-параметры-----
GridLayout.LayoutParams LP =
        new GridLayout.LayoutParams();
LP.width = ViewGroup.LayoutParams.WRAP_CONTENT;
LP.height = ViewGroup.LayoutParams.WRAP_CONTENT;
LP.setGravity(Gravity.CENTER);
GL.setLayoutParams(LP);
```

```
//--Добавление дочерних виджетов-----
for (int i = 1; i <= 7; i++)
{
    Button B      = new Button(this);
    B.setText("Button" + i);
    GL.addView(B);
}
```

```
//--Для 8-й кнопки укажем позицию строки и столбца--
Button     B8      = new Button(this);
```

```
B8.setText("Button8");

//--Номер строки, куда будет помещаться кнопка---
GridLayout.Spec rowSpec = GridLayout.spec(3);

//--Номер столбца, куда будет помещаться кнопка---
GridLayout.Spec colSpec = GridLayout.spec(0);

//--Layout-параметры для кнопки №8-----
GridLayout.LayoutParams lp =
    new GridLayout.LayoutParams(rowSpec, colSpec);

//--Добавляем кнопку в GridLayout-----
GL.addView(B8, lp);

//--Добавление оставшихся виджетов-----
for (int i = 9; i <= 10; i++)
{
    Button B = new Button(this);
    B.setText("Button" + i);
    GL.addView(B);
}

//--Назначаем главный контейнер GridLayout Активности-
this.setContentView(GL);
}

...
}
```

Пример Листинга 2.5 можно найти в модуле «app2» файлов с исходными кодами примеров, которые прилагаются к этому уроку.

В Листинге 2.5 дополнительно жирным выделен код, показывающий, как программно указывать номер строки и столбца для добавляемого виджета. Для этой

цели предназначен класс `android.widget.GridLayout.Spec`. Ознакомится с документацией по этому классу можно по ссылке <http://developer.android.com/reference/android/widget/GridLayout.Spec.html>.

Так же в менеджере раскладки `android.widget.GridLayout` существует возможность объединять строки и столбцы с помощью атрибутов `android:layout_rowSpan` и `android:layout_columnSpan`. Эти атрибуты принимают числовое значение, указывающее сколько строк или столбцов объединяется. В Листинге 2.6 показан пример использования этих атрибутов.

**Листинг 2.6.** Применение атрибутов  
`android:layout_columnSpan` и `android:layout_rowSpan`

```
<GridLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"

    android:orientation="horizontal"
    android:rowCount="4"
    android:columnCount="3"
    android:background="#D0F0D0"
    tools:context=".MainActivity">

    <Button android:text="Button1" />
    <Button android:text="Button2" />
    <Button android:text="Button3" />

    <Button android:text="Button4" />
```

```
    android:layout_columnSpan="2"
    android:layout_gravity="fill"
/>

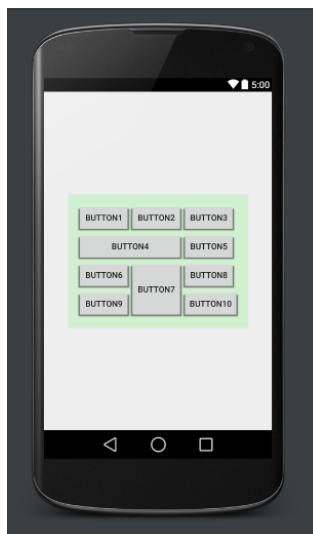
<Button android:text="Button5" />
<Button android:text="Button6" />

<Button android:text="Button7"
        android:layout_rowSpan="2"
        android:layout_gravity="fill"
/>

<Button android:text="Button8" />
<Button android:text="Button9" />
<Button android:text="Button10" />

</GridLayout>
```

Внешний вид результата примера из Листинга 2.6 изображен на Рис. 2.5.



**Рис. 2.5.** Внешний вид примера из Листинга 2.6

### 3. Менеджер раскладки FrameLayout

Менеджер раскладки **android.widget.FrameLayout** является достаточно простым менеджером. Он предназначен для содержания одного дочернего виджета. Если в **android.widget.FrameLayout** поместить несколько дочерних виджетов, то они будут расположены друг над другом в Z-порядке. Применив к этим виджетам атрибут **android:layout\_gravity**, можно добиться более-менее приемлемого расположения этих виджетов, но все-таки основное предназначение этого менеджера раскладки — содержание одного дочернего виджета. Какой в этом смысл? Ответ заключается в том, что в некоторых случаях бывает сложно разместить дочерние виджеты таким образом, чтобы изменение их размера при отображении на разных экранах не приводило бы к тому, чтобы эти виджеты не налазили бы друг на друга. Так вот менеджер раскладки **android.widget.FrameLayout** предназначен именно для решения подобных проблем.

Иерархия классов для класса **android.widget.FrameLayout**:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.FrameLayout
```

Описание полей и методов доступно по ссылке <http://developer.android.com/reference/android/widget/FrameLayout.html>.

Пример, с использованием менеджера раскладки **android.widget.FrameLayout** представлен в Листинге 3.1. Результат примера из Листинга 3.1 изображен на Рис. 3.1.



**Рис. 3.1.** Внешний вид макета Активности из примера в Листинге 3.1

В примере Листинга 3.1 в качестве дочернего виджета для менеджера раскладки **android.widget.FrameLayout** используется **android.widget.GridLayout**.

**Листинг 3.1.** Пример использования менеджера раскладки FrameLayout

```
<FrameLayout  
    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:background="#D0E0D0"
    tools:context=".MainActivity">

    <GridLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:columnCount="3"
        android:rowCount="3"
        android:layout_gravity="center">

        <Button android:text="1" />
        <Button android:text="2" />
        <Button android:text="3" />
        <Button android:text="4" />
        <Button android:text="5" />
        <Button android:text="6" />
        <Button android:text="7" />
        <Button android:text="8" />
        <Button android:text="9" />
    </GridLayout>
</FrameLayout>
```

# 4. Элементы управления CheckBox, RadioButton

## 4.1. CheckBox

Виджет `android.widget.CheckBox` представляет собой кнопку-флажок, которая может находиться в одном из двух состояний: «выбран» / «не выбран». Иерархия классов для `android.widget.CheckBox`:

```
java.lang.Object
 |
 +-- android.view.View
   |
   +-- android.widget.TextView
     |
     +-- android.widget.Button
       |
       +-- android.widget.CompoundButton
         |
         +-- android.widget.CheckBox
```

Документация API для этого класса доступна по ссылке <http://developer.android.com/reference/android/widget/CheckBox.html>. Как видно из иерархии классов для класса `android.widget.CheckBox`, он является прямым наследником класса `android.widget.CompoundButton` (<http://developer.android.com/reference/android/widget/CompoundButton.html>) и именно этот класс определяет функциональность класса `android.widget.CheckBox`.

Создать виджет `android.widget.CheckBox` с помощью XML можно следующим образом (Листинг 4.1):

## Листинг 4.1. Пример XML верстки для виджета android.widget.CheckBox

```
<CheckBox  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="16pt"  
    android:textColor="#006699"  
    android:text="Java"  
    android:id="@+id/cbJava"  
    android:checked="true"  
    android:onClick="имя_обработчика_события_клика"  
/>
```

Как видно из Листинга 4.1 для xml-элемента **<CheckBox>** можно применять следующие атрибуты:

- **android:layout\_width** и **android:layout\_height** — уже знакомые нам атрибуты, применяемые для всех виджетов — задают механизм размещения виджета по ширине и по высоте соответственно.
- **android:textSize** — так же применим для всех виджетов, задает размер шрифта для отображаемого текста (не обязательный атрибут).
- **android:textColor** — задает цвет текста (применим для всех виджетов, необязательный атрибут).
- **android:text** — содержит текст, который выводится рядом с виджетом (не обязательен, но рекомендуемый атрибут).
- **android:id** — применим для всех виджетов, задает уникальный идентификатор виджета.
- **android:checked** — устанавливает состояние флагка («выбран»/«не выбран»), принимает значения **true**

или false. Не обязательный атрибут, по умолчанию содержит значение false, что соответствует состоянию флашка «не выбран».

- **android:onClick** — не обязательный атрибут. Задает имя метода класса Активности, который является обработчиком события клика по виджету **android.widget.CheckBox**. Метод должен быть объявлен в классе Активности: **public void Имя\_Метода (View view)**.

Внешний вид виджета, сверстанного с помощью xml из Листинга 4.1 изображен на Рис. 4.1:



**Рис. 4.1.** Внешний вид виджета из Листинга 4.1

Программно можно узнать состояние виджета **CheckBox** с помощью метода

```
public boolean isChecked();
```

Пример, получения состояния **CheckBox** (Листинг 4.2):

**Листинг 4.2.** Получение состояния виджета  
**android.widget.CheckBox**

```
public boolean isChecked();
CheckBox cbJava = (CheckBox)
    this.findViewById(R.id.cbJava);
boolean isCbJavaChecked = cbJava.isChecked();
if (isCbJavaChecked)
{}
```

```
//--Флажок находится в состоянии "Выбран"-----  
else  
{  
//--Флажок находится в состоянии "Не выбран"-----  
}
```

Установить программно состояние флажка **android.widget.CheckBox** можно с помощью метода

```
void setChecked(boolean checked);
```

Пример установки состояния для **android.widget.CheckBox** (Листинг 4.3):

**Листинг 4.3.** Установка состояния виджета  
**android.widget.CheckBox**

```
CheckBox cbJava = (CheckBox) this.  
        findViewById(R.id.cbJava);  
boolean isCbJavaChecked=  
        Получаем_значение_для_флажка;  
if (isCbJavaChecked)  
{  
//--Флажок нужно установить в состояние "Выбран"--  
    cbJava.setChecked(true);  
}  
else  
{  
//--Флажок нужно установить в состояние "Не выбран"--  
    cbJava.setChecked(false);  
}
```

Кроме события **onClick** у виджета **android.widget.CheckBox** есть событие **onCheckedChangeListener**, которое срабатывает при изменении состояния флажка. Назначить обработчик этого события можно с помощью метода:

```
public void setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener listener);
```

Метод **setOnCheckedChangeListener** принимает в качестве параметра ссылку на объект, реализующий интерфейс **CompoundButton.OnCheckedChangeListener** (<http://developer.android.com/reference/android/widget/CompoundButton.OnCheckedChangeListener.html>). В этом интерфейсе объявлен один метод, который и будет вызван при срабатывании события изменения состояния **android.widget.CheckBox**:

```
public abstract void onCheckedChanged (CompoundButton buttonView, boolean isChecked);
```

Метод принимает два параметра:

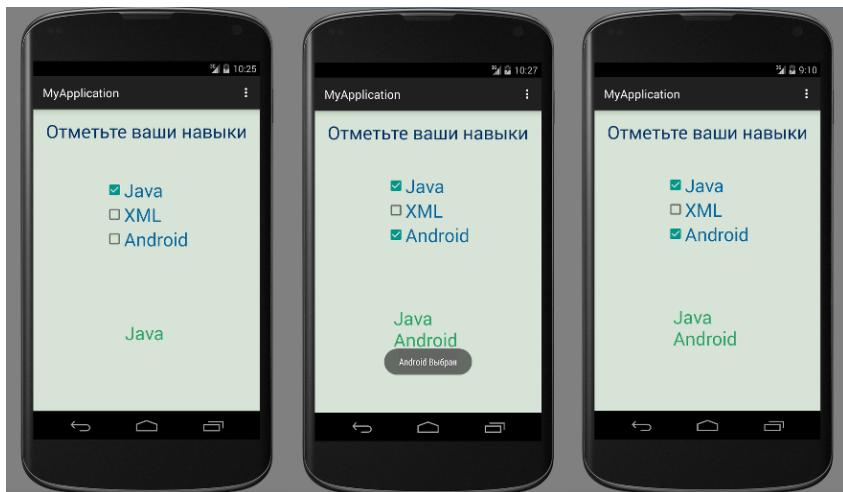
- **CompoundButton buttonView** — ссылка на виджет, который является источником события.
- **boolean isChecked** — признак «выбран» / «не выбран» (true / false соответственно).

Пример назначения обработчика события смены состояния флажка для **android.widget.CheckBox** приведен в Листинге 4.4.

#### **Листинг 4.4.** Пример назначения обработчика события смены состояния флажка в **android.widget.CheckBox**

```
CheckBox cbJava =
    (CheckBox) this.findViewById(R.id.cbJava);
cbJava.setOnCheckedChangeListener(
    new CompoundButton.OnCheckedChangeListener()
{
```

```
@Override
public void onCheckedChanged(
CompoundButton buttonView, boolean isChecked)
{
    String      msg     = (isChecked) ? "Java
    Выбран": "Java Не Выбран";
    Toast.makeText(MainActivity.this, msg,
    Toast.LENGTH_SHORT).show();
}
});
```



**Рис. 4.2.** Внешний вид работы примера из Листинга 4.5 — выбирается CheckBox с надписью «Android»

Для закрепления знаний о виджете **android.widget.CheckBox** рассмотрим пример, в котором пользователю приложения предлагается отметить навыки (предметы), которыми он владеет — «Java», «Xml», «Android». Каждому навыку соответствует отдельный CheckBox. Отмечая или снимая флажок для соответствующего навыка, срабатывает

событие и на экране устройства отображается Тост с информацией о том, какое действие произошло (навык «Выбран» / «Не Выбран») и в нижней части экрана отображается суммарная информация об отмеченных пользователем навыках. Полный исходный текст примера находится в модуле «app3» в файлах с исходными кодами примеров, которые прилагаются к данному уроку. Верстка xml-макета этого примера, ввиду несложности, в листингах урока не приводится — как уже говорилось, в макете находится три checkBox'а (`android.widget.CheckBox`) и Текстовое поле (`android.widget.TextView`), их размещение на макете можно найти самостоятельно в файле `app3/res/layout/activity_main.xml`. Пример исходного кода Java с обработкой событий приведен в Листинге 4.5. В качестве дополнения к пониманию работы примера приводится иллюстрация, изображенная на Рис. 4.2, которая показывает последовательность работы приложения при выборе или отмены выбора любого из виджетов `android.widget.CheckBox`.

**Листинг 4.5.** Пример обработки события смены состояния флагжа CheckBox одним объектом для нескольких виджетов `android.widget.CheckBox`

```
public class MainActivity extends ActionBarActivity
{
    //--Inner Classes -----
    /**
     * Класс, MyCheckedChangeListener реализующий
     * интерфейс CompoundButton.OnCheckedChangeListener.
     * Объект этого класса будет назначен в качестве
     * обработчика события смены состояния флагков
     * (объектов CheckBox) : cbJava, cbXml, cbAndroid
```

```
* (Один объект-обработчик для нескольких объектов
* CheckBox)
*/
class MyCheckedChangelistener implements
    CompoundButton.OnCheckedChangeListener
{
    @Override
    public void onCheckedChanged(CompoundButton
                                buttonView, boolean isChecked)
    {
        String msg = "";

        switch (buttonView.getId())
        {
//--Смена состояния флагка "Java"-----
            case R.id.cbJava :
                msg = (isChecked)?"Java
                    Выбран":"Java Не Выбран";
                break;

//--Смена состояния флагка "XML"-----
            case R.id.cbXml :
                msg = (isChecked)?"XML
                    Выбран":"XML Не Выбран";
                break;

//--Смена состояния флагка "Android"-----
            case R.id.cbAndroid :
                msg = (isChecked)?"Android
                    Выбран":"Android Не Выбран";
                break;
        }

//--Вывод тоста о смене состояния флагка-----
        Toast.makeText(MainActivity.this, msg,
                    Toast.LENGTH_SHORT).show();
    }
}
```

```
//---Формирование суммарной информации о состоянии
//---всех CheckBox'ов-----
        String info = "";
        info += (MainActivity.this.cbJava.
                  isChecked())?"Java\n":"";
        info += (MainActivity.this.cbXml.
                  isChecked())?"Xml\n":"";
        info += (MainActivity.this.cbAndroid.
                  isChecked())?"Android\n":"";

        MainActivity.this.tvInfo.setText(info);
    }

}

//---Class members-----
/***
 * Ссылка на CheckBox с надписью "Java",
 * который будет использован для примера обработки
 * события смены состояния OnCheckedChangeListener
 */
private CheckBox cbJava;

/***
 * Ссылка на CheckBox с надписью "XML",
 * который будет использован для примера обработки
 * события смены состояния OnCheckedChangeListener
 */
private CheckBox cbXml;

/***
 * Ссылка на CheckBox с надписью "Android",
 * который будет использован для примера обработки
 * события смены состояния OnCheckedChangeListener
 */
private CheckBox cbAndroid;

/**
 * Ссылка на TextView. в котором будет выводится
 * суммарная информация о состоянии всех флагков
```

```
* (объектов CheckBox)
*/
private TextView tvInfo;

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

//--Создание объекта-обрабочтика события
//--изменения состояния -----
MyCheckedChangeListenermyCCL =
        new MyCheckedChangeListener();

//--Инициализация полей-ссылок на CheckBox
//--и назначение им обработчика-----
    this.cbJava = (CheckBox) this.
                    findViewById(R.id.cbJava);
this.cbJava.setOnCheckedChangeListener(myCCL);

    this.cbXml = (CheckBox) this.
                    findViewById(R.id.cbXml);
this.cbXml.setOnCheckedChangeListener(myCCL);

    this.cbAndroid = (CheckBox) this.
                    findViewById(R.id.cbAndroid);
this.cbAndroid.setOnCheckedChangeListener(myCCL);

//--Инициализация TextView для вывода суммарной
//--информации-----
    this.tvInfo= (TextView) this.
                    findViewById(R.id.tvInfo);
}

...
}
```

Из примера в Листинге 4.5, для обработчика события создается вложенный класс `MyCheckedChangeListener`, который реализует интерфейс `CompoundButton.OnCheckedChangeListener`. На основе этого класса в методе `onCreate` создается один объект, который последовательно назначается обработчиком события смены состояния нескольким виджетам `android.widget.CheckBox`.

## 4.2. RadioButton, RadioGroup

Виджет `android.widget.RadioButton` представляет кнопку, которая может находиться в одном из двух состояний — «выбран» / «не выбран». Но, в отличии от `android.widget.CheckBox`, виджеты `android.widget.RadioButton` не используются по отдельности, а предназначены для использования в группе. Пользователь имеет возможность отметить (перевести в состояние «выбран») только один виджет из группы, при этом, виджет этой же группы, который был в состоянии «выбран» автоматически перейдет в состояние «не выбран». Для объединения виджетов `android.widget.RadioButton` в группу, используется виджет-контейнер `android.widget.RadioGroup`. Иерархия классов для `android.widget.RadioButton`:

```
java.lang.Object
 |
 +-- android.view.View
   |
   +-- android.widget.TextView
     |
     +-- android.widget.Button
       |
       +-- android.widget.CompoundButton
         |
         +-- android.widget.RadioButton
```

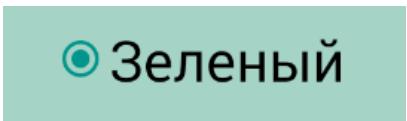
Как видно из иерархии классов, для класса `android.widget.RadioButton`, так же как и для класса `android.widget.CheckBox` родительским классом является класс `android.widget.CompoundButton`. Это означает, что принцип работы с `android.widget.RadioButton` очень похож на работу с виджетом `android.widget.CheckBox`. Подробное описание полей и методов класса `android.widget.RadioButton` — <http://developer.android.com/reference/android/widget/RadioButton.html>.

Создать виджет `android.widget.RadioButton` с помощью XML можно следующим образом (Листинг 4.6):

**Листинг 4.6.** Пример создания виджета `android.widget.RadioButton` с помощью XML

```
<RadioButton  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Зеленый"  
    android:id="@+id/rbGreen"  
    android:checked="true"  
    android:textSize="14pt"  
/>
```

Атрибуты и их значения для виджета `android.widget.RadioButton` такие же как и для виджета `android.widget.CheckBox`. Внешний вид виджета `android.widget.RadioButton` созданного в Листинге 4.6 представлен на Рис. 4.3.



● Зеленый

**Рис. 4.3.** Внешний вид виджета из Листинга 4.6

Как уже говорилось выше, виджеты `android.widget.RadioButton` не используются по одиночке, а используются в группе. Для объединения в группу используется виджет-контейнер `android.widget.RadioGroup` (<http://developer.android.com/reference/android/widget/RadioGroup.html>). Иерархия классов для класса `android.widget.RadioGroup` выглядит так:

```
java.lang.Object
|
+-- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.LinearLayout
|
+--- android.widget.RadioGroup
```

Как видно из иерархии классов, класс `android.widget.RadioGroup` является прямым наследником менеджера раскладки `android.widget.LinearLayout`. Следовательно, `android.widget.RadioGroup` позволяет не только объединять радиокнопки в группу, но и позволяет управлять внешним видом группы радиокнопок, начиная с ориентации (вертикальная или горизонтальная), выравниванием, расположением и всеми теми, что может менеджер раскладки `android.widget.LinearLayout`.

Создать виджет-контейнер `android.widget.RadioGroup` с помощью XML можно следующим образом (Листинг 4.7):

#### **Листинг 4.7.** Создание `android.widget.RadioGroup` с помощью XML

```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:orientation="vertical"
    android:id="@+id/rg1"
    android:layout_gravity="center_horizontal">
    ...
</RadioGroup>
```

Разместим в созданной группе `android.widget.Radio Group` несколько виджетов `android.widget.RadioButton` (Листинг 4.8).

**Листинг 4.8.** Создание группы виджетов `android.widget.RadioButton` с помощью `android.widget.RadioGroup`

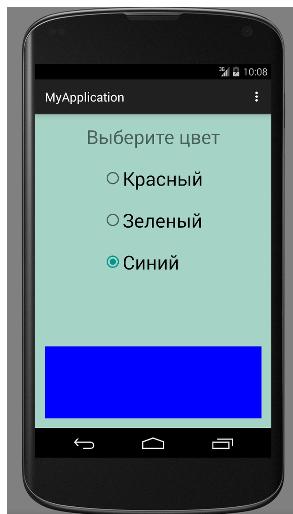
```
<RadioGroup
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:id="@+id/rg1"
    android:layout_gravity="center_horizontal">

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Красный"
        android:id="@+id/rbRed"
        android:layout_marginTop="12pt"
        android:textSize="14pt"
        />

    <RadioButton
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Зеленый"
        android:id="@+id/rbGreen"
        android:layout_marginTop="12pt"
```

```
        android:textSize="14pt"
    />

<RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Синий"
    android:checked="true"
    android:id="@+id/rbBlue"
    android:layout_marginTop="12pt"
    android:textSize="14pt"
    />
</RadioGroup>
```



**Рис. 4.4.** Внешний вид примера из Листингов 4.8 и 4.9

Виджет **android.widget.RadioButton** может обрабатывать событие смены состояния — аналогично виджету **android.widget.CheckBox**. Для назначения обработчика события смены состояния радиокнопки используется метод:

```
public void setOnCheckedChangeListener(CompoundButton.  
    OnCheckedChangeListener listener);
```

Объект обработчика события смены состояния радиокнопки должен реализовывать интерфейс **CompoundButton.OnCheckedChangeListener** (<http://developer.android.com/reference/android/widget/CompoundButton.OnCheckedChangeListener.html>). Об этом интерфейсе уже рассказывалось в предыдущем разделе этого урока.

Завершим изложение материала о виджете **android.widget.RadioButton** примером, исходный код которого находится в модуле «app4» среди файлов исходного кода, прилагаемых к данному уроку.

Суть примера заключается в следующем: с помощью группы радиокнопок пользователь выбирает цвет фона для виджета, расположенного в нижней части приложения (см. Рис. 4.4). в примере обрабатывается событие смены состояния радиокнопок, и в зависимости от выбранной радиокнопки виджету внизу экрана назначается соответствующий цвет. Для нескольких радиокнопок используется один объект-обработчик события, поэтому в примере происходит определение — кто является источником события. Так же нужно отметить, что в данном случае, при переключении между радиокнопками, событие будет срабатывать дважды — один раз для выбранной радиокнопки (переходит в состояние «выбран»), второй раз для той радиокнопки, которая была выбрана (переходит в состояние «не выбран»), поэтому смена цвета осуществляется только при событии перехода радиокнопки в состояние «выбран». Код Java примера из модуля «app4» приведен в Листинге 4.9.

Исходный код XML-верстки макета Активности в листингах этого урока приводится не будет (см. файл app4/res/layout/activity\_main.xml). В качестве виджета, которому меняется цвет фона выступает менеджер раскладки **android.widget.FrameLayout** без дочерних виджетов.

#### Листинг 4.9. Пример приложения, изображенного на Рис. 4.4

```
public class MainActivity extends Activity
{
    //---Inner Classes-----
    class MyCheckedChangeListener implements
        CompoundButton.OnCheckedChangeListener
    {
        @Override
        public void onCheckedChanged(CompoundButton
            buttonView, boolean isChecked)
        {
            //---Событие возникает и когда isChecked ==
            //---false, пропустим его-----
            if (isChecked)
            {
                if (buttonView.getId() == MainActivity.
                    this.rbRed.getId())
                {
                    //---Выбрана радиокнопка с надписью "Красный"-----
                    MainActivity.this.flColor.
                        setBackgroundColor(Color.RED);
                }
                else
                if (buttonView.getId() == MainActivity.
                    this.rbGreen.getId())
                {
                    //---Выбрана радиокнопка с надписью "Зеленый"-----
                    MainActivity.this.flColor.
                        setBackgroundColor(Color.GREEN);
                }
            }
        }
    }
}
```

```
        else
            if (buttonView.getId() ==
                MainActivity.this.rbBlue.getId())
            {
                //---Выбрана радиокнопка с надписью "Синий"-----
                MainActivity.this.flColor.
                    setBackgroundColor(Color.BLUE);
            }
        }

//---Class members-----
/***
 * Ссылка на радиокнопку с надписью "Красный"
 */
private RadioButton rbRed;

/***
 * Ссылка на радиокнопку с надписью "Зеленый"
 */
private RadioButton rbGreen;

/***
 * Ссылка на радиокнопку с надписью "Синий"
 */
private RadioButton rbBlue;

/***
 * Ссылка на виджет, которому будет меняться цвет
 * фона, в зависимости от выбранной радиокнопки
 */
private FrameLayout flColor;

//---Class methods-----
@Override
protected void onCreate(Bundle savedInstanceState)
{
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

//--Инициализация полей объекта-----
this.rbRed      = (RadioButton) this.
                    findViewById(R.id.rbRed);
this.rbGreen    = (RadioButton) this.
                    findViewById(R.id.rbGreen);
this.rbBlue     = (RadioButton) this.
                    findViewById(R.id.rbBlue);
this.flColor= (FrameLayout) this.
                    findViewById(R.id.flColor);

//--Назначение обработчика события смены состояния
//--для радиокнопок-----
MyCheckedChangeListenermyCCL =
        new MyCheckedChangeListener();

this.rbRed.setOnCheckedChangeListener(myCCL);
this.rbGreen.setOnCheckedChangeListener(myCCL);
this.rbBlue.setOnCheckedChangeListener(myCCL);
}

...
}
```

# 5. Получение информации об ориентации устройства. Установка ориентации

Как уже рассказывалось в предыдущем уроке, для разных ориентаций устройства есть возможность создавать соответствующие макеты Активностей. При смене ориентации устройства, Активность создается заново с соответствующим текущей ориентации устройства макетом. В связи с этим, в приложениях не редко возникает необходимость определять текущую ориентацию устройства. Пример получения информации об ориентации устройства приведен в Листинге 5.1.

## Листинг 5.1. Определение ориентации устройства

```
if(getResources().getConfiguration().orientation ==  
    Configuration.ORIENTATION_PORTRAIT)  
{  
    Log.d(TAG, "Портретная ориентация");  
}  
else  
if(getResources().getConfiguration().orientation ==  
    Configuration.ORIENTATION_LANDSCAPE)  
{  
    Log.d(TAG, "Альбомная ориентация");  
}
```

Определение ориентации устройства (Листинг 5.1) осуществляется с помощью объекта класса **android.content**.

`res.Configuration` (<http://developer.android.com/reference/android/content/res/Configuration.html>), который доступен через объект ресурсов приложения `android.content.res.Resources` (<http://developer.android.com/reference/android/content/res/Resources.html>). Получить ссылку на ресурсы приложения можно с помощью метода класса `android.content.Context` (класс Активности наследует этот класс):

```
public Resources getResources();
```

Пример, приведенный в Листинге 5.1 применим для использования в любом методе класса Активности.

Пример программной смены ориентации устройства приведен в Листинге 5.2 (пример применим для использования в любом методе класса Активности):

### Листинг 5.2. Программная смена ориентации устройства

```
if(getResources().getConfiguration().orientation ==
        Configuration.ORIENTATION_PORTRAIT)
{
    Log.d(TAG, "Портретная ориентация,
               меняем на альбомную");
    this.setRequestedOrientation (ActivityInfo.
                                SCREEN_ORIENTATION_LANDSCAPE);
}
else
if(getResources().getConfiguration().orientation ==
        Configuration.ORIENTATION_LANDSCAPE)
{
    Log.d(TAG, "Альбомная ориентация, меняем на
               портретную");
    this.setRequestedOrientation (ActivityInfo.
                                SCREEN_ORIENTATION_PORTRAIT);
}
```

Как видно из Листинга 5.2 (выделено жирным), смена ориентации устройства осуществляется с помощью вызова метода класса Активности `android.app.Activity` (<http://developer.android.com/reference/android/app/Activity.html>):

```
public void setRequestedOrientation (int  
                                  requestedOrientation);
```

**Внимание!** Вызов этой функции приведет к уничтожению текущей Активности и созданию новой Активности!

Программно менять ориентацию устройства можно, например, если разрабатывается приложение, которое предназначено для работы только при одной ориентации устройства (например, только для работы в альбомной ориентации). Существует так-же другой (более правильный) способ зафиксировать нужную ориентацию устройства и запретить при этом пересоздание Активности при поворотах устройства. Делается это с помощью файла Манифеста приложения. В элемент `<activity>` нужно добавить два атрибута, как показано в Листинге 5.3.

### Листинг 5.3. Установка фиксированной альбомной ориентации устройства и запрет на пересоздание Активности при поворотах устройства

```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android=  
              "http://schemas.android.com/apk/res/android"  
              package="com.itstep.myapp4" >  
    <application  
        android:allowBackup="true"  
        android:icon="@mipmap/ic_launcher"  
        android:label="@string/app_name"
```

```
    android:theme="@style/AppTheme" >
    <activity
        android:name=".MainActivity"
        android:screenOrientation="landscape"
        android:configChanges="orientation|screenSize"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name=
                "android.intent.action.MAIN" />

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

</manifest>
```

В Листинге 5.3, с помощью атрибута **android:screenOrientation** устанавливается альбомная ориентация (значения для этого атрибута — **landscape** и **portrait** — «альбомная» и «портретная» ориентация соответственно), а с помощью атрибута **android:configChanges** со значениями «**orientation|screenSize**» осуществляется запрет на пересоздание Активности при повороте устройства. Файл Манифеста приложения можно найти в каждом модуле проекта: имя\_модуля/manifests/AndroidManifest.xml.

# 6. Меню приложения

## 6.1. Создание меню

При создании в проекте AndroidStudio каждого нового модуля для приложения автоматически создается Меню приложения. Разработчик приложения может добавлять новые пункты в меню приложения. Сделать это можно двумя способами — как с помощью внесения новых пунктов меню в xml-файл Меню приложения (имя\_модуля/res/menu/main\_menu.xml), так и с помощью программного кода Java. В этом разделе будут рассмотрены оба способа создания новых пунктов в Меню приложения.

Пример создания Меню приложения с помощью xml-файла ресурсов (имя\_модуля/res/menu/main\_menu.xml) представлен в Листинге 6.1. Более подробно изучить документацию по созданию Меню приложения с помощью xml-файла ресурсов можно по ссылке <http://developer.android.com/intl/ru/guide/topics/resources/menu-resource.html>.

### Листинг 6.1. Создание Меню с помощью xml-файла ресурсов

```
<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_winter"  
        android:title="@string/action_winter"
```

```
        android:orderInCategory="100"
        app:showAsAction="never"
    />

<item
    android:id="@+id/action_summer"
    android:title="@string/action_summer"
    android:orderInCategory="120"
    app:showAsAction="never"
/>

<item
    android:id="@+id/action_autumn"
    android:title="@string/action_autumn"
    android:orderInCategory="130"
    app:showAsAction="never"
/>
</menu>
```

Внешний вид меню, формируемый в Листинге 6.1 изображен на Рис. 6.1.



**Рис. 6.1.** Меню, созданное с помощью кода из Листинга 6.1

В файле ресурсов меню могут находиться следующие XML элементы:

- **<menu>** — является контейнером для пунктов меню. Должен быть корневым элементом XML-файла ресурсов. С помощью этого элемента можно создавать вложенные группы меню.
- **<item>** — определяет один пункт меню. Если этот пункт является пунктом «выпадающего» меню (т.е. Его выбор должен приводить к появлению выпадающего подменю), то в нем указывается вложенный элемент **<menu>**.
- **<group>** — необязательный элемент. Позволяет объединять пункты меню в группы для того чтобы было проще управлять несколькими пунктами меню одновременно (например показать/спрятать группу пунктов, разрешить/запретить и т.д.).

Рассмотрим подробнее некоторые атрибуты элемента **<item>** (Листинг 6.2):

#### **Листинг 6.2.** Часто используемые атрибуты элемента **<item>**

```
<item
    android:id="@[+] [package:]id/resource_name"
    android:title="string"
    android:icon=
        "@[package:]drawable/drawable_resource_name"
    android:onClick="method name"
    android:showAsAction=
        ["ifRoom" | "never" | "withText" | "always" |
         "collapseActionView"]
    android:checkable = ["true" | "false"]
    android:visible = ["true" | "false"]
```

```
    android:enabled=["true" | "false"]  
    android:orderInCategory="integer"  
    />
```

Основные атрибуты элемента `<item>`:

- **android:id** — идентификатор пункта меню. Используется в приложении для идентификации выбранного пункта меню при обработке событий выбора пункта меню пользователем.
- **android:title** — строка — надпись для текста меню. Рекомендуется использовать заранее подготовленные строки из файла ресурсов `/res/values/strings.xml`, чтобы при необходимости упростить работу по локализации для приложения.
- **android:icon** — графический элемент из ресурсов, который будет использоваться в качестве значка для пункта меню. Необязательный атрибут.
- **android:onClick** — имя метода класса Активности, который будет вызван при выборе пункта меню пользователем (метод обработчик события выбора пункта меню). Необязательный атрибут. Можно использовать для обработки событий выбора пункта меню метод **onOptionsItemSelected** класса Активности (будет рассказано ниже).
- **app:showAsAction** — указывает, когда и как этот пункт должен отображаться в «Строчке действий» (Что такое «Строка действий» можно узнать с помощью ссылки <http://developer.android.com/intl/ru/training/appbar/index.html>). Если указать значение «**never**» для пункта

меню, то он никогда не будет отображен в «Строчке действий», но будет отображаться среди остальных пунктов меню. Если указать значение «**always**», то пункт меню будет всегда отображен в «Строчке действий», при этом он не будет отображаться среди остальных пунктов меню (для примера, установим значение этого атрибута в «**always**» для пункта «Зима». Что при этом произойдет — изображено на Рис. 6.2).

- **android:checkable** — если установить значение `true`, то пункт меню будет отображаться вместе с CheckBox-ом.
- **android:visible** — задает видимость/невидимость (`true/false`) пункта меню.
- **android:enabled** — `true` если пункт меню доступен, в противном случае пункт меню отображается, но не доступен (нельзя выбрать).
- **android:orderInCategory** — число — порядок отображения пункта меню в списке пунктов меню. Чем выше значение числа, тем ниже в списке отображается пункт меню. Необязательный атрибут.

Рассмотрим элемент **<group>** (Листинг 6.3):

#### **Листинг 6.3.** Элемент **<group>**

```
<group android:id="@+id/идентификатор_группы">
    <item ... />
    <item ... />
</group>
```

С помощью элемента **<group>** создается группа меню. Группа меню это набор пунктов меню с общими характеристиками. Группу пунктов меню можно показывать/

скрывать одновременно с помощью вызова метода класса **android.view.Menu**:

```
public void setGroupVisible (int group,
                           boolean visible);
```

который принимает идентификатор группы (задается атрибутом **android:id**, см. Листинг 6.3.) и значение **true/false** указывающее, что нужно сделать с группой — показать или скрыть.



**Рис. 6.2.** Пример применения значения «always» для атрибута **android:showAsAction** элемента **<item>**. Пункт меню выносится в «Строчку действий»

Пункты группы меню можно включать или отключать одновременно с помощью вызова метода класса **android.view.Menu**:

```
public void setGroupEnabled (int group, boolean enabled);
```

## С помощью метода класса android.view.Menu

```
public void setGroupCheckable (int group,
                           boolean checkable, boolean exclusive);
```

можно указать, являются ли пункты меню группы выби-  
раемыми. Причем параметр **exclusive** задает, является ли  
группа пунктов меню группой «радиокнопок» (true) или  
группой «флажков-чекбоксов» (false).

Для создания группы необходимо вложить элементы  
**<item>** в элемент **<group>** в файле ресурсов меню или  
указать идентификатор группы с помощью метода класса  
**android.view.Menu**

```
public MenuItem add (int groupId, int itemId,
                     int order, CharSequence title);
```

Определять возможность помечать отдельные пункты  
меню можно с помощью атрибута **android:checkable** в эле-  
менте **<item>**, а для всей группы это делается с помощью  
атрибута **android:checkableBehavior** в элементе **<group>**.  
Например, все пункты этой группы меню можно помечать  
с помощью переключателей (Листинг 6.4):

**Листинг 6.4.** Определение возможности помечать отдель-  
ные пункты меню или все пункты меню с помощью атрибута  
**android:checkableBehavior** для элемента **<group>**

```
<menu xmlns:android=
      "http://schemas.android.com/apk/res/android">
<group android:checkableBehavior="single">
    <item
        android:id="@+id/action_winter"
        android:title="@string/action_winter" />
```

```
<item  
    android:id="@+id/action_spring"  
    android:title="@string/action_spring" />  
<item  
    android:id="@+id/action_summer"  
    android:title="@string/action_summer" />  
<item  
    android:id="@+id/action_autumn"  
    android:title="@string/action_autumn" />  
</group>  
</menu>
```

Атрибут **android:checkableBehavior** принимает следующие значения:

- **single** — только один пункт из группы можно пометить (переключатель радиокнопка).
- **all** — все пункты можно пометить (флажки).
- **none** — пометить нельзя ни один пункт

Когда выбирается пункт, который может быть помечен, система вызывает метод обратного вызова (например, **onOptionsItemSelected()**). Именно здесь необходимо задать состояние флажка, поскольку флажок или переключатель не изменяет свое состояние автоматически. Запросить текущее состояние пункта (в котором он находился до того, как был выбран пользователем) можно с помощью **isChecked()**, а затем задать помеченное состояние с помощью **setChecked()**.

Также можно создавать вложенные пункты меню. Для создания вложенных пунктов меню, необходимо вложить элемент **<menu>** в элемент **<item>**, и для вложенного **<menu>** задать элементы **<item>**. Пример создания вложенных пунктов меню приведен в Листинге 6.5.

**Листинг 6.5.** Создание вложенных пунктов меню

```
<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_winter"
        android:title="@string/action_winter"
        android:orderInCategory="100"
        app:showAsAction="never">

        <menu>
            <item
                android:id="@+id/action_winter_december"
                android:title="@string/action_winter_december"
                android:orderInCategory="10"
                app:showAsAction="never"
            />

            <item
                android:id="@+id/action_winter_january"
                android:title="@string/action_winter_january"
                android:orderInCategory="20"
                app:showAsAction="never"
            />

            <item
                android:id="@+id/action_winter_february"
                android:title="@string/action_winter_february"
                android:orderInCategory="30"
                app:showAsAction="never"
            />
        </menu>
    </item>
</menu>
```

```
<item  
    android:id="@+id/action_spring"  
    android:title="@string/action_spring"  
    android:orderInCategory="110"  
    app:showAsAction="never">  
    ...  
</item>  
...  
</menu>
```

В примере Листинга 6.5 для пункта меню «Зима» создается несколько вложенных пунктов меню («Декабрь», «Январь», «Февраль»). Результат работы Меню из Листинга 6.5 в эмуляторе изображен на Рис. 6.3 (вначале выбирается пункт меню «Зима», и для него появляются вложенные пункты меню «Декабрь», «Январь», «Февраль»).



**Рис. 6.3.** Результат работы примера из Листинга 6.5. Вначале выбирается пункт меню «Зима» (слева), и для него появляются вложенные пункты меню «Декабрь», «Январь», «Февраль» (справа)

Пункты меню можно создавать программно. Для этого предназначен метод-обработчик события

```
public boolean onCreateOptionsMenu(Menu menu);
```

класса Активности. Метод принимает в качестве параметра ссылку на объект **android.view.Menu** который и представляет Меню приложения. Для добавления пунктов меню к объекту **android.view.Menu** используется метод класса **android.view.Menu**:

```
public MenuItem add (int groupId, int itemId,  
                     int order, int titleRes);
```

Метод **add** принимает следующие параметры:

- **groupId** — идентификатор группы меню, в которую добавляется пункт меню. Если пункт не принадлежит ни одному пункту меню, то используется значение **Menu.NONE** (имеет значение 0).
- **itemId** — уникальный числовой идентификатор пункта меню. Используется для идентификации пункта меню в обработчиках события выбора пункта меню.
- **order** — порядок расположения пункта меню относительно других пунктов меню (то же самое что и атрибут **android:orderInCategory**)
- **titleRes** — идентификатор строкового ресурса, содержащего название пункта меню.

Пример программного пункта меню представлен в Листинге 6.6.

**Листинг 6.6.** Программное создание пунктов меню

```
public class MainActivity extends ActionBarActivity
{
    /**
     * ID пункта меню "Зима"
     */
    private final static int MENU_ID_WINTER = 1000;

    /**
     * ID пункта меню "Весна"
     */
    private final static int MENU_ID_SPRING = 1001;

    /**
     * ID пункта меню "Лето"
     */
    private final static int MENU_ID_SUMMER = 1002;

    /**
     * ID пункта меню "Осень"
     */
    private final static int MENU_ID_AUTUMN = 1003;

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        //--- Inflate the menu; this adds items
        //--- to the action bar if it is present.-----
        getMenuInflater().inflate(R.menu.menu_main, menu);

        //---Программное создание пунктов меню -----
        menu.clear();

        menu.add(Menu.NONE, MENU_ID_WINTER, 100,
                R.string.action_winter);

        menu.add(Menu.NONE, MENU_ID_SPRING, 110,
                R.string.action_spring);
    }
}
```

```
        menu.add(Menu.NONE, MENU_ID_SUMMER, 120,
                R.string.action_summer);
        menu.add(Menu.NONE, MENU_ID_AUTUMN, 130,
                R.string.action_autumn);
        return true;
    }
    ...
}
```

Если необходимо создать пункт меню содержащий подменю, то необходимо воспользоваться методом объекта **android.View.Menu**:

```
public SubMenu addSubMenu (int groupId, int itemId,
                           int order, int titleRes);
```

Этот метод возвращает ссылку на объект **android.view.SubMenu** (<http://developer.android.com/intl/ru/reference/android/view/SubMenu.html>). Класс **android.view.SubMenu** является классом, производным от класса **android.view.Menu**. Для добавления пунктов подменю к объекту **android.view.SubMenu** нужно использовать метод:

```
public MenuItem add (int groupId, int itemId,
                     int order, int titleRes);
```

Пример создания пунктов меню с пунктами подменю приведен в Листинге 6.7.

#### **Листинг 6.7.** Пример создания пунктов меню с пунктами подменю

```
public class MainActivity extends ActionBarActivity
{
    ...
    @Override
```

```
public boolean onCreateOptionsMenu(Menu menu)
{
    //---Inflate the menu; this adds items to the action
    //---bar if it is present.-----
    getMenuInflater().inflate(R.menu.menu_main, menu);

    //---Программное создание пунктов меню-----
    menu.clear();

    //---Пункт меню "Зима" с пунктами подменю-----
    android.view.SubMenu mWinter =
        menu.addSubMenu(Menu.NONE, MENU_ID_WINTER,
                      100, R.string.action_winter);

    mWinter.add(Menu.NONE, MENU_ID_WINTER_DECEMBER,
                10, R.string.action_winter_december);

    mWinter.add(Menu.NONE, MENU_ID_WINTER_JANUARY,
                20, R.string.action_winter_january);

    mWinter.add(Menu.NONE, MENU_ID_WINTER_FEBRUARY,
                30, R.string.action_winter_february);

    //---Пункт меню "Весна" с пунктами подменю-----
    android.view.SubMenu mSpring =
        menu.addSubMenu(Menu.NONE, MENU_ID_SPRING,
                      110, R.string.action_spring);

    mSpring.add(Menu.NONE, Menu_ID_SPRING_MARCH,
                10, R.string.action_spring_march);

    mSpring.add(Menu.NONE, MENU_ID_SPRING_APRIl,
                20, R.string.action_spring_april);

    mSpring.add(Menu.NONE, MENU_ID_SPRING_MAY,
                30, R.string.action_spring_may);
```

```

        menu.add(Menu.NONE, MENU_ID_SUMMER, 120,
                R.string.action_summer);
        menu.add(Menu.NONE, MENU_ID_AUTUMN, 130,
                R.string.action_autumn);

        return true;
    }
}

```

Результат работы примера из Листинга 6.7 изображен на Рис. 6.3. Исходные коды примеров данного раздела можно найти в модуле «app5» файлов с исходными кодами, которые прилагаются к текущему уроку.

## 6.2. Обработка события выбора пункта меню

Как уже рассказывалось в предыдущем разделе, каждому пункту меню можно назначать метод обратного вызова, который будет вызываться при выборе пункта меню (метод — обработчик события выбора меню). Метод обратного вызова назначается с помощью атрибута **android:onClick** элемента **<item>** (см. Листинг 6.2). Пример применения атрибута **android:onClick** для обработки события выбора пункта меню приведен в Листингах 6.8 и 6.9.

**Листинг 6.8.** Применение атрибута **android:onClick** для назначения метода обработки события выбора пункта меню

```

<item
    android:id="@+id/action_winter_december"
    android:title="@string/action_winter_december"
    android:orderInCategory="10"
    app:showAsAction="never"
    android:onClick="menuItemClick"
/>

```

**Листинг 6.9.** Метод обратного вызова, назначаемый через атрибут android:onClick в Листинге 6.8

```
public void meniItemClick(MenuItem item)
{
    Toast.makeText(this, item.getTitle().toString(),
        Toast.LENGTH_SHORT).show();
}
```

Как видно из Листинга 6.9, метод обратного вызова принимает в качестве параметра ссылку на объект `android.view.MenuItem` (<http://developer.android.com/intl/ru/reference/android/view/MenuItem.html>), который представляет выбранный пункт меню.

Можно не использовать атрибут `android:onClick` для назначения метода обратного вызова. В этом случае, при выборе пункта меню будет вызываться метод

```
public boolean onOptionsItemSelected(MenuItem item);
```

класса Активности. Этот метод является методом по умолчанию для всех пунктов Меню приложения. Метод `onOptionsItemSelected` в качестве своего параметра принимает ссылку на объект экземпляр класса `android.view.MenuItem` выбранного пункта меню. В Листинге 6.10 приведен пример обработки событий выбора пунктов меню, созданных с помощью файла ресурсов, представленного в Листинге 6.5.

**Листинг 6.10.** Обработка событий выбора пунктов меню в методе Активности `onOptionsItemSelected`

```
public class MainActivity extends Activity
{
    ...
}
```

```
@Override
public boolean onOptionsItemSelected(MenuItem item)
{
    //--Получаем идентификатор выбранного пункта меню--
    int id = item.getItemId();

    //--Строка сообщения для вывода в тосте-----
    String msg = "";

    //--Идентифицируем выбранный пункт меню
    //--и назначаем сообщение-----
    switch (id)
    {
        case R.id.action_winter :
            msg = "Зима - самое холодное время года";
            break;

        case R.id.action_spring :
            msg = "Весна - самое оптимистичное
                  время года";
            break;

        case R.id.action_summer :
            msg = "Лето - самое радостное время года";
            break;

        case R.id.action_autumn :
            msg = "Осень - самое грустное время года";
            break;

        case R.id.action_winter_december :
            msg = "Декабрь - первый зимний месяц";
            break;

        case R.id.action_winter_january :
            msg = "Январь - второй зимний месяц";
            break;
    }
}
```

```

        case R.id.action_winter_february :
            msg    = "Февраль - последний зимний месяц";
            break;

        default:
            return super.onOptionsItemSelected(item);
    }

//--Отображаем тост с информацией-----
    Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();

    return true;
}
}

```

В Листинге 6.11 приведен пример обработки события выбора пунктов меню, созданных программно из Листинга 6.6.

### **Листинг 6.11. Обработка событий выбора пунктов меню, созданных программно**

```

@Override
public boolean onOptionsItemSelected(MenuItem item)
{
//--Получаем идентификатор выбранного пункта меню--
    int id      = item.getItemId();

//--Строка сообщения для вывода в тосте-----
    String msg      = "";

//--Идентифицируем выбранный пункт меню
//--и назначаем сообщение-----
    switch (id)

    {
        case MENU_ID_WINTER :
            msg    = "Зима - самое холодное время года";
            break;
    }
}

```

```
case MENU_ID_SPRING :  
    msg      = "Весна - самое оптимистичное время года";  
    break;  
  
case MENU_ID_SUMMER :  
    msg      = "Лето - самое радостное время года";  
    break;  
  
case MENU_ID_AUTUMN :  
    msg      = "Осень - самое грустное время года";  
    break;  
  
default:  
    return super.onOptionsItemSelected(item);  
}  
  
//--Отображаем тост с информацией-----  
Toast.makeText(this, msg, Toast.LENGTH_SHORT).show();  
  
return true;  
}
```

Если сравнить Листинги 6.10 и 6.11, то можно увидеть, что нет никакой разницы в обработке события выбора пунктов меню, созданных с помощью файла ресурсов и созданных программно. Различие лишь в том, что для пунктов меню из ресурсов используется идентификатор из ресурсов, а для пунктов меню, созданных программно, необходимо самостоятельно задавать в виде константных значений идентификаторы пунктов меню.

Исходные коды примеров данного раздела можно найти в модуле «app5» файлов с исходными кодами, которые прилагаются к текущему уроку.

### 6.3. Контекстное меню

Контекстное меню вызывается длительным нажатием на каком-либо виджете. Для этого необходимо вначале зарегистрировать виджет на получение события формирования контекстного меню с помощью вызова метода Активности:

```
public void registerForContextMenu (View view);
```

Метод принимает ссылку на виджет, для которого необходимо зарегистрировать получение события формирования контекстного меню. Этот метод можно вызвать, например, в методе **onCreate** класса Активности.

Следующий шаг, который необходимо сделать, это создать в классе Активности метод **onCreateContextMenu**, который будет автоматически вызываться для формирования контекстного меню для ранее зарегистрированного виджета. Метод выглядит следующим образом:

```
public void onCreateContextMenu(ContextMenu menu,  
    View view, ContextMenu.ContextMenuItemInfo menuInfo);
```

Метод **onCreateContextMenu** принимает следующие параметры:

- **ContextMenu menu** — объект контекстного меню (<http://developer.android.com/intl/ru/reference/android/view/ContextMenu.html>), к которому необходимо присоединить дочерние пункты контекстного меню.
- **View view** — виджет, для которого нужно сформировать/показать контекстное меню.

- **ContextMenu.ContextMenuInfo menuInfo** — дополнительная информация для контекстного меню, который должен быть создан. Зависит от типа виджета **view**.

Обрабатывать события выбора пунктов контекстного меню необходимо в методе **onContextItemSelected**, который необходимо создать в классе Активности. Метод выглядит следующим образом:

```
public boolean onContextItemSelected(MenuItem item);
```

Параметр **MenuItem menu** это ссылка на пункт контекстного меню, который выбрал пользователь. Принцип обработки события выбора контекстного пункта меню ничем не отличается от обработки события выбора обычного пункта меню.

В Листингах 6.12 и 6.13 приведен пример создания контекстного меню для разных виджетов и обработка события выбора пунктов контекстного меню. Листинг 6.12 содержит xml верстку макета Активности, а Листинг 6.13 содержит непосредственно программный код создания контекстного меню и обработки событий выбора контекстного меню. Весь исходный код примера из Листинга в 6.12 и 6.13 находится в модуле «app5» файлов с исходными кодами примеров, прилагаемыми к этому уроку.

**Листинг 6.12.** Макет Активности с виджетами, для которых будут формироваться контекстные меню в Листинге 6.13

```
<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=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tv1"
        android:text="Hello World"
        android:textSize="16pt"
        android:gravity="center_horizontal"
        android:textColor="#006699"
    />

    <Space
        android:layout_width="match_parent"
        android:layout_height="16dp"
    />

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tv2"
        android:text="Android Forever"
        android:textSize="16pt"
        android:gravity="center_horizontal"
        android:textColor="#009966"
    />

</LinearLayout>
```

**Листинг 6.13.** Регистрация виджетов на получение события создания контекстного меню, создание контекстного меню, обработка события выбора контекстного меню

```
public class MainActivity extends ActionBarActivity
{
    //--Class constants-----
    /**
     * ID первого пункта контекстного меню
     * для TextView tv1
     */
    private final static int CONTEXT_MENU_WORLD = 2000;

    /**
     * ID второго пункта контекстного меню
     * для TextView tv1
     */
    private final static int CONTEXT_MENU_ANDROID =
        2001;
    /**
     * ID третьего пункта контекстного меню
     * для TextView tv1
     */
    private final static int CONTEXT_MENU_GOOGLE =
        2002;
    /**
     * ID первого пункта контекстного меню
     * для TextView tv2
     */
    private final static int CONTEXT_MENU_FOREVER =
        2003;
    /**
     * ID второго пункта контекстного меню для TextView tv2
     */
    private final static int CONTEXT_MENU_RULES =
        2004;
```

```
/***
 * ID третьего пункта контекстного меню для TextView tv2
 */
private final static int CONTEXT_MENU_COOL = 2005;

//--Class members-----
/***
 * Виджет, для которого в этом примере формируется
 * контекстное меню
 */
private TextView tv1;

/***
 * Виджет, для которого в этом примере формируется
 * контекстное меню.
 */
private TextView tv2;

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

//--Получение ссылок на интересующие виджеты-----
    this.tv1 = (TextView) this.findViewById(R.id.tv1);
    this.tv2 = (TextView) this.findViewById(R.id.tv2);

//--Регистрация для получения события отображения
//--контекстного меню-----
    this.registerForContextMenu(this.tv1);
    this.registerForContextMenu(this.tv2);
}

/***
 * Обработчик события создания контекстного меню
```

```
* @param menu - Контекстное меню, созданное
* системой, для прикрепления дочерних пунктов меню.
* @param view - Виджет, для которого необходимо
* сформировать контекстное меню
* @param menuInfo
*/
@Override
public void onCreateContextMenu(
    ContextMenu menu,
    View view,
    ContextMenu.ContextMenuItemInfo menuInfo)
{
    switch (view.getId())
    {
//--Создание контекстного меню для TextView tv1---
        case R.id.tv1 :
            menu.add(Menu.NONE, CONTEXT_MENU_WORLD, 10,
                    "World");
            menu.add(Menu.NONE, CONTEXT_MENU_ANDROID, 20,
                    "Android");
            menu.add(Menu.NONE, CONTEXT_MENU_GOOGLE, 30,
                    "Google");
            break;

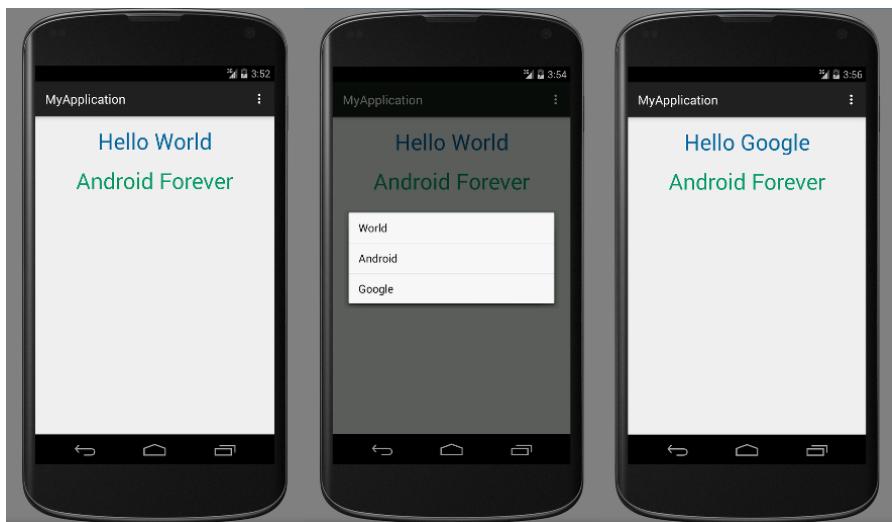
//--Создание контекстного меню для TextView tv2---
        case R.id.tv2 :
            menu.add(Menu.NONE, CONTEXT_MENU_FOREVER,
                    10, "Forever");
            menu.add(Menu.NONE, CONTEXT_MENU_RULES,
                    20, "Rules");
            menu.add(Menu.NONE, CONTEXT_MENU_COOL,
                    30, "Cool");

            break;
    }
}
```

```
/**  
 * Обработчик события выбора пункта контекстного меню.  
 * @param item - Пункт контекстного меню,  
 * который выбран пользователем.  
 * @return - true - событие выбора пункта меню  
 * обработано.  
 */  
@Override  
public boolean onContextItemSelected(MenuItem item)  
{  
    switch (item.getItemId())  
    {  
        case MainActivity.CONTEXT_MENU_WORLD :  
            this.tv1.setText("Hello World");  
            break;  
  
        case MainActivity.CONTEXT_MENU_ANDROID :  
            this.tv1.setText("Hello Android");  
            break;  
  
        case MainActivity.CONTEXT_MENU_GOOGLE :  
            this.tv1.setText("Hello Google");  
            break;  
  
        case MainActivity.CONTEXT_MENU_FOREVER :  
            this.tv2.setText("Android Forever");  
            break;  
  
        case MainActivity.CONTEXT_MENU_RULES :  
            this.tv2.setText("Android Rules!");  
            break;  
  
        case MainActivity.CONTEXT_MENU_COOL :  
            this.tv2.setText("Android Cool!!!!");  
            break;  
  
        default:  
            return super.onContextItemSelected(item);  
    }  
}
```

```
        return      true;
    }
    ...
}
```

Демонстрация работы примера из Листингов 6.12 и 6.13 представлена на Рис. 6.4. Пользователь, с помощью контекстного меню меняет надписи в виджетах **TextView**. Для тестирования примера в эмуляторе — нажмите мышкой на одном из виджетов **android.widget.TextView** и удерживайте нажатие в течении нескольких секунд, до тех пор пока не появится контекстное меню.



**Рис. 6.4.** Демонстрация примера из Листингов 6.12 и 6.13. Формирование и отображение контекстного меню для виджета с текстом «Hello World». После выбора пункта контекстного меню «Google» в виджете меняется текст на «Hello Google»

# 7. Менеджер раскладки ScrollView. Создание прокручиваемых элементов

Менеджер раскладки **android.widgetScrollView** позволяет пользователю прокручивать дочерние виджеты в случае, если все виджеты, помещенные в **android.widgetScrollView** превышают физические размеры экрана устройства. Иерархия классов для класса **android.widgetScrollView**:

```
java.lang.Object
  |
  +-- android.view.View
    |
    +-- android.view.ViewGroup
      |
      +-- android.widget.FrameLayout
        |
        +-- android.widgetScrollView
```

Как видно из иерархии классов, класс **android.widgetScrollView** является производным от класса **android.widgetFrameLayout**, следовательно, для объекта **android.widgetScrollView** может быть только один дочерний виджет. Чаще всего таким дочерним является другой менеджер раскладки (контейнер). Справочная информация API по классу **android.widgetScrollView** находится по адресу: <http://developer.android.com/intl/ru/reference/android/widget/ScrollView.html>.

Объекты класса `android.widgetScrollView` поддерживают только вертикальную прокрутку дочерних виджетов. Для горизонтальной прокрутки предназначен класс `android.widget.HorizontalScrollView` (<http://developer.android.com/reference/android/widget/HorizontalScrollView.html>). Принцип работы с `android.widget.HorizontalScrollView` аналогичен работе с `android.widgetScrollView` и поэтому в рамках этого урока рассматриваться не будет.

Давайте рассмотрим пример использования `android.widgetScrollView`. Чтобы не осуществлять верстку дочерних виджетов вручную (ведь чтобы увидеть прокрутку необходимо сверстать много виджетов), виджеты для данного примера будут создаваться программно. Макет Активности будет содержать все необходимые родительские контейнеры, к которым в примере будут добавляться дочерние виджеты. Макет Активности для примера представлен в Листинге 7.1.

**Листинг 7.1.** Макет Активности для примера контейнера `android.widgetScrollView`

```
<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=".MainActivity">  
  
    <TextView  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"
```

```
    android:text="ScrollView Example"
    android:gravity="center_horizontal"
    android:textSize="14pt"
    android:textColor="#006699"/>

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#E0E0E0"
    android:id="@+id/sv1">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:id="@+id/l11">

        </LinearLayout>

    </ScrollView>

</LinearLayout>
```

Макет Активности из Листинга 7.1 в качестве главного контейнера использует менеджер раскладки **android.widget.LinearLayout** с вертикальной ориентацией, который содержит два дочерних виджета: **android.widget.TextView**, который используется чтобы отобразить заголовок примера и рассматриваемый нами контейнер **android.widgetScrollView**. В качестве дочернего виджета для **android.widgetScrollView** используется еще один контейнер **android.widget.LinearLayout** (с идентификатором «@+id/l11»), в который и будет осуществляться добавление виджетов для демонстрации прокрутки.

В Листинге 7.2 указан программный код, создающий множество дочерних виджетов для демонстрации прокрутки. Виджеты создаются в обработчике **onCreate** Активности. Внешний вид работы примера из Листингов 7.1 и 7.2 изображен на Рис. 7.1.



**Рис. 7.1.** Внешний вид примера из Листингов 7.1 и 7.2

**Листинг 7.2.** Добавление множества виджетов  
в контейнер для демонстрации прокрутки в контейнере  
`android.widget ScrollView`

```
public class MainActivity extends ActionBarActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```

//-- Получение ссылки на контейнер, куда будем
//-- добавлять дочерние виджеты -----
    LinearLayout      LL1 = (LinearLayout) this.
                                findViewById(R.id.ll1);

//--Заполнение дочерними виджетами -----
    for (int i = 0; i < 50; i++)
    {
        TextView TV = new TextView(this);
        TV.setText("Hello World " + i);
        TV.setTextSize(TypedValue.COMPLEX_UNIT_PT, 10);
        TV.setTextColor(Color.rgb(128, 64 + i * 2, 128));
        TV.setGravity(Gravity.CENTER_HORIZONTAL);
        LL1.addView(TV);
    }
}
...
}

```

У `android.widget.ScrollView` также имеется возможность осуществлять прокрутку программными средствами. Для демонстрации этих возможностей, модифицируем текущий пример, добавив в него кнопки (виджеты `android.widget.Button`) с помощью которых будем делать программный скроллинг (см. Листинг 7.3).

### **Листинг 7.3.** Добавление кнопок для демонстрации программного управления скроллингом

```

<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=".MainActivity">

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="ScrollView Example"
    android:gravity="center_horizontal"
    android:textSize="14pt"
    android:textColor="#006699"/>

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="PgUp"
        android:id="@+id/btnPgUp"
        android:onClick="btnArrowClick" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="PgDn"
        android:id="@+id/btnPgDn"
        android:onClick="btnArrowClick" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Up"
        android:id="@+id/btnUp"
        android:onClick="btnArrowClick" />

    <Button
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="Dn"
        android:id="@+id/btnDn"
        android:onClick="btnArrowClick" />
</LinearLayout>

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#E0E0E0"
    android:id="@+id/sv1"
    >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:id="@+id/l11"
        >

        </LinearLayout>
    </ScrollView>
</LinearLayout>

```

Для программного скроллинга в классе **android.widget ScrollView** предназначены следующие методы:

- Скроллинг относительно текущей координаты:

```
public void scrollBy (int x, int y);
```

Метод принимает значения — на сколько пикселей нужно сместиться относительно текущей позиции скроллинга. Положительное значение по оси Y означает смещение вниз, отрицательное — вверх.

- Скроллинг в указанную координату:

```
public void scrollTo (int x, int y);
```

- «Плавный» скроллинг относительно текущей координаты:

```
public final void smoothScrollBy (int dx, int dy);
```

- «Плавный» скроллинг в указанную координату:

```
public final void smoothScrollTo (int x, int y);
```

- Скроллинг «Страница вверх» / «Страница вниз» («Page Up» / «Page Down»):

```
public boolean pageScroll (int direction);
```

**Примечание:** «Плавный» не зря помещено в кавычки — чтобы увидеть плавность в эмуляторе необходимо внимательно присмотреться.

Итак, вернемся к нашему примеру. Как видно из Листинга 7.3, кнопкам управления скроллингом назначен обработчик события клика **btnArrowClick**. Этот метод находится в классе Активности нашего примера и представлен в Листинге 7.4.

**Листинг 7.4.** Программное управление прокруткой в виджете `android.widget ScrollView`

```
public void      btnArrowClick(View view)
{
    //-- Получение размера высоты отдельного дочернего
    //-- виджета -----
    LinearLayout LL1 = (LinearLayout) this.
        findViewById(R.id.ll1);
    int scrollSize = (LL1.getChildCount() > 0)?LL1.
        getChildAt(0).getHeight():10;
```

```
//--- Получение ссылки на объект android.widget.  
//--- ScrollView -----  
    ScrollView SV = (ScrollView) this.  
                findViewById(R.id.sv1);  
  
//--- Обработка событий кнопок "Up", "Down", "PgUp",  
//--- "PgDn" -----  
    switch (view.getId())  
    {  
//---Прокрутка вниз на высоту дочернего виджета---  
    case R.id.btnDn :  
        // SV.scrollBy(0, scrollSize);  
        SV.smoothScrollBy(0, scrollSize);  
        break;  
  
//--- Прокрутка вниз на высоту виджета  
//--- android.widget.ScrollView -----  
    case R.id.btnPgDn :  
        SV.pageScroll(ScrollView.FOCUS_DOWN);  
        break;  
  
//--- Прокрутка вверх на высоту дочернего виджета -  
    case R.id.btnUp :  
        SV.scrollBy(0, -scrollSize);  
        break;  
// -- Прокрутка вверх на высоту виджета  
// -- android.widget.ScrollView -----  
    case R.id.btnPgUp :  
        SV.pageScroll(ScrollView.FOCUS_UP);  
        break;  
    }  
}
```

Когда пользователь кликнет на кнопку «Dn» (Вниз), то пример осуществит программную прокрутку содержимого виджета **android.widget.ScrollView** вниз на количество

пикселей, равное размеру высоты дочернего виджета (в примере все дочерние виджеты имеют одинаковую высоту). Для получения высоты дочернего виджета (которым является виджет `android.widget.TextView`) предназначен код, инициализирующий локальную переменную `scrollSize`, которой присваивается значение высоты первого дочернего виджета. При постраничной прокрутке с помощью метода `pageScroll` высчитывать высоту страницы нет необходимости — метод сделает это автоматически.

Внешний вид работы примера из Листингов 7.3 и 7.4 изображен на Рис. 7.2. Исходный код примера можно найти в модуле «аррб» файлов исходных кодов, прилагаемых к данному уроку.



**Рис. 7.2.** Внешний вид работы примера из Листингов 7.3 и 7.4.

Последовательность слева направо: первоначальный вид приложения, прокрутка при нажатии на кнопку «Dn» («Вниз»), прокрутка при нажатии на кнопку «PgDn» («Страница вниз»)

# 8. Создание виджетов с помощью LayoutInflater

До сих пор мы создавали виджеты программно с помощью оператора new для создания объектов классов. Это хороший способ программного создания виджетов. Но такой способ становится громоздким, если необходимо создавать сложный набор виджетов. Для упрощения такой задачи существует так же и другой способ создания виджетов. Этот способ заключается в использовании xml-макета для программного (динамического) создания виджетов, который (xml-макет) содержит верстку создаваемой структуры виджетов. Данный способ актуален, например, при создании сложных элементов всевозможных списков.

Для программного создания виджетов с помощью xml-макета предназначен объект `android.view.LayoutInflater`. Иерархия классов для `android.view.LayoutInflater` очень простая:

```
java.lang.Object  
|  
+-- android.view.LayoutInflater
```

Объект класса `android.view.LayoutInflater` (<http://developer.android.com/intl/ru/reference/android/view/LayoutInflator.html>) никогда не нужно создавать обычным способом. Этот объект создается системой и наше приложение может получить на него ссылку при необходимости.

Для получения ссылки на объект **android.view.LayoutInflater** используется метод **Активности** (Активность наследует этот метод из класса **android.content.Context**):

```
public LayoutInflator getLayoutInflater();
```

Также для получения ссылки на объект **android.view.LayoutInflater** используется метод класса **android.content.Context**:

```
public final T getSystemService (Class<T> serviceClass);
```

который возвращает ссылку на объект системного уровня по имени класса. Для получения ссылки на объект **android.view.LayoutInflater** нужно передать в метод **getSystemService** значение **Context.LAYOUT\_INFLATER\_SERVICE**:

```
LayoutInflater inflater =  
    (LayoutInflater)context.getSystemService  
    (Context.LAYOUT_INFLATER_SERVICE);
```

Конечно же, использование метода **getLayoutInflater** в методах класса **Активности** кажется более удобным, поэтому в примерах этого раздела будет использоваться этот метод.

Получив ссылку на объект **android.view.LayoutInflater**, можно создавать наборы виджетов с помощью вызова метода **inflate**:

```
public View inflate (int resource, ViewGroup root);
```

Метод создает новый набор (иерархию) виджетов (**android.widget.View**) на основе идентификатора

xml-ресурса, содержащего макет. Идентификатор xml-ресурса передается в метод с помощью параметра **resource**. Второй параметр **ViewGroup root** — ссылка на виджет, который будет родительским виджетом для создаваемого набора виджетов. Метод **inflate** возвращает ссылку на корневой элемент набора (иерархии) созданных виджетов. Созданный набор виджетов прикрепляется к родительскому виджету **root**.

Существует еще несколько методов **inflate**. Познакомиться с ними можно в описании класса **android.view.LayoutInflater** по ссылке <http://developer.android.com/intl/ru/reference/android/view/LayoutInflator.html>. Мы же рассмотрим еще один метод **inflate**:

```
public View inflate (int resource, ViewGroup root,  
                     boolean attachToRoot);
```

Этот метод принимает те же параметры, что и предыдущий метод, но позволяет с помощью последнего параметра **boolean attachToRoot** указывать, нужно ли прикреплять созданную иерархию виджетов к родителю (**true**) или нет (**false**).

Итак, давайте на примере научимся использовать объект **android.viewLayoutInflater**. Для этого в файлах исходных кодов, прилагаемых к этому уроку, создан модуль «app7».

Внешний вид макета активности представлен в Листинге 8.1. Макет имеет достаточно простой вид. В контейнер **android.widget.LinearLayout** с идентификатором «@+id/mainLL» будут добавляться создаваемые с помощью объекта **android.view.LayoutInflater** наборы виджетов.

**Листинг 8.1.** Макет Активности для примера  
использования объекта android.view.LayoutInflater

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

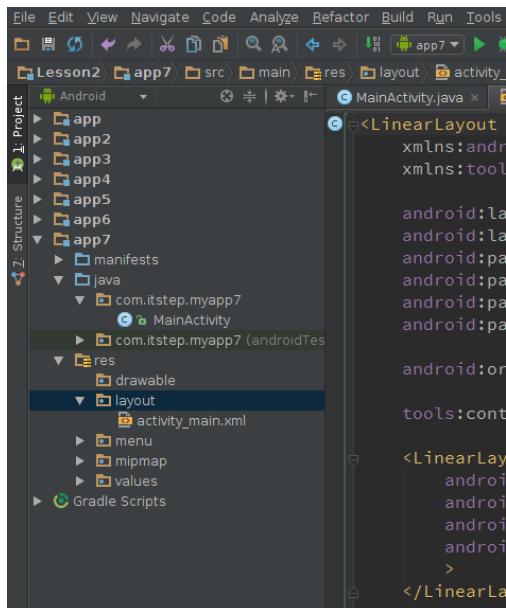
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/mainLL"
        android:orientation="vertical">
    </LinearLayout>

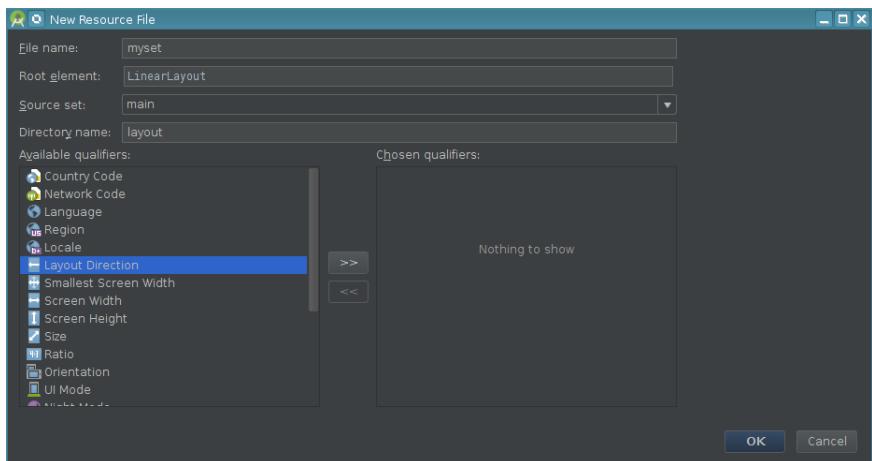
</LinearLayout>
```

Теперь необходимо создать файл ресурсов, содержащий xml-разметку для макета, на основе которого будут в примере создаваться программно наборы виджетов. Для этого кликните правой кнопкой мыши по папке res/layout вашего модуля (см. Рис. 8.1) и в появившемся контекстном меню выберите пункт «New». Появится еще одно контекстное меню, в котором необходимо выбрать пункт «Layout resource file». Появится диалоговое окно (см. Рис. 8.2), в котором необходимо указать имя файла ресурсов (оно же будет идентификатором ресурсов), корневой менеджер раскладки (опция Root Element) и т.д.

## 8. Создание виджетов с помощью LayoutInflater



**Рис. 8.1.** Папка res/layout модуля, в которую добавляется файл ресурсов с xml-макетом



**Рис. 8.2.** Окно конфигурации создаваемого файла ресурсов xml-макета

Заполните поля так же, как указано на Рис. 8.2 и нажмите кнопку «OK». После этого Android Studio добавит в наш модуль файл ресурсов с именем **myset.xml**. В этом файле нужно выполнить верстку набора виджетов, которые будут создаваться с помощью объекта **android.view.LayoutInflater**. Верстка для нашего примера представлена в Листинге 8.2.

**Листинг 8.2.** xml-макет для набора виджетов, которые будут создаваться с помощью объекта **android.view.LayoutInflater** в Листинге 8.3

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

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/itemTxt"
        />

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/itemCbx"
        />
</LinearLayout>
```

Обратите внимание, что в Листинге 8.2 для виджетов макета указаны идентификаторы. Эти идентификаторы будут использованы для получения доступа к этим виджетам в созданном наборе, например для заполнения этих виджетов данными, назначения дополнительных стилей и т. д.

Итак, приступим непосредственно к созданию виджетов на основе созданного макета. Как это делается отображено в Листинге 8.3. Листинг оснащен подробным комментарием.

**Листинг 8.3.** Использование объекта  
android.view.LayoutInflater для создания набора виджетов  
на основе xml-макета из файлов ресурсов

```
public class MainActivity extends ActionBarActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //-- Находим контейнер LinearLayout -----
        LinearLayout mainLL = (LinearLayout) this.
                                         findViewById(R.id.mainLL);

        //-- Получение ссылки на объект
        //-- android.view.LayoutInflater -----
        LayoutInflator ltInflater = getLayoutInflater();

        //-- Создаем набор виджетов из макета в файле
        //-- ресурсов -----
        View view = ltInflater.inflate(R.layout.myset,
                                       mainLL, false);
```

```
//--Заполним содержимое созданных виджетов-----
Button      B = (Button)
            view.findViewById(R.id.itemBtn);
B.setText("Кнопка");

TextView    TV = (TextView)
            view.findViewById(R.id.itemTxt);
TV.setText("Текстовое поле");

CheckBox    CB = (CheckBox)
            view.findViewById(R.id.itemCbx);
CB.setText("Флажок");

//--Добавляем созданный виджет в главный
//--LinearLayout-----
        mainLL.addView(view);
    }
    ...
}
```

Обратите внимание, что в примере при вызове метода **inflate** в качестве параметра, указывающего необходимость автоматически прикреплять созданный набор виджетов к родительскому контейнеру, передается значение **false**. Это делается потому, что позже происходит вызов метода **mainLL.addView(view)**, с помощью которого и происходит добавление созданного набора вижетов к родительскому контейнеру. Если в метод **inflate** передать третьим параметром значение **true**, то необходимости в вызове **mainLL.addView(view)** не было бы. Можете самостоятельно это проверить.

Также еще обратите внимание на то, как происходит заполнение некоторых виджетов (Кнопка, Текстовое поле, Флажок) из созданного набора. Ссылки на эти виджеты

получают с помощью хорошо знакомого вам метода **findViewById**. Однако, впервые этот метод вызывается не для текущего объекта Активности, а для корневого виджета созданного с помощью объекта **android.view.LayoutInflater** набора виджетов. Возьмите на заметку такую возможность импльзования метода **findViewById** (в Листинге 8.3 выделено жирным).

Внешний вид работы примера из Листингов 8.1, 8.2, 8.3 изображен на Рис. 8.3.



**Рис. 8.3.** Внешний вид работы примера из Листингов 8.1, 8.2, 8.3

Также необходимо добавить, что на основе одного xml-макета из файлов ресурсов можно создавать любое количество наборов виджетов. Например, для формирования списков. Об этом будет рассказано в следующем разделе.

## 9. Пример формирования списка с помощью LayoutInflater и размещение его в ScrollView

В этом разделе мы рассмотрим создание не одного набора виджетов с помощью `android.view.LayoutInflater`, а нескольких наборов. Каждый набор будет представлять элемент списка. Конечно же, это будет некоторый условный список. Ведь для организации списков в Android есть специальные классы списков (например `android.widget.ListView`), которые будут рассматриваться в наших уроках позже. Тем не менее, пример данного раздела интересен с точки зрения применения на практике полученных знаний.

Весь исходный код примера данного раздела находится в модуле «app8» файлов с исходными кодами, прилагаемых к этому уроку.

Пример будет демонстрировать работу приложения со списком продуктов. Список продуктов будет располагаться в виджете `android.widget ScrollView` (вернее в его дочернем виджете `android.widget LinearLayout`, который и будет прокручиваться виджетом `android.widget ScrollView`). Каждый продукт списка будет представлен экземпляром класса `Product` (см. Листинг 9.1), в котором находятся поля «Название продукта», «Стоимость продукта» (в условных единицах), «Вес продукта» (в граммах). Для отображения каждого продукта в списке, будет создаваться набор виджетов на основе xml-макета из

ресурсов с помощью объекта `android.view.LayoutInflater`. Макет из файла ресурсов (`/res/layout/productitem.xml`) представлен в Листинге 9.3.

Список продуктов в примере является модифицируемым, то есть пользователь имеет возможность добавлять, изменять и удалять продукты по своему желанию. Для этого в макете Активности предназначены кнопки «Добавить», «Обновить», «Удалить» (см. Рис. 9.1). Макет Активности данного примера приведен в Листинге 9.2.

**Листинг 9.1.** Класс `Product`, объекты которого будут использоваться в качестве элементов списка рассматриваемого примера

```
/**
 * Class Product - Продукт (он же Товар).
 */
class Product
{
    //---Class members-----
    /**
     * Название продукта
     */
    public String name;

    /**
     * Стоимость продукта
     */
    public double price;

    /**
     * Вес продукта
     */
    public int weight;
```

```
//--Class methods-----  
public Product(String name, double price, int weight)  
{  
    this.name = name;  
    this.price = price;  
    this.weight= weight;  
}  
}
```



**Рис. 9.1.** Внешний вид макета Активности из Листинга 9.2

**Листинг 9.2.** Верстка макета Активности рассматриваемого в этом разделе примера

```
<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"
```

## 9. Пример формирования списка с помощью LayoutInflater...

```
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"

    android:orientation="vertical"
    android:background="#60FFFF"

    tools:context=".MainActivity">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Список товаров"
        android:textSize="16pt"
        android:gravity="center_horizontal"
        android:textColor="#006699" />

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

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#274E77"
            android:textColor="#FFFFFF"
            android:textSize="7pt"
            android:padding="8dp"

            android:layout_margin="8dp"
```

```
        android:onClick="btnActionClick"
        android:id="@+id/btnDel"
        android:text="Удалить" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#274E77"
        android:textColor="#FFFFFF"
        android:textSize="7pt"
        android:padding="8dp"
        android:layout_margin="8dp"
        android:onClick="btnActionClick"
        android:id="@+id/btnUpd"
        android:text="Обновить" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#274E77"
        android:textColor="#FFFFFF"
        android:textSize="7pt"
        android:padding="8dp"
        android:layout_margin="8dp"
        android:onClick="btnActionClick"
        android:id="@+id/btnAdd"
        android:text="Добавить" />
    </LinearLayout>

<ScrollView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/svOne">

    <LinearLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:id="@+id/mainLL">

</LinearLayout>
</ScrollView>
</LinearLayout>
```

В макете Активности из Листинга 9.2 корневым контейнером является менеджер раскладки **android.widget.LinearLayout** с вертикальной ориентацией. В него вложены следующие дочерние виджеты:

- Текстовое поле (**android.widget.TextView**) содержащее заголовок для приложения «Список товаров» (см. Рис. 9.1);
- Контейнер **android.widget.LinearLayout** с горизонтальной ориентацией, в который помещены три кнопки «Удалить», «Обновить», «Добавить» (см. Рис. 9.1). Кнопкам назначен обработчик события нажати — метод **btnActionClick** который будет рассмотрен ниже;
- Контейнер **android.widget.ScrollView** (на Рис. 9.1 изображен в виде прямоугольной области более темного цвета), в который добавлен контейнер **android.widget.LinearLayout** с вертикальной ориентацией и идентификатором «**@+id/mainLL**» в который и будут добавляться наборы виджетов, представляющие каждый объект продукт (**Product**).

Верстка макета (файл ресурсов `/res/layout/productitem.xml`) для набора виджетов для формирование элементов списка (Листинг 9.3).

**Листинг 9.3.** Файл ресурсов /res/layout/productitem.xml, содержащий xml-макет на основе которого будут создаваться наборы виджетов для каждого продукта, представленного в списке

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="50dp">

    <CheckBox
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:id="@+id/cb1"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:id="@+id/tvName"
        android:textColor="#4A708B"
        android:textStyle="bold"
        android:textSize="10pt"
        android:layout_weight="1"
        />

    <LinearLayout
        android:layout_width="30pt"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="right"
        >
```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvPrice"
    android:gravity="center"
    android:textStyle="bold"
    />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvWeight"
    android:gravity="center"
    />

</LinearLayout>
</LinearLayout>

```

Внешний вид набора виджетов, Который представляет один объект продукт изображен на Рис. 9.2. На примере рисунка отображаются: флагок **android.widget.CheckBox**, текстовое поле с название продукта («Продукт 3»), текстовое поле со стоимостью продукта (19,0 у.е.) и текстовое поле — вес продукта (490 г.)



**Рис. 9.2.** Внешний вид набора виджетов, представленный в файле ресурсов из Листинга 9.3

Итак, теперь перейдем к созданию списка продуктов. В нашем примере список продуктов (как коллекция объектов **Product**) и заполнение виджета **android.widget.LinearLayout** (идентификатор «@+id/mainLL») виджетами

на основе макета (с помощью `android.view.LayoutInflater`) из Листинга 9.3 осуществляется в методе `onCreate` класса Активности. Программный код создания этих объектов представлен в Листинге 9.4.

#### Листинг 9.4. Создание наборов виджетов для отображения списка товаров на экране

```
public class MainActivity extends ActionBarActivity
{
    //--Class members-----
    /**
     * Контейнер LinearLayout, в который будут
     * добавляться элементы списка в виде набора
     * виджетов соответствующих полям объекта Product
     */
    private LinearLayout mainLL;

    /**
     * Сам список продуктов, содержимое которого будет
     * отображаться в нашем "списке".
     */
    private ArrayList<Product> products =
        new ArrayList<>();

    /**
     * Счетчик для генерации номера (идентификатора)
     * добавляемых продуктов
     */
    private int idCnt = 0;

    //--Class methods-----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
```

## 9. Пример формирования списка с помощью LayoutInflater...

```
setContentView(R.layout.activity_main);

//--Инициализация контейнера LinearLayout
//--(id "mainLL")-----
    this.mainLL = (LinearLayout) this.
                    findViewById(R.id.mainLL);

//--Заполнение списка "случайными" продуктами-----
for (int i = 0; i < 50; i++)
{
    this.products.add(new Product("Продукт " +
        (++this.idCnt), Math.round(Math.random() *
        10000) / 100, (int)(Math.random() * 50) * 10));
}

//--Получение ссылки на android.view.LayoutInflater--
LayoutInflater inflater = this.getLayoutInflater();

//--Формирование элементов списка-----
for (int i = 0; i < this.products.size(); i++)
{
    View         view   = inflater.inflate(
        R.layout.productitem, this.mainLL, false);

    Product product = this.products.get(i);

//--Название продукта-----
    TextView      tvName =
        (TextView)  view.findViewById(R.id.tvName);
    tvName.setText(product.name);

//--Стоимость продукта-----
    TextView      tvPrice     =
        (TextView)  view.findViewById(R.id.tvPrice);
    tvPrice.setText(String.valueOf(
        (product.price) + "y.e."));
```

```

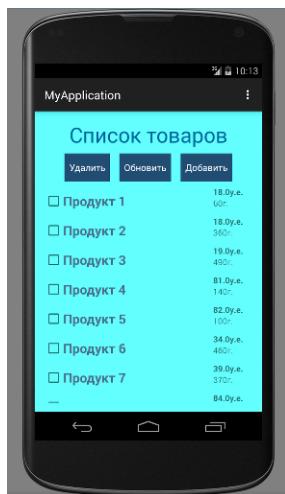
//---Вес продукта-----
        TextView tvWeight= (TextView) view;
        findViewById(R.id.tvWeight);
        tvWeight.setText(String.valueOf(product.
                weight) + "г.");
    }

//---Добавление созданного набора виджетов
//---в контейнер-----
        this.mainLL.addView(view);
    }
}

...
}

```

В Листинге 9.4 осуществляется перебор всех объектов **Product**, находящихся в коллекции **products** (поле объекта Активности рассматриваемого примера) и для каждого объекта **Product** создается набор виджетов с помощью



**Рис. 9.3.** Внешний вид списка продуктов, созданного с помощью программного кода из Листингов 9.2, 9.3, 9.4

объекта `android.view.LayoutInflater`. Затем каждый набор инициализируется в значения, соответствующие текущему продукту. Этот код в Листинге 9.4. выделен жирным текстом. Внешний вид полученного списка изображен на Рис. 9.3.

Запустив приложение, пользователь может просматривать все продукты, находящиеся в списке. При этом пользователь имеет возможность прокручивать список вверх или вниз.

Так же пользователю предоставляется возможность модифицировать список продуктов. Для этого необходимо нажать на одну из кнопок: «Удалить», «Обновить», «Добавить» (см. Рис. 9.3). Всем этим кнопкам (виджеты `android.widget.Button`) назначен один обработчик события нажатия — метод `btnActionClick`. Код этого метода приведен в Листинге 9.5.

**Листинг 9.5.** Код метода `btnActionClick`, обрабатывающего события нажатия на кнопки «Удалить», «Обновить», «Добавить» рассматриваемого примера

```
public void          btnActionClick(View view)
{
    //---Получение ID кнопки, по которой осуществлено
    //---нажатие-----
    int id      = view.getId();

    //---Выполнение действия, соответствующего
    //---текущей кнопке-----
    switch (id)
    {
        //---Кнопка "Удалить"-----
        case R.id.btnDelete :
            this.delCheckedProducts();
            break;
    }
}
```

```
//--Кнопка "Обновить"--  
        case R.id.btnUpd :  
            this.updCheckedProducts();  
            break;  
  
//--Кнопка "Добавить"--  
        case R.id.btnAdd :  
            this.addNewProduct();  
            break;  
    }  
}
```

Как видно из Листинга 9.5, метод **btnActionClick** построен классическим образом — вначале происходит идентификация кнопки, которая является источником события, затем происходит обработка события нажатия соответствующее нажатой кнопке. Для события кнопки «Удалить» вызывается метод **delCheckedProducts**, который удаляет все продукты, которые отметил пользователь с помощью флагжка (виджет **android.widget.CheckBox** напротив каждого продукта). Для события кнопки «Обновить» вызывается метод **updCheckedProducts**, который модифицирует все продукты, которые отметил пользователь. Модификация продукта осуществляется путем увеличения его стоимости на 1 у.е. и увеличения его веса на 10 г. Для события «Добавить» вызывается метод **addNewProduct**, который создает новый объект **Product**, добавляет его в коллекцию **products**, создает с помощью **android.view.LayoutInflater** набор виджетов и прикрепляет этот набор виджетов к контейнеру списка (**android.widget.LinearLayout** с идентификатором «mainLL»). Все эти методы являются методами, объявленными в классе Активности рассматриваемого примера. Рассмотрим подробнее эти методы.

Метод **delCheckedProducts** приведен в Листинге 9.6:

**Листинг 9.6.** Метод delCheckedProducts — удаление из списка всех отмеченных пользователем продуктов

```
/**  
 * Метод удаляет все отмеченные продукты  
 * в списке this.products и их отображения  
 * виджетов this.mainLL в списке  
 */  
private void delCheckedProducts()  
{  
    //---Определяем количество элементов списка  
    //---как количество "наборов" дочерних виджетов----  
    int cnt = this.mainLL.getChildCount();  
  
    //---Перебираем все наборы дочерние виджетов  
    //---в поисках отмеченных чекбоксов-----  
    for (int i = cnt - 1; i >= 0; i--)  
    {  
  
        //---Получаем текущий набор виджетов для одного  
        //---Продукта-----  
        View view = this.mainLL.getChildAt(i);  
  
        //---Получаем Флажок и его состояние-----  
        CheckBox cb = (CheckBox) view.  
                           findViewById(R.id.cb1);  
        if (cb.isChecked())  
        {  
  
            //---Продукт нужно удалить-----  
            this.mainLL.removeView(view);  
            this.products.remove(i);  
        }  
    }  
}
```

Метод **delCheckedProducts** выполняет следующую последовательность шагов:

1. Определяет количество элементов в списке, как количество дочерних наборов виджетов для контейнера **android.widget.LinearLayout** (который в нашем примере имеет идентификатор «mainLL» и является контейнером, содержащим список продуктов) с помощью вызова метода:

```
int getChildCount();
```

2. Перебирает все дочерние наборы виджета с помощью цикла и для каждого набора анализирует состояние флагажка. Получение ссылки на дочерний виджет происходит с помощью вызова метода:

```
View getChildAt(int index);
```

3. Если флагок установлен, то происходит удаление отмеченного продукта из коллекции **products** и удаление соответствующего набора виджетов из списка на экране. Для удаления дочернего виджета используется метод:

```
void removeView(View view);
```

Следующий метод **updCheckedProducts** приведен в Листинге 9.7:

**Листинг 9.7.** Метод **updCheckedProducts** — изменение всех выбранных пользователем продуктов

```
/**  
 * Метод обновляет все отмеченные продукты  
 * в списке this.products и их отображения в списке
```

```

* виджетов this.mainLL.
* Обновление осуществляется путем увеличения
* значений стоимости и веса для отмеченных
* продуктов на 1 у.е. и 10 г соответственно.
*/
private void updCheckedProducts()
{
//--Определяем количество элементов списка
//--как количество "наборов" дочерних виджетов---
    int cnt = this.mainLL.getChildCount();

//--Перебираем все наборы дочерние виджетов
//--в поисках отмеченных чекбоксов
    for (int i = cnt - 1; i >= 0; i--)
    {
//--Получаем текущий набор виджетов для одного
//--Продукта-----
        View view = this.mainLL.getChildAt(i);

//--Получаем Флажок и его состояние-----
        CheckBox cb = (CheckBox) view.
                        findViewById(R.id.cb1);
        if (cb.isChecked())
        {
//--Продукт нужно обновить - находим объект Product,
//--который соответствует выбранному набору в списке
//--виджетов-----
            Product P = this.products.get(i);

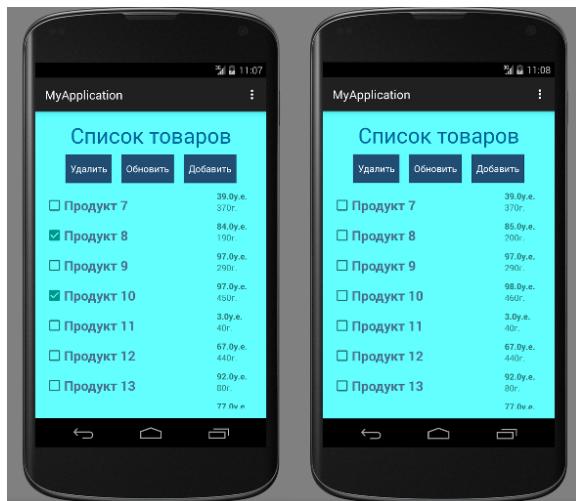
//--Изменяем значения в объекте Product-----
            P.price          += 1.0;
            P.weight         += 10;

//--Отображаем измененные значения в виджетах
//--выбранного набора ---
            TextView tvPrice =
                (TextView) view.findViewById(R.id.tvPrice);

```

```
        tvPrice.setText(String.valueOf(P.price) +  
                "у.е.");  
  
        TextView tvWeight = (TextView) view.  
                findViewById(R.id.tvWeight);  
        tvWeight.setText(String.valueOf(P.weight) +  
                "г.");  
  
        //---Снимаем выделение с Флажка-----  
        cb.setChecked(false);  
    }  
}  
}
```

Метод `updCheckedProducts` делает точно такую же последовательность шагов, что и метод `delCheckedProducts`, только вместо удаления метод вносит изменения в выбранные



**Рис. 9.4.** Пользователь выбирает два продукта (слева) и нажимает кнопку «Обновить». Справа — список после обновления информации в выбранных продуктах (отработал метод `updCheckedProducts`)

объекты **Product** из коллекции **products** и вносит изменения в виджеты, соответствующие этим выбранным объектам **Product**. Код внесения изменений в виджеты (текстовые поля **android.widget.TextView**) в Листинге 9.7 выделен жирным текстом. После внесения изменений, метод **updCheckedProducts** снимает выделение с флажка (чекбокса) модифицированного продукта. Внешний вид работы метода **updCheckedProducts** изображен на Рис. 9.4.

Следующий метод **addNewProduct** приведен в Листинге 9.8:

### **Листинг 9.8.** Метод addNewProduct — добавление нового продукта в список

```

/**
 * Метод добавляет новый продукт (объект Product)
 * в список продуктов this.products и в список
 * виджетов this.mainLL
 */
private void addNewProduct()
{
    //---Создаем новый объект Product-----
    Product P = new Product("Продукт " +
        (++this.idCnt), Math.round(Math.
            random() * 10000) / 100, (int)
            (Math.random() * 50) * 10);

    //---Добавляем в коллекцию продуктов-----
    this.products.add(P);

    //---Для добавления в список виджетов - создадим
    //---новый набор виджетов
    LayoutInflator inflater = this.getLayoutInflater();
    View view = inflater.inflate(R.layout.
        productitem, this.mainLL, false);
}

```

```
//--Название товара-----
    TextView tvName = (TextView) view.
                    findViewById(R.id.tvName);
    tvName.setText(P.name);

//--Стоимость товара-----
    TextView tvPrice = (TextView) view.
                    findViewById(R.id.tvPrice);
    tvPrice.setText(String.valueOf(P.price) +
                    "у.е.");

//--Вес товара-----
    TextView tvWeight= (TextView) view.
                    findViewById(R.id.tvWeight);
    tvWeight.setText(String.valueOf(P.weight) + "г.");

//--Добавление созданного набора виджетов в контейнер--
    this.mainLL.addView(view);

/*
 * Прокрутим ScrollViewer к концу списка
 * -----
 */
    ScrollView SV = (ScrollView) this.
                    findViewById(R.id.svOne);

//--Определим высоту каждого элемента списка
//--(каждого набора виджетов)-----
    int height = this.mainLL.getChildAt(0).getHeight();

//--Величина прокрутки = количество элементов *
//--высоту каждого элемента-----
    SV.smoothScrollTo(0, height * this.mainLL.
                    getChildCount());
}
```

Метод **addNewProduct** создает новый объект **Product**, заполняет поля этого объекта случайными значениями. После этого созданный объект добавляется в коллекцию **products**. Но добавление в коллекцию **products** не приводит к добавлению нового продукта на экране. Для этого необходимо создать с помощью объекта **android.view.LayoutInflater** новый набор виджетов, который будет представлять в списке на экране вновь созданный продукт. Созданный набор виджетов инициализируется значениями из созданного объекта **Product**, при этом ссылки на виджеты из набора («Название продукта», «Стоимость», «Вес») определяются с помощью вызова метода **findViewById**. Этот код в Листинге 9.8 выделен жирным. Затем созданный набор виджетов добавляется к контейнеру **mainLL** с помощью вызова метода **addView**.

Отдельный интерес представляет попытка выполнить программно прокрутку объекта **android.widget ScrollView** к только что добавленному набору виджетов. За эту функциональность отвечают последние три команды метода **addNewProduct**. Несмотря на правильность расчетов величины координат прокрутки, прокрутка происходит не к последнему вновь добавленному виджету, а к предпоследнему. Очевидно, что после добавления виджета сразу не происходит перерасчет размеров дочернего виджета в контейнере **android.widget ScrollView**, что является грустным фактом и приводит к недостатку в нашем примере.

В завершении особое внимание уделим основным неудобствам рассматриваемого в данном разделе примера:

- Необходимо одновременно вносить изменения как в коллекцию **products** так и в виджеты на экране.

Удобнее было бы вносить изменения в какую-то одну из коллекций. И такая возможность в Android разработке существует — это Адаптеры данных (<http://developer.android.com/intl/ru/reference/android/widget/Adapter.html>). Адаптер данных это своеобразный мостик между данными и их отображением в некотором объекте (View). Чаще всего адаптеры применяются для наборов данных, поэтому Адаптер также отвечает за создание View-компонента для каждого элемента данных из этого набора данных. Об Адаптерах данных будет рассказано ниже в этом уроке.

- Для каждого объекта **Product** из списка продуктов **products** создается свой набор виджетов, хотя на экране устройства в любой момент времени отображается лишь часть списка. Это означает, что остальные (невидимые) наборы виджетов занимают место в памяти, что является некорректным, так как в мобильных устройствах такой ресурс как память ограничен и его нужно экономить. Если поставить себе цель оптимизировать работу нашего списка из примера, то необходимо будет дописывать еще немаленькую функциональность. Однако, в Android разработке такой факт уже учтен и в стандартных списках (**android.widget.ListView**, **android.widget.ExpandableListView**) совместно с Адаптерами данных такая проблема экономии памяти для невидимых виджетов уже решена.

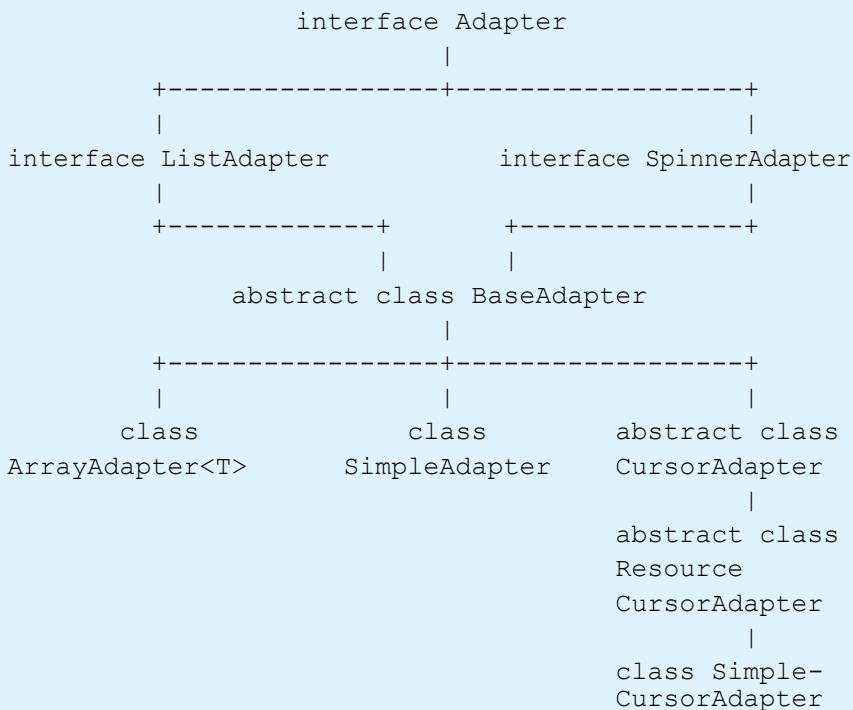
Итак, проанализировав неудобства программирования данного примера, возникает интерес к объекту Адаптера данных. Давайте перейдем к его изучению.

# 10. Понятие Адаптера данных (Adapter). Виды адаптеров данных

Адаптер данных служит мостом между Видом (View-компонент, отвечающий за отображение данных) и набором данных, предназначенным для отображения в этом Виде. Адаптер предоставляет доступ к этому набору данных и также отвечает за создание Видов (виджетов) для каждого элемента данных из этого набора. Но самое важное заключается в том, что где-бы не происходили изменения данных — в виджетах в результате действий пользователя или в наборе данных в результате работы программы — синхронизировать виджеты и набор данных не нужно! Адаптер данных это сделает автоматически!

В Android существует множество классов Адаптеров данных, каждый из которых ориентирован на соответствующий виджет списка. Родительским классом для всех классов Адаптеров является класс `android.widget.BaseAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/BaseAdapter.html>).

Иерархия некоторых классов и основных интерфейсов Адаптеров данных имеет следующий вид:



## 10.1. Интерфейсы Adapter, ListAdapter, SpinnerAdapter. Абстрактный класс BaseAdapter

В основе иерархии лежит интерфейс `android.widget.Adapter` (<http://developer.android.com/intl/ru/reference/android/widget/Adapter.html>), в котором объявлены следующие методы:

- **`abstract int getCount()`** — возвращает количество элементов в наборе данных, которые представляет Адаптер данных.
- **`abstract Object getItem(int position)`** — возвращает ссылку на объект элемента данных из набора (спи-

ска) данных. В качестве параметра принимает индекс элемента в наборе (в списке).

- **abstract long getItemId(int position)** — возвращает идентификатор строки, соответствующей позиции в списке данных.
- **abstract View getView(int position, View convertView, ViewGroup parent)** — возвращает виджет, который отображает данные из набора данных. Принимаемые параметры: **position** — индекс элемента данных в списке данных; **convertView** — предыдущий виджет, который использовался для этого (или другого) элемента списка, для возможного повторного использования этого виджета (обязательно необходимо проверить этот параметр на null перед использованием); **parent** — родительский контейнер, к которому будет прикреплен создаваемый виджет. В этом методе можно создать виджет или с помощью **new** или с помощью **android.view.LayoutInflater**.
- **abstract boolean isEmpty()** — возвращает true, если набор данных пуст (не содержит элементов).
- **abstract void registerDataSetObserver(DataSetObserver observer)** — регистрирует объект-наблюдатель, который будет получать уведомления об изменениях, происходящих в наборе данных.
- **abstract void unregisterDataSetObserver(DataSetObserver observer)** — отменяет регистрацию объекта-наблюдателя, который был ранее зарегистрирован с помощью метода **registerDataSetObserver**.

Далее, интерфейс **android.widget.ListAdapter** (<http://developer.android.com/intl/ru/reference/android/widget/>)

[ListAdapter.html](#)), предназначен для реализации в классах Адаптеров данных, предназначенных для взаимодействия с виджетом `android.widget.ListView` (<http://developer.android.com/reference/android/widget/ListView.html>). Виджет `android.widget.ListView` будет рассмотрен в следующих уроках. В интерфейсе `android.widget.ListAdapter` объявляются методы:

- **abstract boolean areAllItemsEnabled()** — возвращает `true`, если все элементы в Адаптере данных разрешены (то есть их можно выбрать с помощью клика или нажатия).
- **abstract boolean isEnabled(int position)** — возвращает `true`, если элемент, представленный индексом (`position`), в списке данных Адаптера разрешен для выбора с помощью клика/нажатия.

В интерфейсе `android.widget.SpinnerAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/SpinnerAdapter.html>) объявлены методы, необходимые для реализации в классах Адаптеров, которые предназначены для выпадающих (drop down) списков:

- **View getDropDownView(int position, View convertView, ViewGroup parent)** — возвращает виджет, который отображает выпадающий элемент списка. Принимаемые параметры: `position` — индекс элемента данных в наборе данных Адаптера; `convertView` — ссылка на виджет, который ранее использовался для этого или другого элемента списка, может быть `null`; `parent` — ссылка на родительский виджет-контейнер, к которому будет прикреплен создаваемый этим методом виджет.

Абстрактный класс `android.widget.BaseAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/BaseAdapter.html>) является базовым классом для всех классов Адаптеров данных. Класс `android.widget.BaseAdapter` реализует некоторые методы интерфейсов, которые наследует. Производным классам остаются на обязательную реализацию методы: `getView`, `getItemId`, `getItem`, `getCount`. Если хотите создать свой Адаптер данных — рекомендуется взять в качестве родительского класса `android.widget.BaseAdapter`.

## 10.2. Класс `ArrayAdapter<T>`

Далее по иерархии классов рассмотрим класс `android.widget.ArrayAdapter<T>` (<http://developer.android.com/intl/ru/reference/android/widget/ArrayAdapter.html>). Это готовый Адаптер данных, который предназначен для использования в таких списках как `android.widget.Spinner` и `android.widget.ListView`. Принимает в качестве набора данных список (коллекцию, реализующую интерфейс `List<T>` — например коллекцию `ArrayList<T>`) или массив объектов. По умолчанию, `android.widget.ArrayAdapter<T>` перебирает каждый элемент набора данных и вставляет строковое значение, которое формируется с помощью вызова метода `toString()` для отображаемого объекта данных, в указанный виджет `android.widget.TextView` (виджет `TextView` в виде идентификатора ресурсов передается в конструктор класса `android.widget.ArrayAdapter<T>`).

Класс `android.widget.ArrayAdapter<T>` кроме наследуемых методов содержит методы по работе с коллекцией данных — `add`, `insert`, `remove`, `sort`, `clear` и метод

`setDropDownViewResource` для задания макета из ресурсов для отображения пунктов выпадающего списка.

Если вы хотите использовать для отображения элементов списка другой виджет или виджеты вместо предлагаемого по умолчанию виджета `android.widget.TextView`, то для этих целей необходимо создать класс, производный от `android.widget.ArrayAdapter<T>` и переопределить в нем метод `getView(int, View, ViewGroup)`, в котором или с помощью оператора `new` или с помощью объекта `android.view.LayoutInflater` реализовать создание нужного вам виджета, отображающего элемент данных из набора данных Адаптера. Переопределение метода `getView` Адаптера данных будет продемонстрировано позже в уроках этого курса.

Поскольку класс Адаптера данных `android.widget.ArrayAdapter<T>` будет часто использоваться нами в уроках данного курса, то рассмотрим методы этого класса более подробно.

Конструкторы класса `android.widget.ArrayAdapter<T>`:

- `ArrayAdapter(Context context, int resource, T[] objects)` — Принимает идентификатор ресурса содержащего макет, представляющий внешний вид каждого элемента списка и массив объектов который является набором данных для Адаптера. Поскольку в качестве набора данных выступает массив, то такой набор данных является не модифицируемым с точки зрения добавления и удаления элементов, но менять содержимое в массиве объектов можно. Если нужен модифицируемый набор данных с точки зрения добавления и удаления элементов, то нужно использовать конструктор, принимающий `List<T>` (см. ниже по списку).

Если не переопределяется метод `getView` Адаптера, то идентификатор ресурса рекомендуется представлять виджетом `android.widget.TextView`.

- **ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)** — то же что и предыдущий конструктор, только принимает дополнительный параметр `textViewResourceId`, который содержит идентификатор виджета `android.widget.TextView` в макете, который представляется параметром `resource`. То есть в этот конструктор можно передать более сложный макет, состоящий из нескольких виджетов, среди которых должен быть виджет `android.widget.TextView` с идентификатором `textViewResourceId`, в который объект `ArrayAdapter<T>` будет размещать строковое представление объекта (получаемое с помощью `toString()`) из набора данных `objects`. Опять же, в набор данных нельзя будет добавлять новые элементы и удалять старые.
- **ArrayAdapter(Context context, int resource, List<T> objects)** — данный конструктор предназначен для работы с модифицируемым с точки зрения добавления и удаления набором данных. Во всем остальном, этот конструктор совпадает с первым в этом списке конструктором.
- **ArrayAdapter(Context context, int resource, int textViewResourceId, List<T> objects)** — данный конструктор предназначен для работы с модифицируемым с точки зрения добавления и удаления набором данных. Во всем остальном, этот конструктор совпадает с конструктором, который описан во втором пункте этого списка.

Часто используемые методы класса `android.widget.ArrayAdapter<T>`:

- `public void add(T object)` — добавляет новый объект `object` в набор данных Адаптера. Объект добавляется в конец списка.
- `public void clear()` — удаляет все объекты из набора данных Адаптера.
- `public int getCount()` — возвращает количество объектов в наборе данных Адаптера (количество элементов списка).
- `public View getDropDownView (int position, View convertView, ViewGroup parent)` — возвращает виджет, который отображает объект из набора данных Адаптера в выпадающем списке. Параметр `position` содержит индекс объекта в наборе данных Адаптера, параметр `convertView` содержит ссылку на старый виджет, который может быть использован повторно (если ссылка не равна `null` и повторное использование возможно, в противном случае метод создает новый виджет), параметр `parent` содержит ссылку на родительский виджет, к которому будет прикреплен виджет, возвращаемый этим методом. Этот метод можно переопределять в производных классах, для того чтобы, например, менять внешний вид создаваемых виджетов.
- `public T getItem (int position)` — возвращает ссылку на объект из набора данных Адаптера по индексу `position`.
- `public int getPosition (T item)` — возвращает индекс объекта `item` из набора данных.

- **public View getView (int position, View convertView, ViewGroup parent)** — возвращает виджет, который отображает объект из набора данных Адаптера в обычном списке. Параметр **position** содержит индекс объекта в наборе данных Адаптера, параметр **convertView** содержит ссылку на старый виджет, который может быть использован повторно (если ссылка не равна null и повторное использование возможно, в противном случае метод создает новый виджет), параметр **parent** содержит ссылку на родительский виджет, к которому будет прикреплен виджет, возвращаемый этим методом. Этот метод можно переопределять в производных классах, для того чтобы, например, менять внешний вид создаваемых виджетов.
- **public void insert (T object, int index)** — вставляет новый объект **object** в набор данных Адаптера в указанную позицию. Позиция, в которую осуществляется вставка, указывается параметром **index**.
- **public void notifyDataSetChanged ()** — метод предназначен для оповещения виджета-списка о том, что в назначенному им Адаптере данных произошли изменения и виджету-списку необходимо обновить свое представление (выполнить перерисовку). Обычно, этот метод не нужно вызывать явно — по умолчанию любые изменения в Адаптере данных (методы **add, insert, remove**) приводят к изменению внешнего вида виджета-списка и наоборот — изменения данных пользователем с помощью виджета приводят к обновлению данных в Адаптере данных. Однако, если отключить такую возможность с помощью метода

**setNotifyOnChange**, то автоматических обновлений происходить не будет.

- **public void remove (T object)** — удаляет объект **object** из набора данных Адаптера.
- **public void setDropDownViewResource (int resource)** — назначает макет внешнего вида из файла ресурсов, который будет использоваться для создания виджетов для элементов выпадающего списка.
- **public void setNotifyOnChange (boolean notifyOn-Change)** — если передать true (значение, установленное в Адаптере по умолчанию), то методы **add**, **insert**, **remove**, **clear** будут автоматически вызывать метод **notifyDataSetChanged**, в противном случае (если передать false), метод **notifyDataSetChanged** необходимо вызывать самостоятельно.
- **public void sort (Comparator<? super T> comparator)** — сортирует набор данных Адаптера, используя в качестве критерия сортировки объект **comparator**.

Наверное, у вас возник вопрос, а как назначать Адаптер данных виджету, который является списком? Все очень просто. Для этой цели в классах списках (например в классах **android.widget.Spinner** или **android.widget.ListView**) существует метод:

```
public void setAdapter (Adapter adapter);
```

Ну и соответственно, получить ссылку на Адаптер данных, назначенный списку можно с помощью метода:

```
public T getAdapter ();
```

### 10.3. Классы SimpleAdapter, CursorAdapter, SimpleCursorAdapter

Класс `android.widget.SimpleAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/SimpleAdapter.html>) является готовым к использованию Адаптером данных. Но в отличии от `android.widget.ArrayAdapter<T>`, который предназначен для отображения элементов списка, содержащих одно значение, класс `android.widget.SimpleAdapter` предназначен для отображения элементов списка, содержащих множество значений. Одно из существенных отличий `android.widget.SimpleAdapter` от `android.widget.ArrayAdapter<T>` это его конструктор:

```
public SimpleAdapter (Context context,
    List<? extends Map<String, ?>>
        data, int resource,
    String[] from, int[] to);
```

Конструктор принимает список Map-объектов, где каждый Map-объект — это список атрибутов, в виде пары «Ключ-Значение». Ключом является некое имя, а значением является величина, которую необходимо отобразить в виджете, представляющем элемент списка. Кроме этого конструктор принимает два массива — `from[]` и `to[]`. В массиве `to[]` указываются идентификаторы виджетов, а во `from[]` указываются имена из объектов Map, значения которых должны быть вставлены в соответствующие (из `from[]`) виджеты.

Также `android.widget.SimpleAdapter` содержит методы по наполнению виджетов значениями из Map — `setViewImage`, `setViewText`, `setViewBinder`. То есть можно

осуществлять привязку не только текстовых данных, но и изображений.

Класс `android.widget.CursorAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/CursorAdapter.html>) является абстрактным классом. Реализует абстрактные методы класса `android.widget.BaseAdapter`. Добавляет свои методы по работе с курсором. Для справки, курсор это интерфейс `android.database.Cursor` (<http://developer.android.com/intl/ru/reference/android/database/Cursor.html>) который предоставляет доступ для чтения и записи к результирующему набору данных, полученных из базы данных. Так вот, класс `android.widget.CursorAdapter` предназначен для извлечения данных из курсора `android.database.Cursor` и размещением этих данных в виджете `android.widget.ListView`.

Класс `android.widget.SimpleCursorAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/SimpleCursorAdapter.html>). Готовый класс Адаптера данных похож на `android.widget.SimpleAdapter`, только использует не набор объектов `Map`, а `android.database.Cursor`, т.е. набор строк с полями. Соответственно в массив `from[]` заносится наименования полей, значения которых необходимо извлечь в соответствующие виджеты из массива `to[]`:

```
public SimpleCursorAdapter (Context context,  
    int layout, Cursor c, String[] from, int[] to);
```

АдAPTERЫ данных, относящиеся к работе с `android.database.Cursor` будут использоваться нами в уроках рассматривающих взаимодействие Android приложений с Базами Данных.

# 11. Виджет Spinner — выпадающий список

Виджет **android.widget.Spinner** представляет собой выпадающий список (<http://developer.android.com/intl/ru/reference/android/widget/Spinner.html>). Виджет позволяет отображать только один элемент списка, который является выбранным. Пользователю предоставляется выбрать другой элемент списка, при этом все элементы, доступные для выбора, «выпадают» или «распахиваются». Именно поэтому виджет **android.widget.Spinner** и называют выпадающим списком.

Иерархия классов для класса **android.widget.Spinner**:

```
java.lang.Object
 |
 +--- android.view.View
   |
   +--- android.view.ViewGroup
     |
     +--- android.widget.AdapterView<
           android.widget.SpinnerAdapter>
       |
       +--- android.widget.AbsSpinner
         |
         +--- android.widget.Spinner
```

Элементы, которые отображаются в выпадающем списке **android.widget.Spinner** берутся из Адаптера данных. Поэтому нам очень пригодятся знания, полученные из раздела посвященного Адаптерам данных.

Для получения мгновенной информации о смене выбранного элемента предназначено событие **onItemSelected**. Для получения этих событий, необходимо создать класс, реализующий интерфейс **android.widget.AdapterView.OnItemSelectedListener** (<http://developer.android.com/intl/ru/reference/android/widget/AdapterView.OnItemSelectedListener.html>) в котором объявлены два метода:

```
public void onItemSelected (AdapterView<?> parent,  
                           View view, int position, long id);
```

И

```
public void onNothingSelected (AdapterView<?> parent);
```

Метод **onItemSelected** будет вызван при выборе пользователем элемента списка. Принимает следующие параметры:

- **AdapterView<?> parent** — ссылка на виджет-контейнер, чьи дочерние элементы находятся в Адаптере данных. В нашем случае это будет ссылка на Spinner.
- **View view** — виджет, который является элементом списка и который выбран пользователем.
- **int position** — индекс элемента в наборе данных Адаптера данных, который выбран.
- **long id** — идентификатор строки списка в которой выбран элемент.

Метод **onNothingSelected** вызывается при отсутствии выбора элемента в списке.

Также, для получения информации о выбранном элементе используются методы:

- **Object getSelectedItem()** — возвращает ссылку на объект из набора данных Адаптера, который представляется в выбранном элементе списка **android.widget.Spinner**.
- **int getSelectedItemPosition()** — возвращает индекс выбранного элемента.
- **View getSelectedView()** — возвращает ссылку на виджет, который представляет выбранный пользователем элемент списка.

Для того, чтобы сделать некоторый элемент списка **android.widget.Spinner** выбранным используется метод:

```
void setSelection(int position);
```

В следующих разделах рассматриваются примеры использования списка **android.widget.Spinner** с Адаптерами данных.

## 11.1. Применение **ArrayAdapter<T>** для **Spinner**

В данном разделе рассматривается пример применения списка **android.widget.Spinner** с Адаптером данных **android.widget.ArrayAdapter<T>**. Полную версию примера данного раздела можно найти в модуле «app9» среди файлов с исходными кодами примеров, прилагаемых к этому уроку.

Макет Активности примера представлен в Листинге 11.1.

### Листинг 11.1. Макет Активности для примера текущего раздела

```
<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=".MainActivity">

<Spinner
    android:id="@+id/month_spinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    />

</LinearLayout>

```

Создание Адаптера данных **android.widget.ArrayAdapter<T>** для данного примера осуществляется в методе **onCreate** класса Активности. Программный код, создающий Адаптер данных, представлен в Листинге 11.2.

**Листинг 11.2. Создание Адаптера  
android.widget.ArrayAdapter<String> и назначение  
его списку android.widget.Spinner**

```

public class MainActivity extends Activity
{
    //--Class members-----
    /**
     * Массив с названиями месяцев для размещения
     * в выпадающем списке
     */
    private String[] month =
    {
        "Январь", "Февраль", "Март", "Апрель",
        "Май", "Июнь", "Июль", "Август",
        "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"
    };
}

```

```
//--Class methods-----  
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
//--Получение ссылки на Spinner-----  
    Spinner spinner = (Spinner) this.findViewById(R.  
        id.month_spinner);  
  
//--Создание ArrayAdapter<String> на основе массива  
//--строк и стандартного макета для Spinner  
    ArrayAdapter<String> adapter =  
        new ArrayAdapter<>(this,  
            android.R.layout.simple_spinner_item,  
            month);  
  
//--Назначение макета для виджета элемента  
//--выпадающего списка-----  
    adapter.setDropDownViewResource(android.R.layout.  
        simple_spinner_dropdown_item);  
  
//--Назначение Адаптера данных списку Spinner-----  
    spinner.setAdapter(adapter);  
  
//--Назначение обработчика события выбора элемента  
//--списка-----  
    spinner.setOnItemSelectedListener(new  
        AdapterView.OnItemSelectedListener()  
{  
        @Override  
        public void onItemSelected(AdapterView<?> parent,  
            View view, int position, long id)  
        {  
            Toast.makeText(MainActivity.this,  
                MainActivity.this.month[position],  
                Toast.LENGTH_SHORT).show();  
        }  
    }  
}
```

```
    }

    @Override
    public void onNothingSelected(AdapterView<?>
        parent)
    {
    }
}) ;
}

...
}
```

В примере из Листинга 11.2 создается выпадающий список, содержащий в качестве элементов названия месяцев. Для названий месяцев создан массив **month** который является закрытым членом класса Активности **MainActivity**. Массив **month** проинициализирован строками, содержащими названия месяцев. Эти строки мы и увидим в выпадающем списке.

Далее, в методе **onCreate** создается Адаптер **android.widget.ArrayAdapter<T>** которому передается ссылка на Контекст приложения, идентификатор ресурса макета для виджета списка и непосредственно сам массив с названиями месяцев:

```
ArrayAdapter<String> adapter =
    new ArrayAdapter<>(this,
        android.R.layout.simple_spinner_item,
        month);
```

Обратите внимание, что в примере не создавался макет виджета для списка. Вместо этого использовался

стандартный макет (в Листинге 11.2 выделен жирным, представляет собой обычный `TextView`), идентификатор которого задается с помощью константы `android.R.layout.simple_spinner_item`. Компания Google позаботилась о разработчиках Android приложений и подготовила для них (значит и для нас с вами) простейшие макеты виджетов, представляющие элементы списка для всех существующих виджетов-списков. Это удобно и экономит время. Ниже в этом разделе будет показан пример с использованием созданных нами макетов.

Далее, созданному Адаптеру (локальная переменная `adapter`) с помощью метода `setDropDownViewResource` назначается идентификатор ресурса, содержащего макет представляющий каждый элемент выпадающего списка. Опять же, в данном примере мы воспользовались стандартным макетом, представленным с помощью константы `android.R.layout.simple_spinner_dropdown_item` (опять же в примере Листинг 11.2 выделено жирным, представляет собой обычный `TextView`).

Следующим шагом является назначение Адаптера `adapter` списку `android.widget.Spinner` с помощью вызова метода `setAdapter`.

Последним действием примера является назначение списку `android.widget.Spinner` обработчика события выбора элемента списка пользователем. Обработчик события представлен объектом безымянного класса, наследуемым от `android.widget.AdapterView.OnItemSelectedListener`. В методе `onItemSelected` происходит вывод с помощью объекта `Toast` названия выбранного пользователем месяца.

Внешний вид работы примера изображен на Рис. 11.1.



**Рис. 11.1.** Внешний вид списка `android.widget.Spinner` из рассматриваемого примера. Слева — в обычном состоянии, справа — в распахнутом состоянии

Если есть необходимость использовать отличные макеты для элементов списка от стандартных макетов, то необходимо поступить следующим образом. Во-первых, создать файл ресурсов макета. Как это делается, вы уже должны знать — об этом рассказывалось в предыдущих уроках. В нашем примере добавим два макета — один для обычного вида списка (файл ресурсов /res/layout/my\_spinner\_item.xml), второй для представления выпадающих элементов списка (файл ресурсов /res/layout/my\_dropdown\_spinner\_item.xml). В Листинге 11.3 представлен xml код для макета виджета обычного вида списка `android.widget.Spinner` (`TextView` с текстом зеленоватого цвета размером 14 pt):

**Листинг 11.3.** Файл ресурсов /res/layout/my\_spinner\_item.xml для макета виджета выбранного элемента списка

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#2A6F3A"
    android:textSize="14pt">
</TextView>
```

В Листинге 11.4 представлен xml код для макета виджета, представляющего выпадающий элемент списка **android.widget.Spinner** (TextView с текстом синего цвета размера 12 pt, курсив):

**Листинг 11.4.** Файл ресурсов /res/layout/my\_dropdown\_spinner\_item.xml для макета виджета элемента выпадающего списка

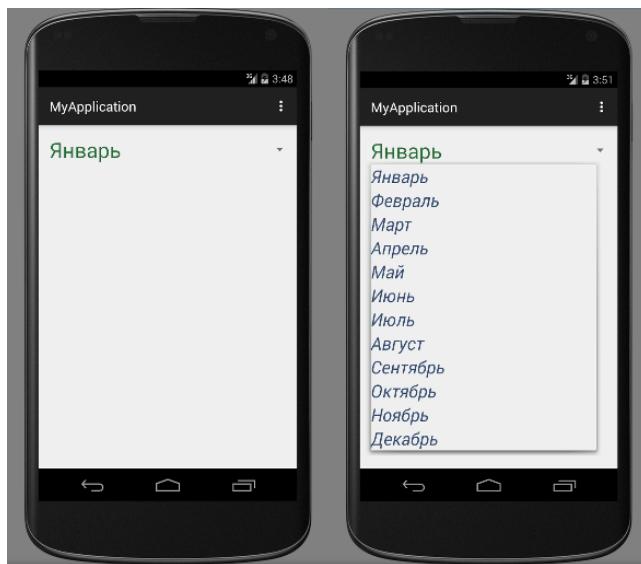
```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#364B6F"
    android:textSize="12pt"
    android:textStyle="italic">
</TextView>
```

Чтобы назначить эти макеты нашему списку **android.widget.Spinner** с названиями месяцев, необходимо заметить в нашем примере значения идентификаторов файлов ресурсов для макетов (выделено жирным):

```
ArrayAdapter<String> adapter =  
    new ArrayAdapter<>(this,  
        /*android.R.layout.simple_spinner_item*/  
        // Вместо стандартного  
        R.layout.my_spinner_item,  
        // Ставим свой ID  
        month);
```

И

```
adapter.setDropDownViewResource(  
    /*android.R.layout.simple_spinner_dropdown_item*/  
    // Вместо стандартного  
    R.layout.my_dropdown_spinner_item);  
    // Ставим свой ID
```



**Рис. 11.2.** Внешний вид списка android.widget.Spinner из рассматриваемого примера с нашими макетами виджетов для элементов списка. Слева — список в обычном состоянии, справа — список в распахнутом состоянии

Внешний вид примера с нашими макетами для элементов списка `android.widget.Spinner` изображены на Рис. 11.2.

## 11.2. Применение SimpleAdapter для Spinner

Особый интерес представляет применение Адаптера данных `android.widget.SimpleAdapter` (<http://developer.android.com/intl/ru/reference/android/widget/SimpleAdapter.html>) для списков. Как уже рассказывалось выше в этом уроке, этот Адаптер данных предназначен для набора данных, который содержит элементы списка каждый из которых состоит из нескольких значений. Конечно же, Адаптер `android.widget.SimpleAdapter` более подходит для такого списка как `android.widget.ListView`, чем для списка `android.widget.Spinner`. Но поскольку список `android.widget.ListView` будет рассматриваться в следующих уроках, а список `android.widget.Spinner` уже рассмотрен нами, то для примера этого раздела будем использовать именно `android.widget.Spinner`. Тем более, что принципы работы Адаптера `android.widget.SimpleAdapter` с разными списками абсолютно одинаков.

Пример, рассматриваемый в этом разделе находится в модуле «app9» файлов исходного кода, прилагаемых к текущему уроку.

Для примера на вновь понадобится класс `Product`, который мы уже использовали ранее в этом уроке (см. Листинг 9.1). Наш выпадающий список `android.widget.Spinner` будет содержать список Продуктов. Как раз то что надо — каждый продукт состоит из нескольких значений (название, стоимость, вес).

Первым делом в примере создается макет виджета для элементов списка `android.widget.Spinner`. Чтобы упростить пример, будет создан один макет который будет применен как для выбранного элемента списка так и для выпадающих элементов списка. Макет виджета находится в файле ресурсов `/res/layout/my_spinner_item2.xml` и представлен в Листинге 11.5:

**Листинг 11.5.** Макет виджета для списка `android.widget.Spinner` рассматриваемого примера

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:id="@+id/tvName"
        android:textColor="#003366"
        android:textStyle="bold"
        android:textSize="10pt"
        android:layout_weight="1" />

    <LinearLayout
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:layout_gravity="right">

        <TextView
```

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvPrice"
    android:gravity="center"
    android:textStyle="bold"    />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvWeight"
    android:gravity="center"    />

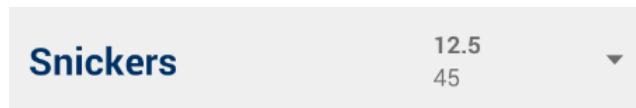
</LinearLayout>

</LinearLayout>

```

В Листинге 11.5 жирным выделены строки, содержащие идентификаторы виджетов `TextView` в которые и будут размещаться значения каждого продукта из набора данных которые будут находиться в Адаптере данных `android.widget.SimpleAdapter`.

Внешний вид макета виджета из Листинга 11.5 изображен на Рис. 11.3.



**Рис. 11.3.** Внешний вид макета из Листинга 11.5

Создание объекта Адаптера данных `android.widget.SimpleAdapter` и назначение его списку `android.widget.Spinner` осуществляется в методе `onCreate` класса Активности нашего примера. Программный код представлен в Листинге 11.6:

**Листинг 11.6.** Создание Адаптера данных  
android.widget.SimpleAdapter и назначение его списку  
android.widget.Spinner

```
public class MainActivity extends Activity
{
    //---Class methods-----
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //---Получение ссылки на нужный Spinner-----
        Spinner spnProducts = (Spinner) this.
            findViewById(R.id.spnProducts);

        //---Шаг 1. Создание массива продуктов-----
        Product[] products =
        {
            new Product("Snickers", 12.5, 45),
            new Product("Mars", 11.7, 50),
            new Product("CocaCola", 15.9, 1000),
            new Product("Bounty", 16.2, 55)
        };

        //---Шаг 2. Создание Map и List коллекций для
        //---android.widget.SimpleAdapter-----
        ArrayList<Map<String, Object>> prodList =
            new ArrayList<>();

        for (int i = 0; i < products.length; i++)
        {
            HashMap<String, Object> prodItem =
                new HashMap<>();
            prodItem.put("Name", products[i].name);
            prodItem.put("Price", products[i].price);
        }
    }
}
```

```
        prodItem.put("Weight", products[i].weight);

        prodList.add(prodItem);
    }

//--Шаг 3. Создание Адаптера данных
//--android.widget.SimpleAdapter --
SimpleAdapter prodAdapter = new SimpleAdapter(
    this,
    prodList,
    R.layout.my_spinner_item2,
    new String[] { "Name", "Price", "Weight" },
    new int[] { R.id.tvName, R.id.tvPrice,
        R.id.tvWeight });

prodAdapter.setDropDownViewResource(R.layout.
    my_spinner_item2);

//--Шаг 4. Назначение списку spnProducts Адаптера
//--данных-----
spnProducts.setAdapter(prodAdapter);

//--Шаг 5. Назначение обработчика события
//--выбранного элемента для spnProducts-----
spnProducts.setOnItemSelectedListener(
    new AdapterView.OnItemSelectedListener()
{
    @Override
    public void onItemSelected(
        AdapterView<?> parent, View view,
        int position, long id)
    {
//--Получение выбранного объекта Product
//--из Адаптера данных-----
        HashMap<String, Object> prodItem =
            (HashMap<String, Object>)
                parent.getAdapter().getItem(position);
```

```

//--Формирование строки для вывода в Тосте-----
    String msg =
        "Name : " + prodItem.get("Name") + "\n" +
        "Price: " + prodItem.get("Price") + "\n" +
        "Weigth:" + prodItem.get("Weight");

//--Вывод информации о выбранном продукте в Тосте---
    Toast.makeText(MainActivity.this,
        msg, Toast.LENGTH_SHORT).show();
}

@Override
public void onNothingSelected(AdapterView<?>
    parent)
{
}
});

}

...
}
}

```

**Первым шагом** примера из Листинга 11.6 это создание массива Продуктов **products** (массив объектов **Product**). В нашем примере массив **products** играет роль источника данных. В реальном приложении источником данных будет например База Данных или файл с данными. Массив **products** заполняется несколькими продуктами.

**Вторым шагом** является создание коллекции **List<? extends Map<String, ?>** которая необходима для передачи Адаптеру данных **android.widget.SimpleAdapter**. Вначале объявляется сама коллекция **prodList**:

```

ArrayList<Map<String, Object >> prodList =
    new ArrayList<>();

```

Затем перебирается массив **products** и каждый объект **Product** из массива помещается в отдельный объект **HashMap<String, Object>** (переменная **prodItem**):

```
HashMap<String, Object> prodItem = new HashMap<>();
prodItem.put("Name", products[i].name);
prodItem.put("Price", products[i].price);
prodItem.put("Weight", products[i].weight);

prodList.add(prodItem);
```

Обратите внимание, что каждый объект **Product** помещается в коллекцию **Map** не одним целым объектом, а по отдельности каждым полем. Причем каждому значению поля из объекта **Product** назначается строковый ключ (выделено жирным). Этот ключ будет позже использоваться для связи значений (название, цена, вес) и виджетов, в которых эти значения должны будут отображаться.

Каждый объект **Map** (переменная **prodItem**) затем помещается в коллекцию **prodList**.

**Третьим шагом** примера является создание Адаптера данных **android.widget.SimpleAdapter** (переменная **prodAdapter**):

```
SimpleAdapter prodAdapter = new SimpleAdapter(
    this,
    prodList,
    R.layout.my_spinner_item2,
    new String[] { "Name", "Price", "Weight" },
    new int[] { R.id.tvName, R.id.tvPrice,
        R.id.tvWeight });

prodAdapter.setDropDownViewResource(R.layout.
    my_spinner_item2);
```

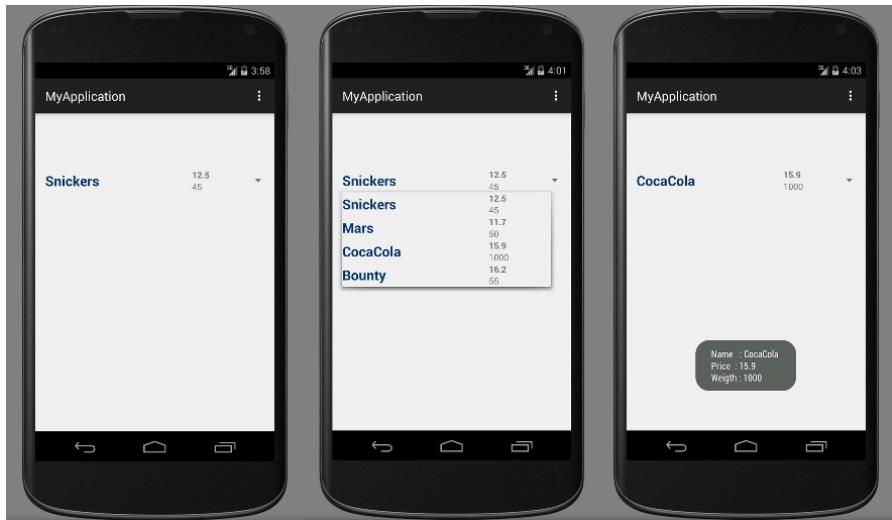
На этом шаге, в качестве виджета для выбранного элемента списка и выпадающих элементов списка используется макет из файла ресурсов /res/layout/my\_spinner\_item2 (идентификатор R.layout.my\_spinner\_item2). Обратите внимание, что жирным шрифтом на этом шаге выделены массивы соответствий **from[]** и **to[]**. В массиве **from[]** указаны названия ключей для отображаемых значений (название, цена, вес) из коллекций **Map**, а в массиве **to[]** указаны идентификаторы виджетов, в которых эти значения должны будут отображаться. Из Листинга видно, что значение объекта **Product** с ключом «Name» нужно отобразить в виджете с идентификатором **R.id.tvName** из макета **R.layout.my\_spinner\_item2**. А значения с ключами «Price» и «Weight» нужно отобразить в виджетах с идентификаторами **R.id.tvPrice** и **R.id.tvWeight** соответственно.

**Четвертым шагом** примера (см. Листинг 11.6) является назначение объекта Адаптера **android.widget.simpleAdapter** (переменная **prodAdapter**) списку **android.widget.Spinner** (переменная **spnProducts**).

**Пятым шагом** примера (см. Листинг 11.6) является назначение списку **spnProducts** обработчика события выбора элемента списка пользователем. В обработчике события происходит извлечение значений объекта **Product** который соответствует выбранному пользователем элементу списка и отображение этих значений в объекта **Toast**.

Внешний вид работы примера из Листингов 11.5 и 11.6 изображен на Рис. 11.4.

## 11. Виджет Spinner — выпадающий список



**Рис. 11.4.** Внешний вид работы примера из Листингов 11.5 и 11.6

# 12. Необходимость сохранения состояния Активности при повороте устройства

Как уже неоднократно упоминалось, при повороте устройства происходит пересоздание объекта Активности приложения. То есть происходит уничтожение существующего текущего объекта Активности и создание нового объекта Активности, к которому система применяет параметры текущей ориентации устройства (в частности, применяется раскладка макета Активности). Это означает, что все поля текущего объекта Активности, содержащие значения (будем называть их состояниями) необходимые для выполнения приложения, уничтожаются вместе с этим объектом. Затем при создании нового объекта Активности значения этих полей принимают свои начальные величины (но никак не текущие величины для исполняемого приложения). Важно понимать и владеть механизмами передачи значений между уничтожаемым объектом Активности и вновь создаваемым. Это и является темой текущего раздела данного урока.

## 12.1. Жизненный цикл Активности при повороте устройства

В прошлом уроке уже рассматривался жизненный цикл объекта Активности и состояния Активности, через которые она проходит на протяжении своего жизненного

цикла. Этот материал в этом уроке повторно рассказывается не будет — вы можете самостоятельно еще раз прочитать соответствующие разделы предыдущего урока. Однако для удобства восприятия информации (и чтобы легче было вспомнить основные состояния Активности), диаграмма жизненного цикла Активности из прошлого урока изображается в этом уроке на Рис. 12.1.



**Рис. 12.1.** Жизненный цикл Активности и состояния, через которые она проходит

Нас интересует последовательность событий, через которые проходит Активность при повороте устройства. Для этой цели, создадим пример (в модулях с исходным кодом данный пример отсутствует из-за своей несложности), код которого представлен в Листинге 12.1. Суть примера заключается в том, чтобы добавить методы-обработчики событий для всех событий, через которые проходит Активность на диаграмме, изображенной на Рис. 12.1.

**Листинг 12.1.** Обработчики событий из диаграммы Рис. 12.1, через которые проходит Активность

```
public class MainActivity extends Activity
{
    //---Class constants-----
    /**
     * Для вывода отладочных сообщений через Log.d
     */
    private final static String TAG = "MainActivity";

    /**
     * События, через которые проходит Активность
     */
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(TAG, "===== onCreate : Активность создается");
    }

    @Override
    protected void onStart()
    {
```

```
super.onStart();
Log.d(TAG, "===== onStart :
Становится видимой для пользователя");
}

@Override
protected void onResume()
{
    super.onResume();
    Log.d(TAG, "===== onResume :
Выходит на передний план");
}

@Override
protected void onPause()
{
    super.onPause();
    Log.d(TAG, "===== onPause :
Уходит с переднего плана");
}

@Override
protected void onStop()
{
    super.onStop();
    Log.d(TAG, "===== onStop : Перестает быть видимой");
}

@Override
protected void onDestroy()
{
    super.onDestroy();
    Log.d(TAG, "===== onDestroy :
Завершается или уничтожается");
}
...
```

Запустим пример из Листинга 12.1 в эмуляторе и повернем устройство с помощью эмулятора. Напомним, что для смены ориентации устройства используется комбинация клавиш Ctrl+F11 для эмуляторов Google. Однако данный пример можно смело применять и для других эмуляторов, например для Genymotion.

Итак, запустив пример и выполнив поворот устройства, мы увидим в окне LogCat сообщения нашего приложения, изображенные в Листинге 12.2:

**Листинг 12.2.** Сообщения, выводимые в окно LogCat примером из Листинга 12.1

```
MainActivity: ===== onPause      : Уходит с переднего плана
MainActivity: ===== onStop       : Перестает быть видимой
MainActivity: ===== onDestroy   : Завершается или
                           уничтожается
MainActivity: ===== onCreate     : Активность создается
MainActivity: ===== onStart      : Становится видимой
                           для пользователя
MainActivity: ===== onResume    : Выходит на передний план
```

Это и есть последовательность событий, через которые проходит Активность при повороте устройства:

$$\begin{matrix} \text{onPause} & \rightarrow & \text{onStop} & \rightarrow & \text{onDestroy} & \rightarrow \\ & & \text{onCreate} & \rightarrow & \text{onStart} & \rightarrow & \text{onResume} \end{matrix}$$

Пока можно сделать предварительные выводы о том, что для сохранения состояний можно использовать обработчики событий **onStop** или **onDestroy**, а для восстановления состояний можно использовать обработчик события **onCreate**. Однако не будем спешить. Разумеется, компания Google предоставляет разработчикам Android приложений механизмы сохранения состояний и события

**onStop** и **onDestroy** возможно и не придется использовать вовсе, кроме разве что ситуации, когда вы будете применять свой собственный механизм сохранения/восстановления состояний (никто не говорит что это не возможно).

## **12.2. Пример, демонстрирующий необходимость сохранения состояний**

Для рассмотрения существующих механизмов сохранения состояний нам понадобится пример, на основе которого мы, во-первых, увидим суть проблемы, а во-вторых, будем применять рассматриваемые механизмы для решения проблемы. Примером нашего раздела будет игра «Угадай слово». Суть игры следующая — приложение загадывает случайное слово из массива предустановленных английских слов и пользователю дается некоторое количество попыток угадать одну букву или назвать все слово целиком. Угадывание одной буквы или одного слова считается одной попыткой. Когда все попытки будут исчерпаны, то приложение сообщает пользователю, что он проиграл и загадывает следующее слово. Если пользователь угадывает слово или отгадывает все буквы слова и у него не исчерпаны все попытки, то приложение сообщает ему, что он выиграл и загадывает следующее слово. В общем, несложное приложение. Исходный код этого приложения находится в модуле «app10» файлов исходного кода, прилагаемых к этому уроку. В виду важности примера (на случай, если по каким-то причинам у вас был утерян исходный код), Листинги файлов `activity_main.xml` и `MainActivity.java` приводятся в этом разделе в Листингах 12.3 и 12.4.

соответственно. Сама логика работы приложения игры «Угадай слово» не будет объясняться. При желании (а это приветствуется), вы можете самостоятельно разобраться с программной логикой организации игры (исходные коды снабжены комментариями). Также необходимо отметить, что для уменьшения размеров xml кода примера виджетам назначались только минимальные визуальные стили и не создавались строковые константы для строковых сообщений используемых в приложении (что на самом деле не рекомендуется).

### Листинг 12.3. Макет внешнего вида Активности приложения «Угадай слово»

```
<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=".MainActivity">

    <TextView
        android:text="Угадай слово"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="14pt" />

    <Space
        android:layout_width="match_parent"
        android:layout_height="16dp"/>
```

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:text="Количество попыток : " />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:text="10"
        android:id="@+id/tvTryCount" />

</LinearLayout>

<Space
    android:layout_width="match_parent"
    android:layout_height="16dp" />

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16pt"
        android:id="@+id/tv1"
```

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16pt"
        android:id="@+id/tv2"
        android:text="*"      />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16pt"
        android:id="@+id/tv3"
        android:text="*"      />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16pt"
        android:id="@+id/tv4"
        android:text="*"      />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="16pt"
        android:id="@+id/tv5"
        android:text="*"      />

</LinearLayout>

<Space
    android:layout_width="match_parent"
    android:layout_height="16dp" />

<EditText
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:id="@+id/edtMain" />

<Space
    android:layout_width="match_parent"
    android:layout_height="16dp" />

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="btnClick"
        android:text="Угадать букву"
        android:id="@+id	btnSymbol" />

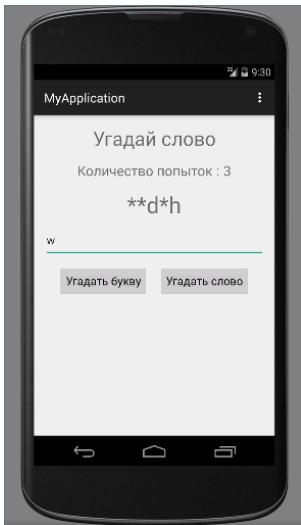
    <Space
        android:layout_width="16dp"
        android:layout_height="wrap_content"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="btnClick"
        android:text="Угадать слово"
        android:id="@+id	btnWord" />

</LinearLayout>

</LinearLayout>
```

Внешний вид макета Активности рассматриваемого примера «Угадай слово» изображен на Рис. 12.2.



**Рис. 12.2.** Внешний вид макета Активности рассматриваемого примера «Угадай слово»

Как видно из Листинга 12.3 и Рис.12.2 в макете Активности есть текстовое поле (`android.widget.TextView`) с идентификатором `tvTryCount` — в это текстовое поле приложение выводит количество оставшихся у пользователя попыток на отгадывание слова. Так же есть текстовые поля (`android.widget.TextView`) с идентификаторами `tv1`, `tv2`, `tv3`, `tv4`, `tv5` — в эти текстовые поля приложение выводит информацию об отгаданных буквах загаданного слова. Звездочка означает что буква еще не угадана. Виджет `android.widget.EditText` с идентификатором `edtMain` предназначен для ввода пользователем буквы или слова. Ну и кнопки «Угадать букву» или «Угадать слово» представляют пользователю сделать свой ход.

Программный код Java приложения приведен в Листинге 12.4. Не обязательно сразу же вникать в смысл

программного кода из этого Листинга. Как уже говорилось выше, из-за важности рассматриваемого примера, его код приводится в уроке в максимально допустимом для ознакомления виде — то есть в таком виде, чтобы можно было разобраться с примером не обращаясь к файлам исходного кода.

#### Листинг 12.4. Исходный код Java приложения «Угадай слово»

```
public class MainActivity extends ActionBarActivity
{
    //--Class constants-----
    /**
     * Максимальное количество попыток отгадывания букв
     */
    private final static int MAX_TRY_COUNT = 10;

    /**
     * Количество символов в загаданном слове
     * в игре одинаково
     */
    private final static int SYMBOLS_IN_WORD = 5;

    //--Class members-----
    /*
     * Поля объекта, относящиеся к игре
     * -----
     */

    /**
     * Массив слов для загадывания
     */
    private String[] words =
    {
        «Apple», «Lemon», «Hello», «World», «Board»,
        «House», «Horse», «April», «Woman», «Table»,
    }
```

```
    «River», «Dream», «Green», «Stone», «Water»,
    «Rover», «Width», «Drive», «Pause»
};

/***
 * Слово, которое загадано
 */
private String word;

/***
 * Текущее состояние отгадываемого слова
 */
private String curWord;

/***
 * Счетчик попыток отгадывания букв.
 */
private int count;

/*
 * Поля объекта, относящиеся к виджетам
 * -----
 */
/***
 * Текстовое поле, содержащее количество
 * оставшихся попыток
 */
private TextView tvTryCount;

/***
 * Массив текстовых полей TextView, в которых
 * будут располагаться буквы загаданного слова
 * (Идентификаторы в макете : tv1, tv2, tv3, tv4, tv5)
 */
private TextView[] tvSymbols =
    new TextView[MainActivity.SYMBOLS_IN_WORD];
```

```
/**  
 * Массив числовых идентификаторов текстовых полей  
 * TextView, в которых располагаются буквы загаданного  
 * слова. Массив необходим для инициализации  
 * массива tvSymbols  
 */  
private int[] tvSymbolsId =  
{  
    R.id.tv1, R.id.tv2, R.id.tv3, R.id.tv4, R.id.tv5  
};  
  
/**  
 * EditText, куда будут вводится буквы или слово  
 */  
private EditText edtMain;  
  
//--Class methods-----  
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    //--Инициализация полей, относящихся к виджетам---  
    this.tvTryCount = (TextView) this.  
        findViewById(R.id.tvTryCount);  
    this.edtMain = (EditText) this.  
        findViewById(R.id.edtMain);  
  
    for (int i = 0; i < this.tvSymbols.length; i++)  
    {  
        this.tvSymbols[i] = (TextView) this.  
            findViewById(this.tvSymbolsId[i]);  
    }  
  
    //--Инициализация игры-----  
    this.initGame();  
}
```

```
/***
 * Метод осуществляет инициализацию игры
 * И
 * Отображение внешнего вида состояния игры
 */
private void initGame()
{
    //--Вывод Тоста с сообщением-----
    Toast.makeText(this, «Новая игра»,
        Toast.LENGTH_SHORT).show();

    //--Получение случайного слова из массива
    //--предустановленных слов-----
    this.word = this.words[(int) (Math.random() *
        this.words.length)];

    //--Текущее отгадываемое слово - все буквы
    //--из звездочек-----
    this.curWord      = «*****»;

    //--Количество попыток на отгадывание равно
    //--начальному количеству-----
    this.count       = MainActivity.MAX_TRY_COUNT;

    //--Очищаем поле ввода буквы или слова-----
    this.edtMain.setText("");

    //--Отображение внешнего вида состояния игры-----
    this.showGame();
}

/***
 * Отображение внешнего вида состояния игры
 */
private void showGame()
{
```

```
//--Вывод количества оставшихся попыток-----
    this.tvTryCount.setText(String.valueOf(this.
        count));;

//--Вывод состояния отгадываемого слова в поля
//--tv1-tv5-----
    for (int i = 0; i < this.curWord.length();
        i++)
    {
        this.tvSymbols[i].setText(this.curWord.
            substring(i, i + 1));
    }
}

/***
 * Обработчик событий нажатия на кнопки
 * "Угадать букву" и "Угадать слово" @param v -
 * ссылка на виджет Button, на который нажал
 * пользователь
 */
public void btnClick(View v)
{
//--Проверка на начало новой игры-----
    if (this.count == 0)
    {
        this.initGame();
        return;
    }

//--Читаем значение текстового поля edtMain-----
    String      S      = this.edtMain.getText().
        toString().toLowerCase();

//--Проверка на пустое поле-----
    if (S.isEmpty())
    {
```

```
        Toast.makeText(this, "Вы ничего не ввели",
                      Toast.LENGTH_SHORT).show();
    return;
}

//--Уменьшаем счетчик оставшихся попыток-----
this.count--;

//--Идентифицируем кнопку-----
if (v.getId() == R.id.btnSymbol)
{
//--Пользователь нажал на кнопку "Угадать букву"---

S = S.substring(0, 1);
if (this.word.toLowerCase().contains(S))
{
    Toast.makeText(this, "Есть такая буква",
                  Toast.LENGTH_SHORT).show();

//--Открываем отгаданную букву в загаданном слове--
for (int i = 0;
     i < this.tvSymbols.length; i++)
{
    String symbol =
        this.word.substring(i, i + 1);
    if (symbol.toLowerCase().equals(S))
    {
        this.curWord= this.curWord.
        substring(0, i) + symbol +
        this.curWord.substring(i + 1);
    }
}

//--Проверяем, что все буквы отгаданы-----
if (this.curWord.equals(this.word))
{
```

```
        this.count = 0;
        Toast.makeText(this,
                      "Вы угадали!",
                      Toast.LENGTH_SHORT).show();
    }
}
else
{
    Toast.makeText(this, "Нет такой буквы",
                  Toast.LENGTH_SHORT).show();
}
}
else
if (v.getId() == R.id.btnExit)
{
//--Пользователь нажал на кнопку "Угадать слово"--+
    if (S.equals(this.word.toLowerCase()))
    {
//--Пользователь угадал слово-----
        this.count = 0;
        Toast.makeText(this, "Вы угадали!",
                      Toast.LENGTH_SHORT).show();

//--Открываем угаданное слово-----
        this.curWord      = this.word;
    }
    else
    {
        Toast.makeText(this, "Слово не угадано",
                      Toast.LENGTH_SHORT).show();
    }
}

//--Очищаем значение текстового поля edtMain-----
this.edtMain.setText("");
```

```

//--- Отображаем текущее состояние игры -----
    this.showGame();

//--- Если количество попыток исчерпано, то сообщаем
//--- о конце игры -----
    if (this.count == 0)
    {
        Toast.makeText(this, "Игра завершена.\nНажмите любую кнопку для новой игры.",
                    Toast.LENGTH_SHORT).show();
    }
}

...
}

```

В Листинге 12.4. жирным выделены три поля объекта класса **MainActivity**:

```

/***
 * Слово, которое загадано
 */
private String word;

/***
 * Текущее состояние отгадываемого слова
 */
private String curWord;

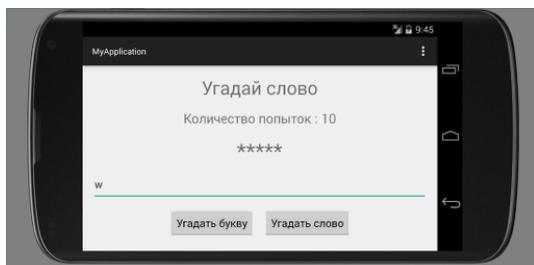
/***
 * Счетчик попыток отгадывания букв
 */
private int count;

```

Эти поля и есть те состояния, которые мы должны научится сохранять и восстанавливать в данном примере. Но сейчас достаточно всего лишь запустить на исполнение

рассматриваемое приложение и сыграть для ознакомления несколько раз в игру, пока не поворачивая устройство. Сыграли? Отлично! Отгадали? Тоже отлично!

Теперь начните играть и в середине игрового процесса поверните устройство. Если до поворота устройства приложение выглядело как на Рис. 12.2, то после поворота внешний вид приложения станет как на Рис. 12.3.



**Рис. 12.3.** Внешний вид приложения «Угадай слово»  
после поворота устройства

Разумеется, после поворота устройства потерялись значения полей **word**, **curWord** и **count** класса Активности. И фактически пользователь начал новую игру. Обратите внимание, что состояние виджета **android.widget.EditText** с идентификатором **edtMain** осталось таким же как и до поворота. Дело в том что некоторые виджеты умеют сохранять свое состояние при повороте Активности. Но в нашем примере это ничего не меняет — поля **word**, **curWord**, **count** были инициализированы заново, так как их инициализация осуществляется в методе **onCreate** при создании Активности, а мы не знаем — создается ли Активность впервые или в результате поворота устройства.

Пока не знаем.

## 12.3. Сохранение состояний Активности с помощью Bundle. Событие onSaveInstanceState

В этом разделе рассматривается механизм сохранения состояний, который для разработчиков Android приложений считается стандартным. Это механизм с использованием объекта `android.os.Bundle` (<http://developer.android.com/intl/ru/reference/android/os/Bundle.html>). Дословный перевод слова `bundle` с английского языка означает «пакет», «пачка», «сверток».

Объект `android.os.Bundle` представляет из себя коллекцию объектов вида «ключ-значение». В качестве ключа выступает объект `String`, а в качестве значений могут быть или простейшие типы Java, или объекты, реализующие интерфейс `Serializable`, или объекты реализующие интерфейс `android.os.Parcelable` (<http://developer.android.com/intl/ru/reference/android/os/Parcelable.html>). Об интерфейсе `android.os.Parcelable` будет рассказано в последующих уроках данного курса. Здесь всего лишь упомянем о том, что `android.os.Parcelable` является более быстрой альтернативой сериализации и используется при передаче данных между Активностями.

Операционная система сообщает объекту Активности о необходимости сохранения состояния приложения с помощью события `onSaveInstanceState`:

```
public void onSaveInstanceState(Bundle  
        savedInstanceState);
```

Методу, обрабатывающему это событие, передается уже созданный системой объект коллекция `android.`

**os.Bundle** (параметр `savedInstanceState`). Разработчику приложения необходимо всего лишь воспользоваться этой коллекцией для размещения в ней значений, которые необходимо сохранить и восстановить при следующем создании объекта Активности.

Обратите внимание, что обработчик события `onCreate` принимает в качестве параметра ссылку на объект `android.os.Bundle`:

```
protected void onCreate(Bundle savedInstanceState);
```

Если ссылка на этот объект (параметр `savedInstanceState`) не равна `null`, значит наше приложение ранее уже выполнило сохранение данных в `android.os.Bundle` и необходимо эти данные восстановить. Все просто.

Когда же вызывается метод `onSaveInstanceState?` Обычно этот метод вызывается системой перед `onPause`, `onStop` и `onDestroy`. Вы можете самостоятельно это проверить, добавив путем переопределения в пример из Листинга 12.1 метод `onSaveInstanceState`, как показано в Листинге 12.5.

#### Листинг 12.5. Добавление метода `onSaveInstanceState` в класс Активности приложения

```
@Override
public void onSaveInstanceState(Bundle
        savedInstanceState)
{
    super.onSaveInstanceState(savedInstanceState);
    Log.d(TAG, "===== onSaveInstanceState");
}
```

Метод `onSaveInstanceState` используется не только при поворотах. Активность также может быть уничтожена

когда пользователь не будет взаимодействовать с устройством длительное время (например отойдет от устройства), а Операционной системе Android понадобится освободить память. ОС Android никогда не уничтожит для освобождения памяти выполняемую Активность. Чтобы Активность была уничтожена, она должна находиться в приостановленном или остановленном состоянии. Если Активность приостановлена или остановлена, значит метод **onSaveInstanceState** был вызван. При сохранении данных в объекте **android.os.Bundle** в методе **onSaveInstanceState** ОС Android заносит объект **android.os.Bundle** в так называемую Запись активности (*Activity Record*). Когда Активность сохранена, объект **Activity** не существует, но объект Записи активности существует в ОС. При необходимости Операционная система может воссоздать Активность по Записи активности.

В некоторых ситуациях ОС Android не только уничтожает Активность, но и полностью завершает процесс приложения. Даже в этом случае Запись активности продолжает существовать некоторое время, что позволяет быстро перезапустить Активность при возвращении пользователя.

Запись активности пропадает когда пользователь нажимает кнопку «Back». При этом Активность уничтожается и Запись активности теряется. Записи активности также уничтожаются при перезагрузке и возможно их уничтожение если они слишком долго не используются.

И еще хочется добавить, что Активность может перейти в сохраненное состояние без вызова метода **onDestroy**. Однако методы **onPause** и **onSaveInstanceState** будут вызваны

всегда. Обычно метод **onSaveInstanceState** используется для сохранения данных в объект **android.os.Bundle**, а **onPause** для всех остальных действий, которые могут понадобиться выполнить вашему приложению.

Давайте рассмотрим некоторые методы класса **android.os.Bundle** (<http://developer.android.com/intl/ru/reference/android/os/Bundle.html>), для того чтобы иметь представление об этом классе.

Методы класса **android.os.Bundle** для размещения данных:

- **void putПростейшийТип(String key, ПростейшийТип value)** — помещает в объект **android.os.Bundle** значение простейшего типа данных (**String, boolean, byte, int** и т. д.) Методы соответственно называются **putString**, **putBoolean**, **putByte** и т.д.
- **void putПростейшийТипArray(String key, ПростейшийТип[] value)** — помещает в объект **android.os.Bundle** массив простейших типов. Методы соответственно будут называться **putStringArray**, **putBooleanArray** и т.д.
- **void putSerializable(String key, Serializable value)** — помещает в объект **android.os.Bundle** объект, реализующий интерфейс **Serializable**.
- **void putParcelable(String key, Parcelable value)** — помещает в объект **android.os.Bundle** объект, реализующий интерфейс **android.os.Parcelable**.

Методы класса **android.os.Bundle** для извлечения данных:

- **ПростейшийТип getПростейшийТип(String key)** — извлекает из объекта **android.os.Bundle** значение

простейшего типа (**String**, **boolean**, **byte**, **int** и т. д.). Методы будут соответственно называться **getString**, **getBoolean**, **getByte** и т. д. В качестве параметра, методы принимают ключ, который был использован при сохранении этого значения.

- **ПростейшийТип[] getПростейшийТипArray(String key)** — извлекает из объекта **android.os.Bundle** массив значений простейшего типа.
- **ПростейшийТип getПростейшийТип(String key, ПростейшийТип defaultValue)** — извлекает из объекта **android.os.Bundle** значение простейшего типа. В этом методе предлагается задать значение по умолчанию, которое будет возвращено в случае, если значения с указанным ключом **key** в объекте **android.os.Bundle** не найдется.
- **Object get(String key)** — извлекает из **android.os.Bundle** помещенный туда ранее объект с ключом **key**.

Другие методы класса **android.os.Bundle**:

- **void remove(String key)** — удаляет значение из коллекции **android.os.Bundle** с ключом **key**.
- **void clear()** — удаляет все значения из коллекции **android.os.Bundle**.
- **int size()** — возвращает количество значений в коллекции **android.os.Bundle**.
- **Set<String> keySet()** — возвращает набор ключей из коллекции **android.os.Bundle**.

Теперь вернемся к нашему примеру «Угадай слово» и модифицируем его чтобы значения полей **word**, **curWord**

и **count** сохранялись в объекте **android.os.Bundle** при повороте устройства.

Исходные коды, содержащие изменения в приложении «Угадай слово», которые будут осуществляться в этом разделе, вы не найдете среди файлов с исходными кодами. Это сделано специально, чтобы вы внесли изменения самостоятельно в исходный код примера модуля «app10».

Первым шагом будет создание констант для класса Активности **MainActivity** которые будут строковыми ключами. Использоваться эти строковые ключи будут при размещении значений и при извлечении значений в **android.os.Bundle**. Создание строковых ключей представлено в Листинге 12.6.

**Листинг 12.6.** Шаг 1. Добавление в класс **MainActivity** строковых констант, являющихся ключами

```
public class MainActivity extends Activity
{
    //--- Class constants -----
    /**
     * Ключ для сохранения в объект
     * android.os.Bundle значения поля word
     */
    private final static String BUNDLE_KEY_WORD =
        "keyWord";

    /**
     * Ключ для сохранения в объект
     * android.os.Bundle значения поля curWord
     */
    private final static String BUNDLE_KEY_CURWORD =
        "keyCurWord";
```

```

    /**
     * Ключ для сохранения в объект
     * android.os.Bundle значения поля count
     */
    private final static String BUNDLE_KEY_COUNT =
            "keyCount";
    ...
}

```

Вторым шагом изменений будет создание (переопределение) метода **onSaveInstanceState** для того чтобы сохранить в объект **android.os.Bundle** значения полей **word**, **curWord** и **count**. Изменения этого шага представлены в Листинге 12.7.

### **Листинг 12.7.** Шаг 2. Сохранение в android.os.Bundle необходимых полей в методе onSaveInstanceState

```

@Override
public void onSaveInstanceState(Bundle
                               savedInstanceState)
{
    super.onSaveInstanceState(savedInstanceState);

    //-- Сохранение в android.os.Bundle необходимых
    //-- полей -----
    savedInstanceState.putString(MainActivity.
                                BUNDLE_KEY_WORD, this.word);
    savedInstanceState.putString(MainActivity.
                                BUNDLE_KEY_CURWORD, this.curWord);
    savedInstanceState.putInt    (MainActivity.
                                BUNDLE_KEY_COUNT,   this.count);
}

```

Третьим шагом изменений является модификация кода в методе **onCreate**. Теперь метод должен проверить — не

были ли сохранены в полученном объекте `android.os.Bundle` значения полей `word`, `curWord` и `count`. Если были сохранены, то возобновить игру «Угадай слово», в противном случае создать новую игру. Для этой цели, в методе `onCreate` вместо кода из Листинга 12.4:

```
//-- Инициализация игры -----
this.initGame();
```

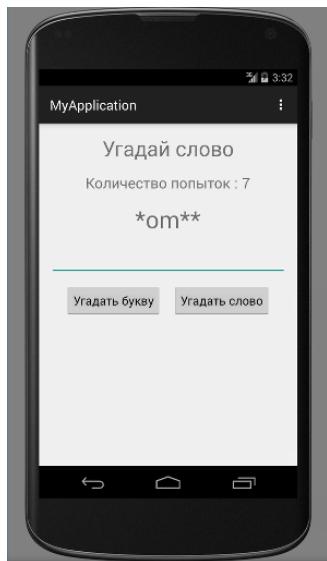
нужно написать код, представленный в Листинге 12.8:

#### Листинг 12.8. Шаг 3. Восстановление ранее сохраненного состояния игры в методе `onCreate`

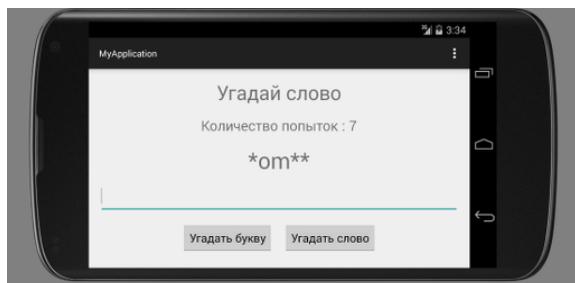
```
//-- Проверка, нужно ли восстановить игру
//-- или начать заново -----
if (savedInstanceState != null)
{
    //-- Состояние игры было сохранено -
    //-- восстанавливаем его -----
    this.word = savedInstanceState.
        getString(MainActivity.BUNDLE_KEY_WORD);
    this.curWord = savedInstanceState.
        getString(MainActivity.BUNDLE_KEY_CURWORD);
    this.count = savedInstanceState.
        getInt (MainActivity.BUNDLE_KEY_COUNT);

    //-- Отображаем текущее состояние игры -----
    this.showGame();
}
else
{
    //-- Инициализация игры -----
    this.initGame();
}
```

Результат работы модифицированного приложения «Угадай слово» изображен на Рис. 12.4 (до поворота устройства) и на Рис. 12.5 (после поворота устройства).



**Рис. 12.4.** Внешний вид модифицированной игры «Угадай слово» до поворота устройства



**Рис. 12.5.** Внешний вид модифицированной игры «Угадай слово» после поворота устройства.  
Состояние игры восстановлено

## 12.4. Использование статических полей для сохранения состояний

Как рассказывалось в предыдущем разделе, при повороте устройства текущий объект Активности уничтожается, и создается новый объект Активности. Но при этом само приложение продолжает работать. Это приводит к идею, что если сделать нужные для сохранения при поворотах поля класса Активности статическими, то их значения будут сохранены. Такое предположение является верным и его можно использовать на практике. Сохранение состояний приложения в статических полях программно реализуется проще, чем например сохранение состояний приложения в объекте `android.os.Bundle`. Однако имеет ряд недостатков. Например, когда ОС Android в результате бездействия пользователя с неактивным приложением не только уничтожает Активность, но и полностью завершает процесс приложения. В этом случае значения статических полей будут потеряны. Но в рассматриваемом нами случае (а мы изучаем сохранение состояний приложения при повороте устройства) значения статических полей будут сохранены.

Модифицируем приложение из Листинга 12.4 таким образом, чтобы поля `word`, `curWord`, `count` были статическими и при повороте устройства состояние игры не менялось.

Первым шагом изменений является перевод полей `word`, `curWord`, `count` из обычных полей уровня объекта в статические поля. Это изменение отображено в Листинге 12.9.

### Листинг 12.9. Шаг 1. Перевод полей класса Активности word, curWord, count в статические поля

```

    /**
     * Слово, которое загадано
     */
    private static String word;

    /**
     * Текущее состояние отгадываемого слова
     */
    private static String curWord;

    /**
     * Счетчик попыток отгадывания букв
     */
    private static int count;

```

Вторым шагом является внесение изменений в методе `onCreate` вместо кода из Листинга 12.4:

```
//---Инициализация игры-----
this.initGame();
```

необходимо написать код из Листинга 12.10:

### Листинг 12.10. Шаг 2. Изменения в методе onCreate для восстановления состояния игры «Угадай слово»

```

//---Проверка, нужно ли восстановить игру
//---или начать заново -----
if (MainActivity.word != null)
{
    //---Отображение текущего состояния игры -----
    this.showGame();
}

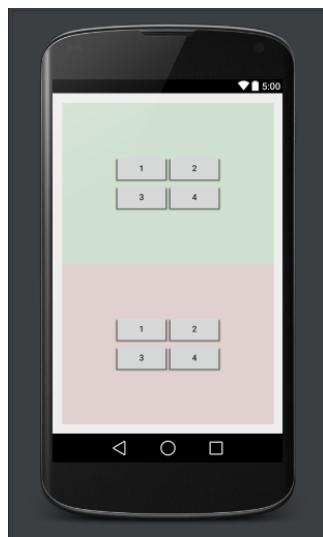
```

```
else
{
    //--Инициализация игры-----
    this.initGame();
}
```

Результат работы изменений из Листингов 12.9 и 12.10 выглядит, как показано на Рис. 12.4 и Рис. 12.5. Чтобы проверить работу механизма сохранения состояний в приложении текущего примера с помощью статических полей, вам необходимо будет проделать шаги из Листингов 12.9 и 12.10 самостоятельно.

# 13. Домашнее задание

1. Используя менеджеры раскладки `android.widget.FrameLayout` и `android.widget.GridLayout`, а также другие полученные знания из уроков данного курса, выполните xml верстку макета Активности как показано на Рис. 13.1.



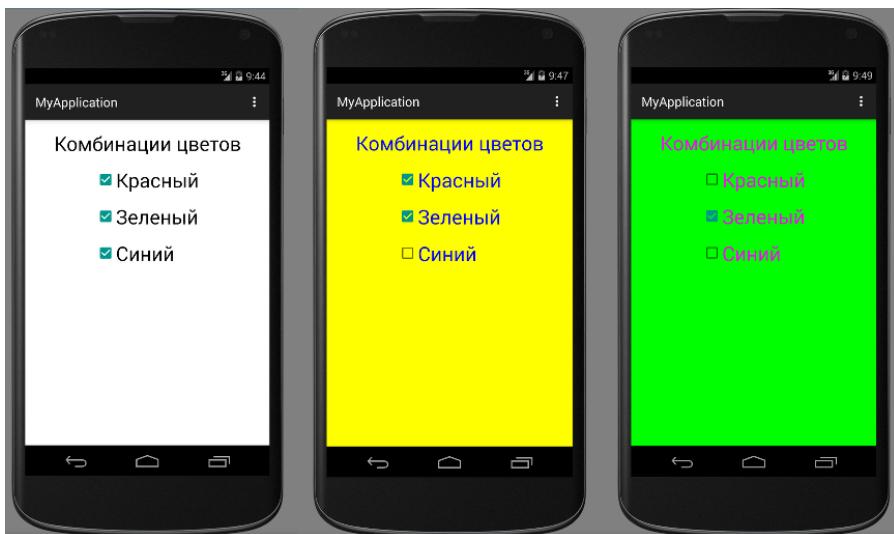
**Рис. 13.1.** Макет Активности для домашнего задания

2. Выполните программную (с помощью Java) верстку макета Активности изображенную на Рис. 13.1.

3. Используя виджеты `android.widget.CheckBox`, реализуйте работу приложения, как показано на Рис. 13.2.

**ПОЯСНЕНИЕ:** С помощью флагков `android.widget.CheckBox` пользователь имеет возможность комбинировать

RGB цвета фона Активности. Обратите внимание, что при этом цвета текста виджетов `android.widget.TextView` инвертированы к цвету фона Активности.



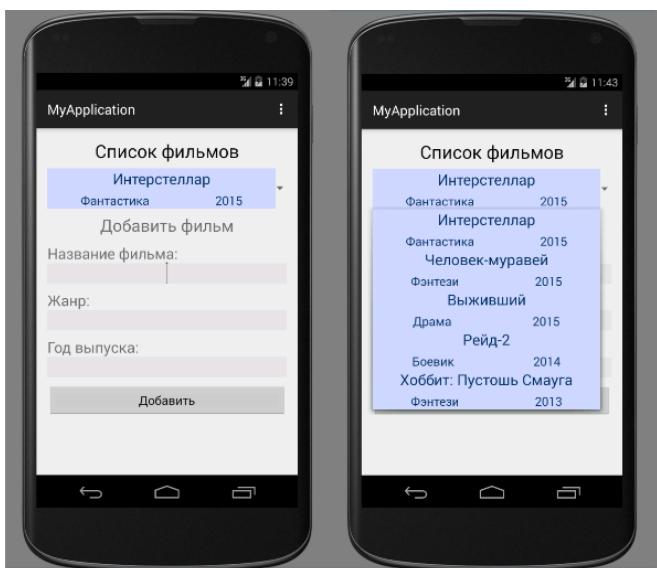
**Рис. 13.2.** Внешний вид работы приложения для домашнего задания

4. Внесите самостоятельно изменения в пример из Листинга 12.4 для сохранения состояния игры «Угадай слово» в объект `android.os.Bundle`, как показано в Листингах 12.6, 12.7, 12.8.

5. Внесите самостоятельно изменения в пример из Листинга 12.4 для сохранения состояния игры «Угадай слово» в статических полях, как показано в Листингах 12.9 и 12.10.

6. Напишите приложение (см. Рис. 13.3), в котором будет использоваться список `android.widget.Spinner`,

в котором будут размещаться (с помощью адаптера данных) объекты класса **Film** (см. Листинг 13.1). Класс **Film** состоит из полей: название фильма, жанр фильма и год выпуска (Поля класса **title**, **genre** и **year** соответственно). Необходимо использовать Адаптер данных **android.widget.SimpleAdapter**, с помощью которого необходимо разместить поля каждого объекта **Film** из набора данных Адаптера таким образом, как показано на Рис. 13.3. Приложение должно предоставлять пользователю возможность добавление новых фильмов (объектов **Film**) в список **android.widget.Spinner** (помните, что новые элементы списка добавляются через адаптер данных). Приложение должно сохранять список объектов при повороте устройства.

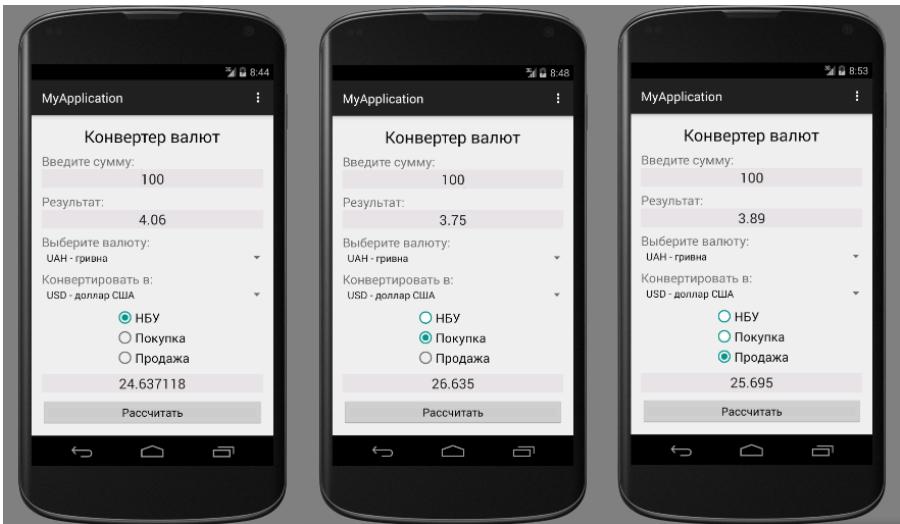


**Рис. 13.3.** Внешний вид приложения  
для домашнего задания

**Листинг 13.1.** Класс Film, объекты которого должны размещаться в списке android.widget.Spinner для приложения из домашнего задания

```
/**  
 * Class Film - Инкапсулирует информацию о фильме  
 * -----  
 */  
class Film  
{  
    //---Class members-----  
    /**  
     * Название фильма  
     */  
    public String title;  
  
    /**  
     * Жанр фильма  
     */  
    public String genre;  
  
    /**  
     * Год выпуска фильма  
     */  
    public int year;  
  
    //---Class methods-----  
    public Film(String title, String genre, int year)  
    {  
        this.title = title;  
        this.genre = genre;  
        this.year = year;  
    }  
}
```

7. Необходимо разработать приложение «Конвертер валют». Внешний вид приложения показан на Рис. 13.4.



**Рис. 13.4.** Внешний вид приложения «Конвертер валют» из домашнего задания

Суть работы приложения следующая. Пользователь вводит сумму в текстовое поле «Введите сумму» (использовать `android.widget.EditText`) и выбирает из списков (использовать `android.widget.Spinner`) валюту, из которой необходимо выполнить конвертацию («Выберите валюту») и валюту, в которую выполняется конвертация («Конвертировать в»). Использовать три валюты: «UAH — гривна», «USD — доллар США», «EUR — Евро». Результат выводится в текстовое поле «Результат» (использовать `android.widget.TextView`). Расчет суммы конвертации валюты зависит от типа курса: «курс НБУ», «Покупка», «Продажа». Для каждого из этих типов курсов существует свой расчетный курс валют (задан в виде констант в приложении), который в зависимости от выбора валют и типа курса (с помощью `android.widget.RadioButton`)

отображается в текстовом поле `android.widget.EditText` над кнопкой «Рассчитать». Пользователь должен иметь возможность вводить свой расчетный курс для каждого типа курса в текстовое поле `android.widget.EditText` (которое над кнопкой «Рассчитать»). Приложение должно запоминать при повороте устройства, введенные пользователем расчетные курсы валют.



## Урок №2

# Структура Android-проекта. Пользовательский интерфейс приложения

© Бояршинов Юрий

© Компьютерная Академия «Шаг»

[www.itstep.org](http://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.