

ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ
ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ

ANDROID



Урок №3

Layout, Views

Содержание

1.	Виджет android.widget.ListView	5
1.1.	Виджет android.widget.ListView	
одиночного выбора		14
1.2.	Виджет android.widget.ListView	
множественного выбора		18
2.	События прокрутки для виджета	
android.widget.ListView.....		22
3.	Создание custom-виджетов для элементов	
списка android.widget.ListView		25
4.	Диалоговые окна.	
Классы AlertDialog, AlertDialog.Builder		45
4.1.	Классы AlertDialog, AlertDialog.Builder.....	46

4.2. AlertDialog со списком одиночного выбора ..	54
4.3. AlertDialog со списком множественного выбора	60
4.4. AlertDialog с пользовательским набором виджетов.....	69
4.5. DatePickerDialog.	
Диалоговое окно для выбора даты	74
4.6. TimePickerDialog.	
Диалоговое окно для выбора времени.....	81
5. Работа с файлами на внешнем носителе	88
5.1. Общедоступные каталоги внешнего носителя	102
5.2. Создание файлов на внешнем носителе	106
5.3. Практический пример: Диалоговое окно для выбора и открытия файлов находящихся на внешнем носителе	117
5.4. Еще несколько слов о работе с внешним носителем	132
6. Работа с файлами на внутреннем носителе	136
7. Каталог assets приложения	142
8. Взаимодействие с файлами растровых изображений. Класс Bitmap.....	148
9. Отображение изображений в приложении. Виджет android.widget.ImageView.....	153
10. Виджет android.widget.GridView.....	161
11. Домашнее задание	176

1. Виджет android.widget.ListView

Виджет **android.widget.ListView** представляет собой список элементов с вертикальной прокруткой. Данные для элементов списка поставляются из Адаптера Данных, который должен наследовать интерфейс **android.widgetListAdapter** (<http://developer.android.com/reference/android/widget/ListAdapter.html>).

Иерархия классов для виджета **android.widget.ListView** выглядит следующим образом:

```
java.lang.Object
|
+-- android.view.View
|
+-- android.view.ViewGroup
|
+-- android.widget.AdapterView<
    android.widget.ListAdapter>
|
+-- android.widget.AbsListView
|
+-- android.widget.ListView
```

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

Создание виджета **android.widget.ListView** с помощью xml ничем не отличается от создания ранее знакомых

вам виджетов. В Листинге 1.1 представлена xml-верстка макета Активности с виджетом **android.widget.ListView**.

Листинг 1.1. Создание виджета android.widget.ListView с помощью xml

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

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

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

    <ListView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/lvMonth">

        </ListView>
</LinearLayout>
```

Создание Адаптера Данных с данными для элементов списка **android.widget.ListView** ничем не отличается от создания Адаптера Данных для списка **android.widget.Spinner**. В Листинге 1.2 представлен программный код

Java, с помощью которого создается Адаптер Данных `android.widget.ArrayAdapter<T>` для заполнения виджета `android.widget.ListView` из Листинга 1.1 списком названий месяцев.

Листинг 1.2. Создание Адаптера Данных `android.widget.ArrayAdapter<T>` для виджета `android.widget.ListView`

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

    //--Получение ссылки на виджет
    //---android.widget.ListView-----
    ListView lvMonth = (ListView)
        this.findViewById(R.id.lvMonth);

    //--Создание адаптера данных-----
    String[] arrMonth =
    {
        "Январь", "Февраль", "Март", "Апрель",
        "Май", "Июнь", "Июль", "Август",
        "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"
    };

    ArrayAdapter<String> lvAdapter1 =
        new ArrayAdapter<>(this, android.R.layout.
            simple_list_item_1, arrMonth);

    //--Назначение Адаптера Данных списку
    //---android.widget.ListView-----
    lvMonth.setAdapter(lvAdapter1);
}
```

Внешний вид примера из Листингов 1.1 и 1.2 представлен на Рис. 1.1.

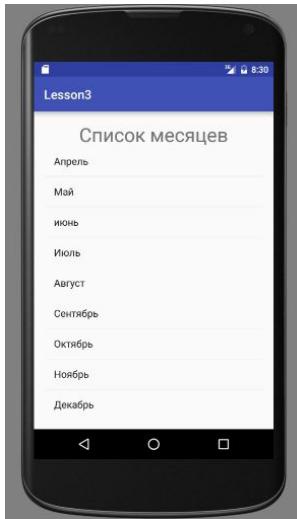


Рис. 1.1. Внешний вид примера из Листингов 1.1 и 1.2

Как видно в Листинге 1.2, в качестве макета для виджетов, представляющих элементы списка взят стандартный макет `android.R.layout.simple_list_item_1`. То есть, для виджета `android.widget.ListView` (так же как и для виджета `android.widget.Spinner`) разработчики компании Google создали небольшой набор готовых макетов для элементов списка. Разумеется, эти готовые виджеты имеют не броский внешний вид и выглядят аскетично. Поэтому давайте сразу дополним рассматриваемый пример собственным макетом для виджета для элементов списка `android.widget.ListView`.

Создадим в модуле новый файл ресурсов с макетом. Для этого нужно кликнуть правой кнопкой мыши по папке `/res/layout` и в появившемся контекстном меню выбрать пункт «*New/Layout resource file*». Дадим имя файлу ресурсов название `my_listview_item` и в качестве корневого элемента макета выберем элемент `TextView` (см. Рис. 1.2).

Урок №3

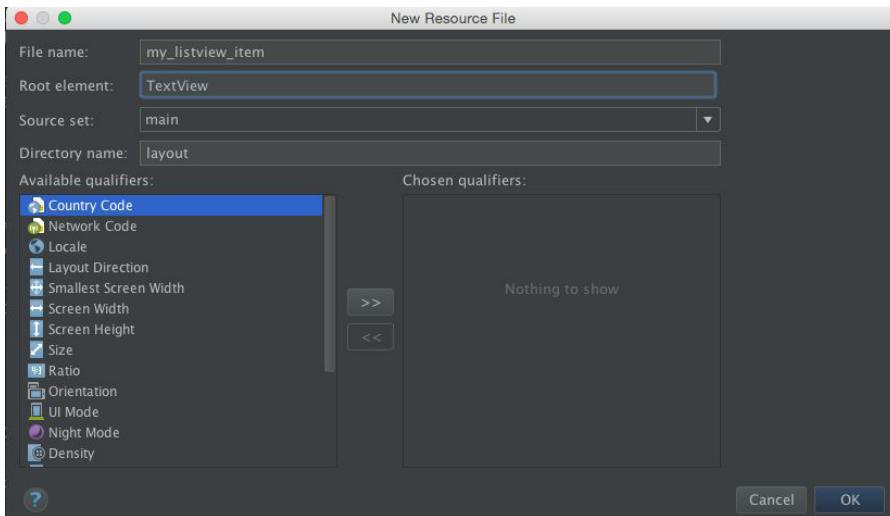


Рис. 1.2. Диалоговое окно создания нового файла ресурсов с макетом виджета для рассматриваемого примера

В созданном файле /res/layout/my_listview_item.xml придадим внешний вид макету виджета который будет представлять каждый элемент списка `android.widget.ListView` (см. Листинг 1.3).

Листинг 1.3. Макет виджета для элементов списка `android.widget.ListView` рассматриваемого примера

```
<?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:id="@+id/text1"
    android:gravity="center_horizontal"
    android:textSize="12pt"
```

```
    android:textColor="#003366"  
>  
  
</TextView>
```

Модифицируем xml файл /res/layout/activity_main.xml макета Активности приложения, добавив в макет еще один виджет **android.widget.ListView** (см. Листинг 1.4).

Листинг 1.4. В Макет Активности рассматриваемого примера добавлен еще один виджет android.widget.ListView

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:orientation="vertical"  
    tools:context="com.itstep.myapp.MainActivity">  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Список месяцев"  
    android:textSize="14pt"  
    android:layout_gravity="center_horizontal" />  
  
<ListView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:id="@+id/lvMonth"  
    android:layout_weight="1" >  
</ListView>
```

```

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

<ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/lvDaysOfWeek"
    android:layout_weight="1"
    android:background="#E0E0E0">
</ListView>
</LinearLayout>

```

Следующим действием будет создание Адаптера Данных для второго списка `android.widget.ListView` в котором будут отображаться названия дней недели. Этому Адаптеру Данных и будет назначен вновь созданный макет `/res/layout/my_listview_item.xml` для отображения элементов списка. Для создания Адаптера в метод `onCreate` класса Активности необходимо дописать код, показанный в Листинге 1.5.

Листинг 1.5. Создание Адаптера Данных для второго списка и назначение этому Адаптеру собственного макета для элементов списка

```

//---Получение ссылки на виджет ListView который
//---будет содержать дни недели-----
ListView lvDaysOfweek = (ListView)
    this.findViewById(R.id.lvDayOfWeeks);

//---Создание Адаптера данных для второго
//---списка android.widget.ListView-----
String[] arrDaysOfWeek =

```

```

{
    "Понедельник", "Вторник", "Среда",
    "Четверг", "Пятница", "Суббота",
    "Воскресенье"
};

ArrayAdapter<String> lvAdapter2 =
    new ArrayAdapter<>(this,
        R.layout.my_listview_item, arrDaysOfWeek);

//--Назначение Адаптера Данных списку ListView---
//--с днями недели-----
lvDaysOfweek.setAdapter(lvAdapter2);

```

Внешний вид примера из Листингов 1.4 и 1.5 изображен на Рис. 1.3.

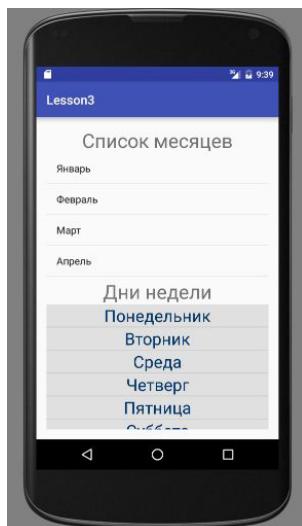


Рис. 1.3. Списки android.widget.ListView со стандартным макетом android.R.layout.simple_list_item_1 (вверху) и с созданным в примере макетом R.layout_my_listview_item (внизу)

В завершении примера рассмотрим обработку события выбора элемента списка `android.widget.listView` пользователем. Для этой цели предназначено событие **OnItemClickListener** которое срабатывает когда пользователь кликнул или прикоснулся к элементу списка. Событие выбора описывается интерфейсом `android.widget.AdapterView.OnItemClickListener` ([http://developer.android.com/reference/android/widget\(AdapterView.OnItemClickListener\).html](http://developer.android.com/reference/android/widget(AdapterView.OnItemClickListener).html)). В этом интерфейсе объявлен всего один метод:

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

Метод принимает ссылку на виджет `android.widget.ListView` (**parent**), ссылку на выбранный пользователем виджет элемента списка (**view**) и индекс выбранного элемента (**position**) в наборе данных Адаптера Данных. Последний параметр **id** это идентификатор выбранного виджета.

Добавим к списку `android.widget.ListView` который отображает список месяцев, обработчик события клика по элементу списка. Для этого в методе **onCreate** класса Активности допишем следующий код (Листинг 1.6):

Листинг 1.6. Добавление обработчика события клика по элементу списка `android.widget.ListView`.

```
//--Назначение обработчика события выбора
//--элемента для списка месяцев-----
lvMonth.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
    @Override
```

```

public void onItemClick(AdapterView<?> parent,
                      View view, int position, long id)
{
    Toast.makeText(MainActivity.this,
                  String.valueOf(position) + " = " +
                  parent.getAdapter().getItem(position),
                  Toast.LENGTH_SHORT).show();
}
);

```

Теперь, при клике на любом названии месяца из списка, будет появляться тост (**Toast**) с индексом выбранного элемента и названием месяца, которое хранится в этом элементе.

Исходные коды рассмотренного примера находятся в модуле «app» файлов исходного кода, прилагаемых к данному уроку.

1.1. Виджет android.widget.ListView одиночного выбора

Существует возможность создавать виджет **android.widget.ListView** с возможностью выбора элементов. Для этой цели необходимо, во-первых, для объекта **android.widget.ListView** вызвать метод:

```
public void setChoiceMode (int choiceMode);
```

который принимает в качестве параметра тип выбора для списка.

Принимаемые значения могут быть следующими:

- **CHOICE_MODE_NONE** (нет никакого выбора),
- **CHOICE_MODE_SINGLE** (список одиночного выбора),

- **CHOICE_MODE_MULTIPLE** (список множественного выбора).

Во-вторых, необходимо Адаптеру Данных передать в качестве макета виджета для элементов списка стандартный макет **android.R.layout.simple_list_item_single_choice** (не обязательно использовать именно этот макет — можно создать и свой макет с радио-кнопкой и текстовым полем).

В Листинге 1.7 приведен программный код Java, демонстрирующий создание списка **android.widget.ListView** одиночного выбора в обработчике события **onCreate** класса Активности.

Листинг 1.7. Создание списка android.widget.ListView одиночного выбора

```
//--Создание списка android.widget.ListView
//--одиночного выбора-----
ListView lvMonth = (ListView) this.
    findViewById(R.id.lvMonth);

String[] arrMonth =
{
    "Январь", "Февраль", "Март", "Апрель",
    "Май", "Июнь", "Июль", "Август", "Сентябрь",
    "Октябрь", "Ноябрь", "Декабрь"
};

lvMonth.setChoiceMode(ListView.CHOICE_MODE_SINGLE);

ArrayAdapter<String> lvAdapter1 =
    new ArrayAdapter<>(this,
        android.R.layout.simple_list_
        item_single_choice, arrMonth);

lvMonth.setAdapter(lvAdapter1);
```

Внешний вид виджета **android.widget.ListView** одиночного выбора изображен на Рис. 1.4.

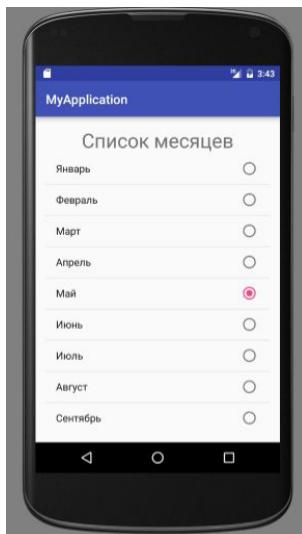


Рис. 1.4. Список android.widget.ListView одиночного выбора

Для того чтобы узнать какой элемент списка выбран пользователем, необходимо воспользоваться методом класса **android.widget.ListView**:

```
public int getCheckedItemPosition ();
```

Метод возвращает индекс выбранного элемента в наборе данных Адаптера Данных или — 1 (**ListView.INVALID_POSITION**) если ни один элемент не выбран.

Давайте добавим в макет Активности /res/layout/activity_main.xml кнопку **android.widget.Button** чтобы при клике на эту кнопку приложение выводило в Тосте информацию о выбранном элементе списка. Макет Активности для текущего примера приведен в Листинге 1.8.

Листинг 1.8. Макет Активности для примера списка android.widget.ListView одиночного выбора

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android=
        "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="itstep.com.myapp2.MainActivity">

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

    <ListView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:id="@+id/lvMonth"
        android:layout_weight="1" >
    </ListView>

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Показать"
        android:onClick="btnClick" />
</LinearLayout>
```

Как видно из Листинга 1.8, у кнопки присутствует обработчик события клика **btnClick**, в котором приложение определит выбранный (отмеченный) элемент списка

и отобразит информацию о нем в Тосте. В Листинге 1.9 приведен код метода обработчика **btnClick**.

Листинг 1.9. Вывод информации о выбранном (отмеченном) элементе списка android.widget.ListView

```
public void btnClick(View v)
{
    ListView lvMonth = (ListView) this.
                        findViewById(R.id.lvMonth);
    int index = lvMonth.getCheckedItemPosition();
    if (index != -1)
    {
        Toast.makeText(this, lvMonth.getAdapter().
                        getItem(index).toString(),
                        Toast.LENGTH_SHORT).show();
    }
    else
    {
        Toast.makeText(this, "Ничего не выбрано",
                        Toast.LENGTH_SHORT).show();
    }
}
```

Весь исходный код примера находится в модуле «app2» среди файлов с исходными кодами, которые прилагаются к данному уроку.

1.2. Виджет android.widget.ListView множественного выбора

Также имеется возможность создавать списки **android.widget.ListView** с множественным выбором элементов. Для этого, во-первых, необходимо с помощью метода класса **android.widget.ListView**:

```
public void setChoiceMode (int choiceMode);
```

установить режим множественного выбора, передав в метод значение `ListView.CHOICE_MODE_MULTIPLE`.

Во-вторых, необходимо в Адаптер Данных передать стандартный макет `android.R.layout.simple_list_item_multiple_choice` (или собственный макет с чекбоксом и текстовым полем).

В Листинге 1.10 приведен программный код Java, демонстрирующий создание списка `android.widget.ListView` множественного выбора в обработчике события `onCreate` класса Активности.

Листинг 1.10. Создание списка android.widget.ListView множественного выбора

```
String[] arrMonth =
{
    "Январь", "Февраль", "Март", "Апрель",
    "Май", "Июнь", "Июль", "Август", "Сентябрь",
    "Октябрь", "Ноябрь", "Декабрь"
};

ListView lvMonth = (ListView) this.
    findViewById(R.id.lvMonth);
lvMonth.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
ArrayAdapter<String>
    lvAdapter1 = new ArrayAdapter<>(
        this, android.R.layout.
        simple_list_item_multiple_choice, arrMonth);

lvMonth.setAdapter(lvAdapter1);
```

Внешний вид списка `android.widget.ListView` множественного выбора изображен на Рис. 1.5.

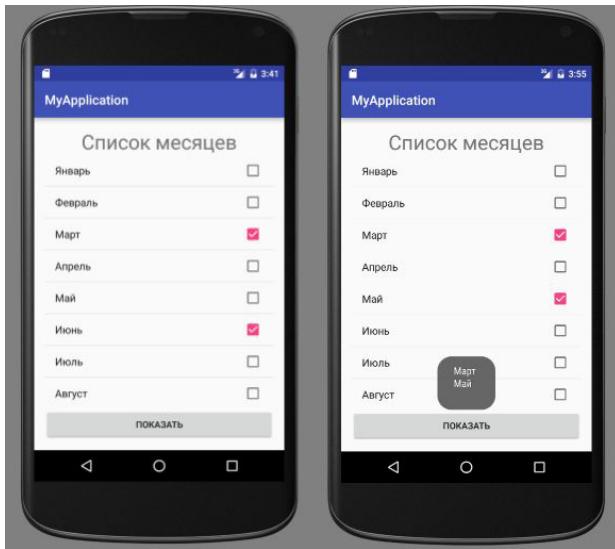


Рис. 1.5. Внешний вид списка android.widget.ListView множественного выбора

Для того чтобы определить какие элементы выбраны, необходимо воспользоваться методом:

```
SparseBooleanArray getCheckedItemPositions();
```

который возвращает коллекцию `android.util.SparseBooleanArray` (<http://developer.android.com/reference/android/util/SparseBooleanArray.html>), которая содержит набор пар «ключ-значение» в котором ключом является индекс элемента в списке `android.widget.ListView`, а значением является логическое true / false которое и сообщает о том, что элемент выбран (true) или не выбран (false).

Пример программного кода, показывающий, как определять выбранные (отмеченные) элементы списка `android.widget.ListView` множественного выбора, показан

в Листинге 1.11 (Код находится в методе обработчике события клика по кнопке «Показать» **btnClick** рассматриваемого примера).

Листинг 1.11. Определение отмеченных элементов списка android.widget.ListView множественного выбора

```
ListView lvMonth = (ListView) this.  
        findViewById(R.id.lvMonth);  
String msg = "";  
  
SparseBooleanArray sbArray = lvMonth.  
        getCheckedItemPositions();  
for (int i = 0; i < sbArray.size(); i++)  
{  
    int key = sbArray.keyAt(i);  
    if (sbArray.get(key))  
    {  
        msg += lvMonth.getAdapter().getItem(key).  
                toString() + "\n";  
    }  
}  
if (msg.isEmpty())  
{  
    Toast.makeText(this, "Ничего не выбрано",  
            Toast.LENGTH_SHORT).show();  
}  
else  
{  
    Toast.makeText(this, msg, Toast.LENGTH_LONG).show();  
}
```

Весь исходный код примера находится в модуле «app2» среди файлов с исходными кодами, которые прилагаются к данному уроку.

2. События прокрутки для виджета android.widget.ListView

Для обработки событий прокрутки элементов списка `android.widget.ListView` необходимо реализовать интерфейс `android.widget.AbsListView.OnScrollListener` (<http://developer.android.com/reference/android/widget/AbsListView.OnScrollListener.html>). В этом интерфейсе объявлено всего два метода. *Первый метод:*

```
public void onScrollStateChanged (AbsListView view,  
                                int scrollState);
```

Вызывается во время прокрутки списка. Первый принимаемый параметр (`view`) — это ссылка на список, который является источником события, второй параметр (`scrollState`) содержит состояние прокрутки, которое задается одним из следующих значений (статические константы `android.widget.AbsListView.OnScrollListener`):

- `SCROLL_STATE_IDLE = 0` список закончил прокрутку
- `SCROLL_STATE_TOUCH_SCROLL = 1` пользователь прокручивает список и его палец находится на экране.
- `SCROLL_STATE_FLING = 2` список «катнули», т.е. при прокрутке отпустили палец и прокрутка дальше идет «по инерции».

Второй метод:

```
public void onScroll (AbsListView view,  
                     int firstVisibleItem,  
                     int visibleItemCount,  
                     int totalItemCount);
```

Вызывается во время прокрутки списка. Принимаемые параметры:

- **AbsListView view** — ссылка на виджет **android.widget.ListView**, являющийся источником события.
- **int firstVisibleItem** — индекс первого видимого элемента на экране устройства.
- **int visibleItemCount** — количество видимых элементов списка на экране устройства.
- **int totalItemCount** — общее количество элементов в Адаптере Данных который назначен данному списку.

Для того, чтобы виджету **android.widget.ListView** назначить обработку событий прокрутки, необходимо воспользоваться методом:

```
void    setOnScrollListener(AbsListView.  
                           OnScrollListener l);
```

Этот метод принимает ссылку на объект, реализующий только что рассмотренный нами интерфейс **android.widget.AbsListView.OnScrollListener**. В Листинге 2.1 приведен программный код примера из модуля «app2» обрабатывающий события прокрутки для списка. Назначение обработчика события с помощью вызова метода **setOnScrollListener** осуществляется в методе **onCreate** класса Активности.

Листинг 2.1. Пример обработки событий прокрутки для списка android.widget.ListView

```
//--Обработка событий прокрутки списка
//--android.widget.ListView-----
lvMonth.setOnScrollListener(new AbsListView.
    OnScrollListener()
{
    @Override
    public void onScrollStateChanged(AbsListView
        view, int scrollState)
    {
        /*
         * SCROLL_STATE_IDLE = 0,
         * список закончил прокрутку
         * SCROLL_STATE_TOUCH_SCROLL = 1,
         * пользователь прокручивает список
         * и его палец находится на экране
         * SCROLL_STATE_FLING = 2, список «катнули»,
         * т.е. при прокрутке отпустили палец и
         * прокрутка дальше идет «по инерции»
         */
        Log.d(TAG, "===== OnScrollStateChanged : " +
            + scrollState);
    }

    @Override
    public void onScroll(AbsListView view,
        int firstVisibleItem, int visibleItemCount,
        int totalItemCount)
    {
        Log.d(TAG, "===== OnScroll : " +
            firstVisibleItem + ":" +
            visibleItemCount + ":" +
            totalItemCount);
    }
});
```

3. Создание custom-виджетов для элементов списка android.widget.ListView

В данном разделе мы рассмотрим на примере создание виджетов для элементов списка `android.widget.ListView`, которые бы имели отличающийся внешний вид от предлагаемых виджетов по умолчанию, которые мы рассмотрели выше (вспомним, что к виджетам по умолчанию относятся следующие рассмотренные нами виджеты: `android.R.layout.simple_list_item_1` для обычного списка, `android.R.layout.simple_list_item_single_choice` для списка одиночного

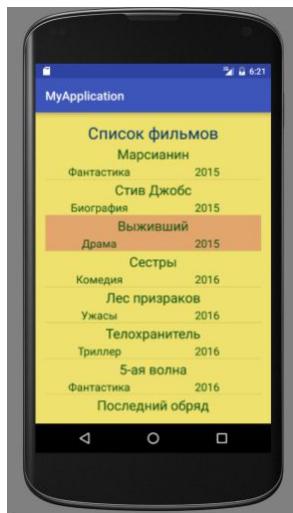


Рис. 3.1. Внешний вид списка `android.widget.ListView` рассматриваемого примера

выбора, `android.R.layout.simple_list_item_multiple_choice` для списка множественного выбора).

В рассматриваемом примере будет создаваться custom-виджет для списка `android.widget.ListView`, в котором можно будет выбрать только один элемент. При этом, выбранный элемент будет иметь специальный цвет фона и не будет иметь радио-кнопки, как в стандартном внешнем виде для списка одиночного выбора. Внешний вид списка, рассматриваемого в данном разделе, изображен на Рис. 3.1.

Xml верстка макета Активности рассматриваемого примера приведена в Листинге 3.1.

Листинг 3.1. Xml верстка макета Активности рассматриваемого примера, внешний вид которого изображен на Рис. 3.1

```
<LinearLayout  
    xmlns:android=  
        "http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
  
    android:orientation="vertical"  
    android:background="#EDE275"  
  
    tools:context="itstep.com.myapp3.MainActivity">  
  
    <TextView  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:layout_gravity="center_horizontal"  
        android:textSize="12pt"
```

```
    android:textColor="#003366"
    android:text="Список фильмов" />

<ListView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/lvFilms"
    android:background="#EDE275">
</ListView>
</LinearLayout>
```

Как видно из Рис. 3.1, список `android.widget.ListView` отображает список фильмов. Каждый элемент списка — Фильм — является объектом класса `Film`, который состоит из трех полей: Название, Жанр и Год Выпуска. Класс `Film` приведен в Листинге 3.2.

Листинг 3.2. Класс `Film`, объекты которого размещены в наборе данных Адаптера Данных и отображаются в списке `android.widget.ListView` рассматриваемого примера

```
/***
 * Class Film - Содержит информацию о фильме.
 * Предназначен быть одним элементом в наборе
 * данных Адаптера Данных для списка
 * android.widget.ListView
 * -----
 */
class Film
{
    //--Class constants-----
    /**
     * 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;  
}  
  
@Override  
public String toString()  
{  
    return "Title : " + this.title + ", Genre : " +  
        this.genre + ", Year : " + this.year;  
}  
}
```

Как видно из Листинга 3.2, объект класса **Film** содержит три значения: Название фильма (поле **title**), Жанр фильма (**genre**) и Год выпуска фильма (**year**). А это значит, что для отображения элементов списка, состоящих из нескольких значений, лучше всего воспользоваться Адаптером Данных `android.widget.SimpleAdapter` (<http://developer.android.com/reference/android/widget/SimpleAdapter.html>). Однако, в данном примере мы поступим по-другому и откажемся от традиционного подхода. Во-первых,

пример с использованием Адаптера Данных `android.widget.SimpleAdapter` рассматривался нами в предыдущем уроке (применение его для `android.widget.ListView` ничем не отличается от применения к другим спискам), а во-вторых, специфика данного раздела урока заключается в том, что рассматривается создание custom-виджетов, т. е. виджетов, являющихся уникальными и непохожими. Так вот, в рассматриваемом примере будет использоваться Адаптер Данных `android.widget.ArrayAdapter<T>` (<http://developer.android.com/reference/android/widget/ArrayAdapter.html>). Этот Адаптер предназначен для отображения элементов списка, содержащих одно значение, а в нашем примере элемент списка содержит три значения. Решение заключается в следующем: Во-первых, Адаптер Данных `android.widget.ArrayAdapter<T>` создается с помощью конструктора:

```
ArrayAdapter(Context context, int resource,
            int textViewResourceId, List<T> objects);
```

который принимает кроме идентификатора макета виджета (`int resource`) для элементов списка еще и идентификатор виджета (`int textViewResourceId`) в макете, в который и должно размещаться значение элемента списка. Мы в этот конструктор в параметр `textViewResourceId` передадим идентификатор любого виджета из макета, представляющего наш элемент списка (Листинг 3.3). А как же остальные виджеты макета — как мы добьемся того, чтобы в них отображались значения для фильма? А для этого нужно сделать «во-вторых».

И во-вторых, необходимо переопределить метод `getView` Адаптера Данных, чтобы вмешаться в процесс

формирования виджетов для нашего списка `android.widget.ListView` и расставить значения полей для каждого Фильма по своим местам (то есть по своим виджетам) в макете. Кроме этого, переопределенный метод `getView` будет использоваться и для подсветки виджета, являющимся виджетом для выбранного элемента списка. В прошлом уроке рассказывалось о методе `getView`, который есть в каждом классе Адаптеров Данных. Давайте еще раз вспомним об этом методе, перед тем как будем его использовать:

```
public View getView (int position, View convertView,  
ViewGroup parent);
```

Метод предназначен для создания виджета, который будет представлять один элемент списка (`android.widget.ListView`, `android.widget.Spinner` и т. д.). Метод принимает следующие параметры:

- **int position** — индекс элемента в наборе данных Адаптера Данных, для которого необходимо сформировать виджет.
- **View convertView** — ссылка на предыдущий виджет, который использовался для этого или другого элемента списка. Этот параметр может иметь значение `null`.
- **ViewGroup parent** — ссылка на список (родительский виджет), к которому будет прикреплен создаваемый этим методом виджет.

Этот метод можно переопределять для того, чтобы внести изменения в формируемые виджеты для получения необходимых результатов в способе отображения

данных в виджете или во внешнем виде виджета. Это и будет делаться в нашем примере.

Итак, приступим к реализации задуманного решения. Для начала, создадим файл ресурсов с макетом виджета, который будет использоваться Адаптером Данных для формирования виджетов для элементов списка `android.widget.ListView`. Файл макета ресурсов называется `/res/layout/film_item.xml`. Содержимое xml верстки макета виджета приведено в Листинге 3.3.

Листинг 3.3. Xml верстка макета виджета из файла ресурсов `/res/layout/film_item.xml` для формирования элементов списка рассматриваемого примера

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="10pt"
        android:layout_gravity="center_horizontal"
        android:layout_margin="2dp"
        android:id="@+id/tvTitle"
        android:textColor="#0F5119" />

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

```

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="8pt"
    android:id="@+id/tvGenre"
    android:gravity="center_horizontal"
    android:layout_weight="1"
    android:textColor="#0F5119" />

<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="8pt"
    android:id="@+id/tvYear"
    android:gravity="center_horizontal"
    android:layout_weight="1"
    android:textColor="#0F5119" />
</LinearLayout>
</LinearLayout>

```

Внешний вид виджета, который будет формироваться для каждого элемента списка на основе макета, представленного в Листинге 3.3 изображен на Рис. 3.2.

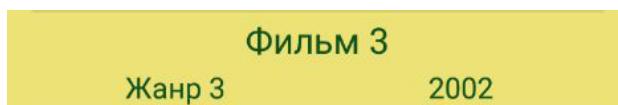


Рис. 3.2. Внешний вид виджета из Листинга 3.3

Далее, создадим список фильмов и поместим их в Адаптер Данных. Для Адаптера Данных переопределим метод **getView**, в котором «расставим» значения полей отображаемых объектов класса **Film** по своим местам в виджете из Листинга 3.3 (в виджет с идентификатором **tvTitle** будет размещаться Название фильма, в виджет

с идентификатором **tvGenre** — жанр фильма, и в виджет с идентификатором **tvYear** — год выпуска фильма). Этот программный код приведен в Листинге 3.4.

Листинг 3.4. Заполнение Адаптера Данных списков фильмов и переопределение метода `getView` Адаптера Данных

```
public class MainActivity extends AppCompatActivity
{
    //--- Class members -----
    /**
     * Список фильмов
     */
    private ListView lvFilms;

    /**
     * Цвет фона не выбранного элемента списка
     */
    private int nrmlColor = Color.rgb(0xED, 0xE2, 0x75);

    /**
     * Цвет фона выбранного элемента списка
     */
    private int slctColor = Color.rgb(0xE2, 0xA7, 0x6F);

    /**
     * Индекс выбранного элемента списка
     */
    private int curItem = -1;

    /**
     * Ссылка на виджет текущего выбранного элемента списка
     */
    private View curView = null;
    //--- Class methods -----
    @Override
    protected void onCreate(Bundle savedInstanceState)
```

```
{  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    //--- Инициализация поля lvFilms -----  
    this.lvFilms = (ListView) this.findViewById(R.  
        id.lvFilms);  
  
    //--- Создание коллекции фильмов -----  
    ArrayList<Film> films = new ArrayList<>();  
    for (int i = 0; i < 50; i++)  
    {  
        films.add(new Film("Фильм " + (i + 1),  
            "Жанр " + (i + 1), 2000 + i % 15));  
    }  
  
    //--- Создание адаптера данных и назначение его  
    //--- списку lvFilms -----  
    ArrayAdapter<Film> adapter =  
        new ArrayAdapter<Film>(this,  
            R.layout.film_item, R.id.tvTitle, films)  
    {  
        @Override  
        public View getView(int position,  
            View convertView, ViewGroup parent)  
        {  
            View view = super.getView(position,  
                convertView, parent);  
  
            //--- Получение ссылки на объект Film, который  
            //--- отображается в этом виджете -----  
            Film f = this.getItem(position);  
            //--- Получение ссылок на виджеты TextView для  
            //--- отображения в них информации о фильме -----  
            TextView tvTitle = (TextView) view.  
                findViewById(R.id.tvTitle);  
    }
```

```
        TextView tvGenre = (TextView) view.  
            findViewById(R.id.tvGenre);  
        TextView tvYear = (TextView) view.  
            findViewById(R.id.tvYear);  
  
        //--- Запись в виджеты TextView информации  
        //--- о текущем фильме -----  
        tvTitle.setText(F.title);  
        tvGenre.setText(F.genre);  
        tvYear.setText(String.valueOf(F.year));  
        return view;  
    }  
};  
  
//--- Назначение Адаптера данных списку  
//--- android.widget.ListView -----  
this.lvFilms.setAdapter(adapter);  
}  
}
```

В примере Листинга 3.4 коллекция фильмов `ArrayList<Film>` заполнялась псевдослучайными фильмами с псевдослучайными жанрами и годами выпуска. Для рассматриваемого примера важно, чтобы список был достаточно большим — от 50 элементов и более. Чуть позже вы поймете почему. Обратите внимание, как в переопределенном методе `getView` осуществляется расстановка значений Фильма по соответствующим замыслу виджетам в макете: переопределенный метод `getView` вызывает родительский метод `getView`, который и создает виджет на основе макета виджета из файла ресурсов. Однако, созданный родительским методом виджет, удовлетворяет нас внешним видом,

но не удовлетворяет нас тем, как расставлены значения полей Фильма. Родительский метод размещает строковое представление объекта **Film** в виджете **tvTitle**, потому что Адаптер Данных **android.widget.ArrayAdapter<T>** предназначен для элементов списка, содержащих одно строковое значение, которое Адаптер Данных **android.widget.ArrayAdapter<T>** получает с помощью вызова метода **toString()** для каждого отображаемого объекта из своего набора данных. И это строковое значение для Фильма отображается в виджете **tvTitle** макета, потому этот виджет был указан при создании Адаптера Данных:

```
ArrayAdapter<Film> adapter =
    new ArrayAdapter<Film>(this,
        R.layout.film_item,
        R.id.tvTitle, films)
```

Поэтому после вызова родительского метода **getView**, переопределенный метод **getView** в созданном виджете находит виджеты **tvTitle**, **tvGenre** и **tvYear** и самостоятельно записывает в них значения соответствующих полей из объекта **Film** текущего элемента (см. Листинг 3.4). Таким образом, мы получаем результат, по внешнему виду идентичный применению для списка **android.widget.ListView** Адаптера Данных **android.widget.SimpleAdapter**. Внешний вид работы переопределенного метода **getView** из Листинга 3.4 изображен на Рис. 3.3.

Отдельно необходимо обратить внимание на очень важный факт последствия такого использования метода **getView**, а именно — мы получаем возможность

прямой работы с объектами (в нашем случае — объектами класса `Film`), а не с их строковыми копиями, как если бы использовался Адаптер Данных `android.widget.SimpleAdapter`. Потому что удобнее программно модифицировать данные объектов в наборе данных Адаптера Данных `android.widget.ArrayAdapter<T>`, чем вручную синхронизировать строковые значения полей из `android.widget.SimpleAdapter` с реальной коллекцией объектов типа `ArrayList<T>`.



Рис. 3.3. Внешний вид работы метода `getView` из Листинга 3.4

Но это еще не все! Наш пример должен выполнять подсветку выбранного с помощью клика (нажатия) элемента списка. Здесь поступим следующим способом: для виджета `android.widget.ListView` зададим обработчик клика по элементу списка `android.widget.AdapterView.OnItemClickListener` (<http://developer.android.com/reference/>)

[android/widget/AdapterView.OnItemClickListener.html](#)). В интерфейсе android.widget.AdapterView.OnItemClickListener объявлен всего один метод:

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

который принимает ссылку на список android.widget.ListView (**parent**), в котором произошел клик (нажатие) по элементу, виджет элемента списка (**view**), по которому кликнули, и индекс элемента в наборе данных Адаптера Данных (**position**).

Так вот, при клике на элемент списка android.widget.ListView, наш пример будет подсвечивать выбранный пользователем виджет специальным цветом, при этом ранее выбранный виджет элемента списка (если такой был) будет подсвечиваться цветом фона списка, т. е. визуально перестанет выглядеть как выбранный. С этой целью в классе Активности нашего примера объявлены следующие поля (Листинг 3.5):

Листинг 3.5. Поля класса Активности, используемые для подсветки выбранных элементов списка

```
/**
 * Цвет фона не выбранного элемента списка
 */
private int nrmlColor = Color.rgb(0xED, 0xE2, 0x75);

/**
 * Цвет фона выбранного элемента списка
 */
private int slctColor = Color.rgb(0xE2, 0xA7, 0x6F);
```

```
/**  
 * Индекс выбранного элемента списка  
 */  
private int curItem = -1;  
  
/**  
 * Ссылка на виджет текущего выбранного  
 * элемента списка  
 */  
private View curView = null;
```

Если с полями **nrmlColor** (цвет не выбранного элемента списка) и **slctColor** (цвет выбранного элемента списка) все понятно, то с остальными полями Листинга 3.5 нужно познакомиться отдельно:

- **int curItem** — в это поле будет записываться индекс выбранного пользователем элемента списка, чтобы в дальнейшем можно было с этим полем выполнять какие-либо действия (например, удаление или редактирование).
- **View curView** — в это поле будем запоминать текущий выбранный виджет, чтобы когда пользователь выберет другой элемент списка, вернуть предыдущему выбранному виджету (на который будет ссылаться это поле) основной цвет фона.

В методе **onCreate** класса Активности допишем программный код, назначающий обработчик события **OnItemClickListener** (см. Листинг 3.6):

Листинг 3.6. Обработчик события OnItemClickListener для списка android.widget.ListView рассматриваемого примера

```

//-- Назначение обработчика события клика
//-- по элементу списка -----
this.lvFilms.setOnItemClickListener(new
        AdapterView.OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent,
            View view, int position, long id)
    {
//-- Снимаем выделение с предыдущего выделенного
//-- элемента -----
if (MainActivity.this.curItem != -1)
{
    MainActivity.this.curView.
        setBackgroundColor(MainActivity.
        this.nrmlColor);
}
//-- Устанавливаем выделение на текущий элемент
//-- списка -----
MainActivity.this.curItem      = position;
MainActivity.this.curView       = view;
MainActivity.this.curView.
        setBackgroundColor(MainActivity.
        this.slctColor);
}
});

```

Как видно из Листинга 3.6, при клике на элемент списка происходит снятие выделения с предыдущего выделенного виджета (если таковой был) и установление выделения в виде специального цвета фона новому выделенному элементу.

Внешний вид работы примера изображен на Рис. 3.4.



Рис. 3.4. Внешний вид работы примера из Листинга 3.6

Однако, не все так просто, как оказалось! Да — выделение элемента происходит замечательно, но при прокрутке списка, когда выделенный элемент уезжает за границы экрана, на экран выезжает совершенно другой элемент отмеченный как выбранный! И так повторяется постоянно — как только выбранный или «лжевыбранный» элемент при прокрутке выезжает за границы экрана, на экран выезжает следующий элемент, который отмечен как выбранный. Этот дефект изображен на Рис. 3.4. В чем же дело? Дело в том, что мобильные приложения должны очень строго экономить ресурсы, поскольку ресурсы мобильных устройств скромнее, чем ресурсы персональных компьютеров. Именно поэтому для нашего списка не создается количество виджетов, равное количеству элементов

списка. Виджеты для элементов списка `android.widget.ListView` создаются в количестве, равном количеству видимых элементов + 2. Почему плюс два дополнительных виджета? — Потому что один виджет предназначен для элемента списка который появится снизу при прокрутке к концу списка, и другой виджет — для элемента списка, который появится сверху при прокрутке к началу списка. Данное поведение на самом деле является реализацией паттерна PlaceHolder (он же Proxy, Заместитель). Схема работы наблюдаемого на нашем примере поведения списка изображена на Рис. 3.5. Это объясняет наблюдаемый факт, что виджет, отмеченный как выбранный специальным цветом фона, появляется на экране устройства с четким периодом. То есть Адаптер Данных всего лишь заполняет значения виджетов, являющихся элементами списка, по

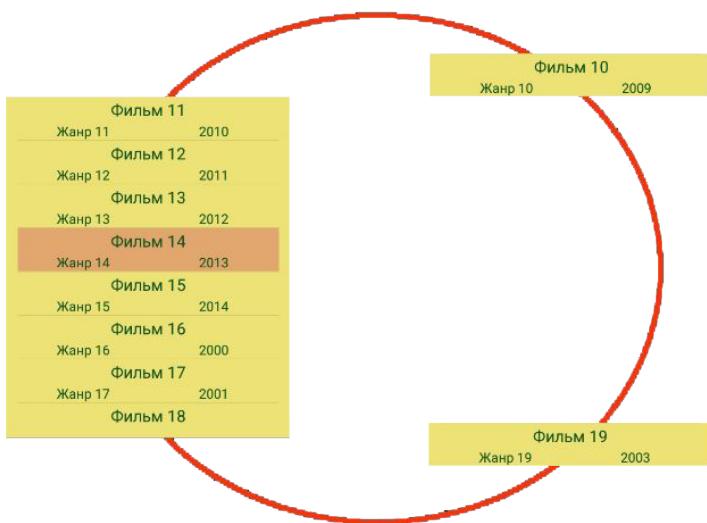


Рис. 3.5. Схема формирования и подстановки виджетов для списка `android.widget.ListView`

принципу карусели, не меняя их внешнего вида. Следовательно, решения из Листинга 3.6 не достаточно, для решения поставленной задачи.

Необходимо учесть механизм использования виджетов, изображенный на Рис. 3.5, для этого в переопределенном методе `getView` внесем дополнительную проверку на то, какой сейчас формируется виджет — для выбранного или для не выбранного элемента списка и подсветим его соответствующим образом. Решение представлено в Листинге 3.7.

Листинг 3.7. Дополнительное решение для формирования custom-виджетов

```
//-- Создание адаптера данных и назначение
//-- его списку lvFilms -----
ArrayAdapter<Film> adapter = new ArrayAdapter<Film>(
        this, R.layout.film_item,
        R.id.tvTitle, films)
{
    @Override
    public View getView(int position, View convertView,
                        ViewGroup parent)
    {
        View view = super.getView(position,
                                   convertView,
                                   parent);

//-- Получение ссылки на объект Film, который
//-- отображается в этом виджете -----
        Film F = this.getItem(position);

//-- Получение ссылок на виджеты TextView
//-- для отображения в них информации о фильме ---
        TextView tvTitle = (TextView) view.
                findViewById(R.id.tvTitle);
```

```
TextView tvGenre = (TextView) view.  
        findViewById(R.id.tvGenre);  
TextView tvYear = (TextView) view.  
        findViewById(R.id.tvYear);  
  
//-- Запись в виджеты TextView информации  
//-- о текущем фильме -----  
    tvTitle.setText(F.title);  
    tvGenre.setText(F.genre);  
    tvYear.setText(String.valueOf(F.year));  
  
//-- Подсветка отмеченного элемента списка -----  
    if (position == MainActivity.this.curItem)  
    {  
        view.setBackgroundColor(MainActivity.  
                            this.slctColor);  
        MainActivity.this.curView = view;  
    }  
    else  
    {  
        view.setBackgroundColor(MainActivity.  
                            this.nrmlColor);  
    }  
    return view;  
}  
};
```

Теперь с формированием custom-виджетов для списка **android.widget.ListView** получилось все, как и задумывалось изначально. Весь исходный код примера находится в модуле «app3» среди файлов исходного кода, прилагаемых к данному уроку.

4. Диалоговые окна. Классы AlertDialog, AlertDialog.Builder

Диалоговое окно — это небольшое окно, которое предоставляет пользователю возможность принять решение или ввести дополнительные данные в приложение. Диалоговые окна, как правило, не занимают всей области экрана и работают в модальном режиме, не давая пользователю закрыть окно без выполнения необходимых действий.

Для реализации функциональности диалоговых окон существует класс `android.app.Dialog` (<http://developer.android.com/reference/android/app/Dialog.html>), который является базовым классом для диалогов. Однако, как правило приходится работать с объектами производных от этого класса классов:

- `android.app.AlertDialog` — представляет диалоговое окно с заголовком, возможностью размещения разных виджетов внутри окна и с возможностью размещения до трех кнопок.
- `android.app.DatePickerDialog` — диалоговое окно для выбора даты.
- `android.app.TimePickerDialog` — диалоговое окно для выбора времени.

В следующих разделах мы рассмотрим применение этих классов более подробно.

4.1. Классы AlertDialog, AlertDialog.Builder

Класс `android.app.AlertDialog` (<http://developer.android.com/reference/android/app/AlertDialog.html>) решает большинство задач по работе с диалоговыми окнами. Иерархия класса выглядит следующим образом:

```
java.lang.Object
|
+-- android.app.Dialog
|
+-- android.app.AlertDialog
```

Чтобы создать объект экземпляр класса `android.app.AlertDialog` необходимо воспользоваться объектом `android.app.AlertDialog.Builder` (<http://developer.android.com/reference/android/app/AlertDialog.Builder.html>), который предназначен для формирования внешнего диалоговых окон `android.app.AlertDialog`.

Последовательность шагов для создания диалогового окна `android.app.AlertDialog` следующая. *Во-первых*, необходимо создать объект `android.app.AlertDialog.Builder`. Делается это с помощью конструктора:

```
public AlertDialog.Builder (Context context,
                           int themeResId);
```

который принимает ссылку на объект контекста приложения и идентификатор темы для Диалогового окна. Если передать в качестве темы значение 0, то будет использована тема по умолчанию для устройства. Так же можно воспользоваться конструктором, принимающим один параметр — ссылку на контекст приложения — в этом

случае опять же Диалоговому окну будет назначена тема по умолчанию. Идентификаторы тем необходимо брать из класса `android.R.style`. Вот пример нескольких тем:

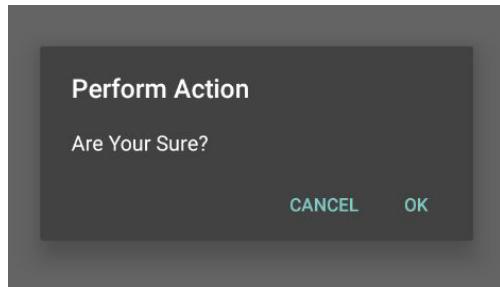


Рис. 4.1. Тема `android.R.style.Theme_DeviceDefault_Dialog_Alert`

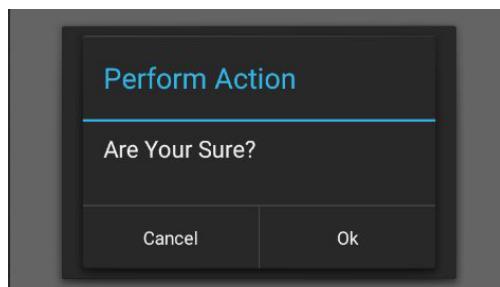


Рис. 4.2. Тема `android.R.style.Theme_Holo_Dialog`

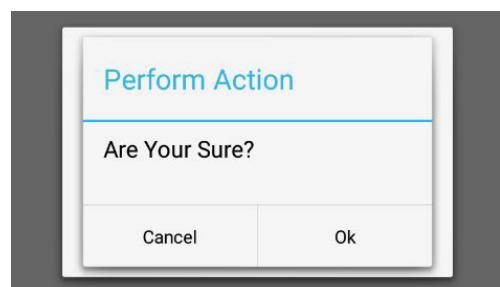


Рис. 4.3. Тема `android.R.style.Theme_Holo_Light_Dialog`

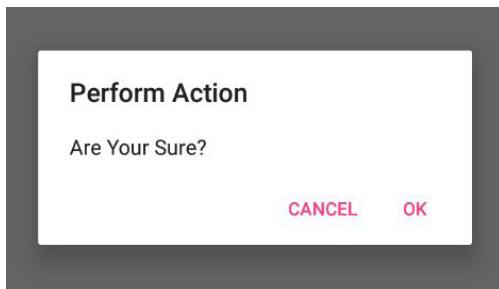


Рис. 4.4. Тема android.R.style.ThemeOverlay_Material_Dialog_Alert

Это неполный список тем для Диалоговых окон. С остальными темами можете познакомиться самостоятельно чтобы выбирать для своих приложений наиболее подходящие вам темы.

Вторым шагом является задание текста для заголовка диалогового окна и текста для содержимого диалогового окна. Установить текст заголовка можно с помощью метода класса `android.app.AlertDialog.Builder`:

```
public AlertDialog.Builder setTitle (CharSequence title);
```

Этот метод принимает текст (параметр `title`) для заголовка Диалогового окна `android.app.AlertDialog`. Как вариант, существует перегруженный метод `setTitle`, принимающий идентификатор строки из файла ресурсов (что более удобно для случая перевода приложения на другой язык).

Для того чтобы установить текстовое сообщение для Диалогового окна, используется метод класса `android.app.AlertDialog.Builder`:

```
public AlertDialog.Builder setMessage (CharSequence message);
```

Этот метод принимает текст (параметр **message**) для сообщения, которое будет выводиться в Диалоговом окне. Так же существует перегруженный метод **setMessage** который принимает идентификатор строки из файла ресурсов.

Часто бывает необходимо разместить в Диалоговом окне не текст, а набор виджетов для ввода пользователем некоторой необходимой приложению информации. Для этой цели вместо метода **setMessage** необходимо воспользоваться методом класса **android.app.AlertDialog.Builder**:

```
public AlertDialog.Builder.setView (View view);
```

который принимает ссылку на виджет (как правило, виджет-контейнер) который необходимо разместить в Диалоговом окне.

На третьем шаге необходимо определиться с кнопками, которые планируется отобразить на Диалоговом окне. Всего на Диалоге можно разместить до трех кнопок: позитивная (positive button, как правило, это кнопка «Да» / «OK»), негативная (negative button, как правило, это кнопка «Нет» / «Cancel») и нейтральная (neutral button) кнопки. Делается это с помощью методов класса **android.app.AlertDialog.Builder**:

```
public AlertDialog.Builder setNegativeButton
    (CharSequence text, DialogInterface.
     OnClickListener listener);

public AlertDialog.Builder setNeutralButton
    (CharSequence text, DialogInterface.
     OnClickListener listener);
```

```
public AlertDialog.Builder setPositiveButton  
    (CharSequence text, DialogInterface.  
     OnClickListener listener);
```

Все эти три метода принимают два параметра: первый — это надпись на кнопке (как вариант может быть использован идентификатор строкового ресурса), второй параметр — это обработчик события клика по кнопке, который должен реализовать интерфейс `android.content.DialogInterface.OnClickListener` (<http://developer.android.com/reference/android/content/DialogInterface.OnClickListener.html>).

В интерфейсе `android.content.DialogInterface.OnClickListener` объявлен всего один метод:

```
void onClick(DialogInterface dialog, int which);
```

который принимает ссылку на объект Диалогового окна (`dialog`), который является источником событии клика по кнопке, и числовое значение (`which`), с помощью которого можно определить по какой конкретно кнопке произошел клик (в виде констант `DialogInterface.BUTTON_NEGATIVE`, `DialogInterface.BUTTON_NEUTRAL`, `DialogInterface.BUTTON_POSITIVE`).

На следующем, четвертом шаге, с помощью объекта `android.app.AlertDialog.Builder` создается объект Диалогового окна с помощью метода:

```
public AlertDialog create();
```

И последнее, что необходимо сделать — это отобразить на экране Диалоговое окно с помощью метода класса `android.app.AlertDialog`:

```
public void show();
```

Диалоговое окно, которое отобразится, будет модальным диалоговым окном. Оно будет закрыто, когда пользователь кликнет на какую-то из кнопок или если пользователь кликнет на область экрана устройства, находящуюся за пределами Диалогового окна. В случае, если пользователь кликнет на области за пределами Диалогового окна, такое действие считается как аннулирование «cancelation» работы Диалогового окна. Аннулирование Диалогового окна не равнозначно нажатию на негативную кнопку «negative button»! Аннулирование приведет только к закрытию Диалогового окна и ни один из обработчиков событий нажатия на кнопку не отработает! Для того чтобы лишить пользователя выполнить аннулирование Диалогового окна, существует метод **android.app.AlertDialog**:

```
void setCancelable(boolean cancelable);
```

Если этому методу передать значение `false`, то пользователь не сможет аннулировать Диалоговое окно.

Программный код, реализующий последовательно вышеперечисленные шаги по созданию Диалогового окна **android.app.AlertDialog**, приведен в Листинге 4.1.

Листинг 4.1. Последовательность действий по созданию, отображению Диалогового окна и получение результата действия пользователя при клике на кнопку

```
public void btnClick(View v)
{
    switch (v.getId())
    {
```

```
case R.id.btnExit:  
{  
    //--- Шаг 1. Создание объекта  
    //--- android.app.AlertDialog.Builder -----  
  
    AlertDialog.Builder builder =  
        new AlertDialog.Builder(this,  
            android.R.style.Theme_Holo_Light_Dialog);  
  
    //--- Шаг 2. Формирование заголовка окна  
    //--- и его содержимого -----  
    builder.setMessage("2 * 2 = 4?");  
    builder.setTitle("Ответьте на вопрос");  
  
    //--- Шаг 3. Назначение диалоговому окну кнопок ---  
    builder.setPositiveButton("Да",  
        new DialogInterface.OnClickListener()  
    {  
        public void onClick(DialogInterface dialog, int id)  
        {  
            Toast.makeText(MainActivity.this,  
                "Верно!", Toast.LENGTH_SHORT).show();  
        }  
    });  
  
    builder.setNegativeButton("Нет",  
        new DialogInterface.OnClickListener()  
    {  
        @Override  
        public void onClick(DialogInterface dialog, int which)  
        {  
            Toast.makeText(MainActivity.this,  
                "Не верно!",  
                Toast.LENGTH_SHORT).show();  
        }  
    });  
}
```

```
//--- Шаг 4. Создание объекта диалогового окна
//--- android.app.AlertDialog -----
    AlertDialog dialog = builder.create();

    dialog.setCancelable(false);
        // Отмена аннулирования окна
        // (необязательно)

// -- Шаг 5. Показываем диалог -----
    dialog.show();
}
break;
}
}
```

Результат работы примера из Листинга 4.1 изображен на Рис. 4.5. Создание и отображение окна осуществляется в методе **btnClick** обработчика события клика на кнопку «Ответить на вопрос», которая имеет идентификатор **R.id.btnOne**.

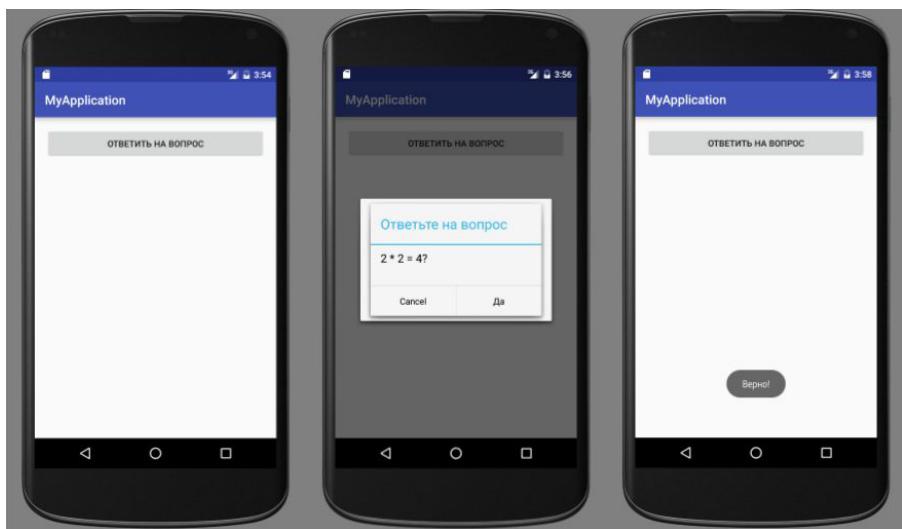


Рис. 4.5. Внешний вид работы примера из Листинга 4.1

4.2. AlertDialog со списком одиночного выбора

В Диалоговом окне `android.app.AlertDialog` можно отображать не только строковые сообщения. В данном разделе мы рассмотрим возможности класса `android.app.AlertDialog` для отображения списка одиночного выбора. Для этой цели вместо метода `setMessage` (метод класса `android.app.AlertDialog.Builder`) необходимо использовать метод:

```
public AlertDialog.Builder setSingleChoiceItems  
    (CharSequence[] items, int checkedItem,  
     DialogInterface.OnClickListener listener);
```

Этот метод устанавливает список одиночного выбора в Диалоговое окно `android.app.AlertDialog`. Метод принимает следующие параметры:

- **CharSequence[] items** — массив строк, содержащий элементы для списка одиночного выбора.
- **int checkedItem** — индекс элемента, который будет установлен как выбранный элемент списка или -1 если ни один элемент списка не выбран.
- **DialogInterface.OnClickListener** — обработчик события выбора элемента списка. Этот интерфейс рассматривался нами в предыдущем разделе этого урока. При клике на элемент списка Диалоговое окно не закрывается! Для закрытия Диалогового окна пользователь должен нажат на одну из кнопок или мы сами должны закрыть Диалоговое окно программно с помощью вызова функции `dismiss()` окна.

Последовательность шагов по созданию Диалогового окна `android.app.AlertDialog` со списком одиночного

выбора такая же как и создание `android.app.AlertDialog` с обычном текстовым сообщением, только, напомним еще раз, что вместо метода `setMessage` необходимо вызвать метод `setSingleChoiceItems`. Пример создания такого Диалогового окна приведен в Листинге 4.3. Для этого, в нашем примере, рассматриваемом в этой главе, добавлена еще одна кнопка с идентификатором `R.id.btnTwo` и надписью «Задать цвет фона». Обработка события клика по этой кнопке также происходит в методе `btnClick`.

В классе Активности `MainActivity` объявлены следующие поля (Листинг 4.2):

Листинг 4.2. Поля для объекта `MainActivity` для реализации рассматриваемого примера

```
public class MainActivity extends AppCompatActivity
{
    //--- Class members -----
    /**
     * Ссылка на главный виджет-контейнер Активности
     */
    private LinearLayout llMain;

    /**
     * Цвет фона для главного виджета-контейнера llMain
     */
    private int clrForMain = Color.rgb(0xFF, 0xFF, 0xFF);

    /**
     * Сюда будем записывать текущий выбираемый
     * пользователем цвет. Если пользователь в Диалоговом
     * окне нажмет "Cancel" то цвет фона виджета llMain
     * вернется в значение, какое указано в clrForMain.
}
```

```

    * Если пользователь в Диалоговом окне нажмет "ОК"
    * то clrForMain = clrTmpMain
    */
private int clrTmpMain;

...
}

```

Программный код из метода **btnClick** приведен в Листинге 4.3.:

Листинг 4.3. Создание android.app.AlertDialog со списком одиночного выбора

```

public void btnClick(View v)
{
    switch (v.getId())
    {
        //-- Обработка события нажатия на кнопку
        //-- "Задать цвет фона" -----
        case R.id.btnTwo :
        {
            //-- Шаг 1. Создание объекта
            //-- android.app.AlertDialog -----
            AlertDialog.Builder builder =
                new AlertDialog.Builder(this,
                    android.R.style.Theme_Holo_Light_Dialog
                ) ;

            //-- Шаг 2. Формирование заголовка окна
            //-- и его содержимого -----
            builder.setTitle("Выберите цвет фона");
            builder.setSingleChoiceItems(
                new String[] {"Красный", "Желтый",
                    "Зеленый", "Белый"}, -1,
                new DialogInterface.OnClickListener()

```

```
{  
    @Override  
    public void onClick(DialogInterface dialog, int which)  
    {  
        switch (which)  
        {  
  
        //-- Выбран красный цвет -----  
        case 0 :  
            MainActivity.this.clrTmpMain =  
                Color.rgb(0xFF, 0x00, 0x00);  
            break;  
  
        //-- Выбран желтый цвет -----  
        case 1 :  
            MainActivity.this.clrTmpMain =  
                Color.rgb(0xFF, 0xFF, 0x00);  
            break;  
  
        //-- Выбран зеленый цвет -----  
        case 2 :  
            MainActivity.this.clrTmpMain =  
                Color.rgb(0x00, 0xFF, 0x00);  
            break;  
  
        //-- Выбран белый цвет -----  
        case 3 :  
            MainActivity.this.clrTmpMain =  
                Color.rgb(0xFF, 0xFF, 0xFF);  
            break;  
        }  
        MainActivity.this.llMain.  
        setBackgroundColor(MainActivity.  
                           this.clrTmpMain);  
    }  
});
```

4. Диалоговые окна. Классы AlertDialog, AlertDialog.Builder

```
//-- Шаг 3. Назначение диалоговому окну кнопок ---
    builder.setPositiveButton("OK",
        new DialogInterface.OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog,
                            int id)
        {
//-- Пользователь подтверждает смену цвета -----
            MainActivity.this.clrForMain =
                MainActivity.this.clrTmpMain;
        }
    });

    builder.setNegativeButton("Cancel",
        new DialogInterface.OnClickListener()
    {
        @Override
        public void onClick(DialogInterface dialog,
                            int which)
        {
//-- Пользователь отменяет решение -
//-- возвращаем предыдущий цвет фона -----
            MainActivity.this.llMain.
                setBackgroundColor(MainActivity.
                    this.clrForMain);
        }
    });
}

//-- Шаг 4. Создание объекта диалогового окна
//-- android.app.AlertDialog -----
    AlertDialog dialog = builder.create();
//-- Шаг 5. Показываем диалог -----
    dialog.show();
}
break;
}
}
```

Внешний вид работы примера из Листинга 4.3 изображен на Рис. 4.6.

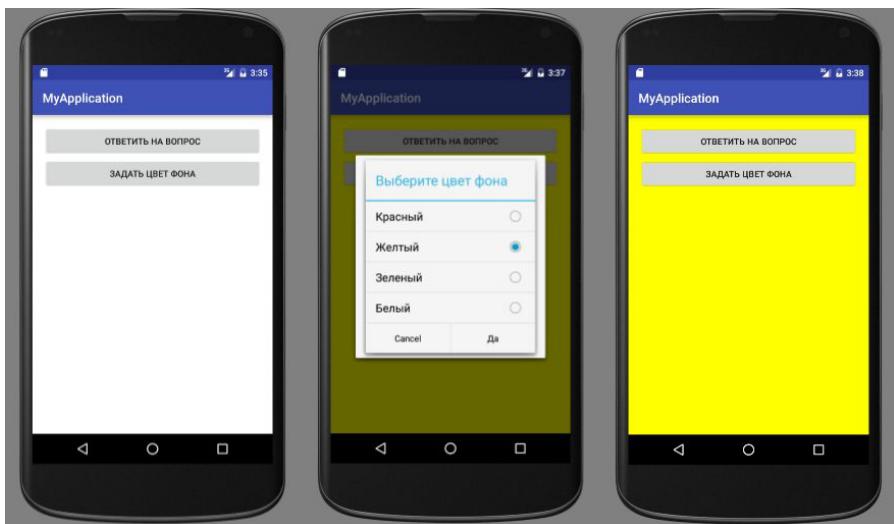


Рис. 4.6. Внешний вид работы примера из Листинга 4.3

В примере из Листинга 4.3 пользователь после нажатия на кнопку «Задать цвет фона» видит Диалоговое окно со списком одиночного выбора, в котором отображаются доступные цвета для цвета фона Активности. При выборе любого элемента списка цвет фона Активности сразу же изменяется. Если пользователь нажмет на кнопку Диалогового окна «Cancel», то Диалоговое окно закроется и цвет фона Активности вернется в прежнее значение. Если же пользователь нажмет на кнопку «OK», то Диалоговое окно закроется и Активность будет иметь цвет фона, выбранный пользователем. Для моментального изменения цвета фона Активности используется обработчик события `DialogInterface.OnClickListener`, ссылка на который

передается в метод **setSingleChoiceItems** (в Листинге 4.3 вызов этого метода выделен жирным).

4.3. AlertDialog со списком множественного выбора

Диалоговое окно **android.app.AlertDialog** может отображать и список множественного выбора. Для назначения Диалоговому окну такого списка предназначен метод класса **android.app.AlertDialog.Builder**:

```
public AlertDialog.Builder setMultiChoiceItems
    (CharSequence[] items, boolean[] checkedItems,
     DialogInterface.OnMultiChoiceClickListener
     listener);
```

Этот метод принимает следующие параметры:

- **CharSequence[] items** — массив строк, содержащий названия для элементов списка множественного выбора.
- **boolean[] checkedItems** — массив значений **true/false** с помощью которого можно указать, какие элементы списка множественного выбора будут изначально отмечены как выбранные (**true**).
- **DialogInterface.OnMultiChoiceClickListener** — ссылка на объект, который является обработчиком события выбора (клика) элемента списка множественного выбора. Этот объект должен реализовать интерфейс **android.content.DialogInterface.OnMultiChoiceClickListener** (<http://developer.android.com/reference/android/content/DialogInterface.OnMultiChoiceClickListener.html>) в котором объявлен один метод:

```
void onClick(DialogInterface dialog,
             int which, boolean isChecked);
```

принимающий ссылку на Диалоговое окно (**dialog**), являющееся источником события, индекс элемента списка (**which**), по которому осуществлен клик (нажатие) и логическое значение (**isChecked**) которое содержит значение `true` если элемент списка отмечен как выбранный и `false` в противном случае.

Рассмотрим программный код, формирующий Диалоговое окно со списком множественного выбора. Для этого в пример, рассматриваемый в данном разделе (модуль «app4») добавлена еще одна кнопка с надписью «Стиль текста» и идентификатором `R.id.btnThree` а также текстовое поле `android.widget.TextView` с текстом «Hello World» и идентификатором `R.id.tvText`. Суть примера в следующем: в Диалоговом окне с помощью списка множественного выбора пользователь будет комбинировать опции для стиля текста (Bold, Italic, Allcaps) в текстовом поле `R.id.tvText`.

В классе `MainActivity` объявляются следующие поля и константы (Листинг 4.4):

Листинг 4.4. Объявление в классе `MainActivity` полей и констант для использования их в примере для Диалогового окна со списком множественного выбора

```
public class MainActivity extends AppCompatActivity
{
    //-- Class constants -----
    /**
     * Индекс в массиве arrStyle для стиля текста BOLD
     */
    private final static int TEXT_STYLE_INDEX_BOLD = 0;
    /**
     * Индекс в массиве arrStyle для стиля текста ITALIC
     */
```

```
private final static int TEXT_STYLE_INDEX_ITALIC = 1;

/**
 * Индекс в массиве arrStyle для стиля текста ALLCAPS
 */
private final static int TEXT_STYLE_INDEX_ALLCAPS = 2;

//--- Class members -----
/***
 * Ссылка на текстовое поле android.widget.TextView
 * стиль текста которого будет меняться с помощью
 * диалогового окна множественного выбора
 */
private TextView tvText;

/***
 * Массив логических значений, содержащий признак
 * наличия соответствующего стиля у текста
 * из виджета tvText. Индексы массива задаются
 * с помощью статических констант
 * TEXT_STYLE_INDEX_XXXXX
 * <br />
 * true означает наличие соответствующего стиля
 * у текста из виджета tvText
 */
private boolean[] arrStyle = new boolean[3];

/***
 * Такой же массив, что и arrStyle, только этот
 * массив непосредственно передается диалоговому
 * окну для значений списка множественного выбора.
 * Этот массив содержит текущие выбираемые
 * пользователем настройки стиля текста.
 * В случае отмены пользователем настроек,
 * предыдущие значения настроек остаются
 * в массиве arrStyle.
 * В случае подтверждения пользователем настроек
 */
```

```

    * стиля, значения этого массива копируются
    * в массив tvStyle.
    */
private boolean[] arrTmpStyle = new boolean[3];

...
}

```

Непосредственно само создание Диалогового окна со списком множественного выбора и применение стилей которые выбрал пользователь в списке, происходит в методе **btnClick** при клике на кнопку с надписью «Стиль текста» (идентификатор кнопки **R.id.btnThree**). Программный код обработки события нажатия на кнопку «Стиль текста» приведен в Листинге 4.5.:

Листинг 4.5. Пример реализации Диалогового окна `android.app.AlertDialog` со списком множественного выбора для реализации возможности комбинирования стилей для текста

```

public void btnClick(View v)
{
    switch (v.getId())
    {
        //-- Обработка события нажатия на кнопку
        //-- "Стиль текста" -----
        case R.id.btnThree :
        {

            //-- Шаг 1. Создание объекта
            //-- android.app.AlertDialog.Builder -----
            AlertDialog.Builder builder = new AlertDialog.
                Builder(this, android.R.style.
                    Theme_Holo_Light_Dialog
            );

```

```
//-- Шаг 2. Формирование заголовка окна и его
//-- содержимого -----
builder.setTitle("Отметьте требуемые стили");

for (int i = 0; i < this.arrStyle.length; i++)
    this.arrTmpStyle[i] = this.arrStyle[i];
builder.setMultiChoiceItems(
    new String[] {"BOLD", "ITALIC", "ALL CAPS"},
    this.arrTmpStyle, new DialogInterface.
    OnMultiChoiceClickListener()
{
    @Override
    public void onClick(DialogInterface
                        dialog, int which,
                        boolean isChecked)
    {
        switch (which)
        {
//-- Выбран элемент списка "BOLD" -----
            case MainActivity.TEXT_STYLE_INDEX_BOLD :
                if (isChecked)
                {
                    MainActivity.this.tvText.
                    setTypeface(Typeface.DEFAULT,
                                (arrTmpStyle[MainActivity.
                                TEXT_STYLE_INDEX_ITALIC])?
                                Typeface.BOLD_ITALIC:Typeface.BOLD);
                }
                else
                {
                    MainActivity.this.tvText.
                    setTypeface(Typeface.DEFAULT,
                                (arrTmpStyle[MainActivity.
                                TEXT_STYLE_INDEX_ITALIC])?
                                Typeface.ITALIC:Typeface.NORMAL);
                }
        }
    }
}
```

```
        break;

//-- Выбран элемент списка "ITALIC" -----
case MainActivity.TEXT_STYLE_INDEX_ITALIC :
    if (isChecked)
    {
        MainActivity.this.tvText.
        setTypeface(Typeface.DEFAULT,
        (arrTmpStyle[MainActivity.
        TEXT_STYLE_INDEX_BOLD])?
        Typeface.BOLD_ITALIC:Typeface.
        ITALIC);
    }
    else
    {
        MainActivity.this.tvText.
        setTypeface(Typeface.DEFAULT,
        (arrTmpStyle[MainActivity.
        TEXT_STYLE_INDEX_BOLD])?
        Typeface.BOLD:Typeface.NORMAL);
    }
    break;

//-- Выбран элемент списка "ALL CAPS" -----
case MainActivity.TEXT_STYLE_INDEX_ALLCAPS :
    MainActivity.this.tvText.
    setAllCaps(isChecked);
    break;
}
}

//-- Шаг 3. Назначение диалоговому окну кнопок ---
builder.setPositiveButton("OK",
    new DialogInterface.OnClickListener()
{
```

```
public void onClick(DialogInterface dialog,
                     int id)
{
    //-- Пользователь подтвердил смену стилей -
    //-- запомним новые значения -----
    for (int i = 0; i < arrStyle.length; i++)
        arrStyle[i] = arrTmpStyle[i];
}
});
```

```
builder.setNegativeButton("Cancel",
                         new DialogInterface.OnClickListener()
{
    @Override
    public void onClick(DialogInterface dialog,
                        int which)
    {
        //-- Пользователь отменил смену стилей -
        //-- восстановим предыдущие значения
        MainActivity.this.tvText.setAllCaps(
            arrStyle[MainActivity.
                TEXT_STYLE_INDEX_ALLCAPS]);

        if (arrStyle[MainActivity.
            TEXT_STYLE_INDEX_BOLD] &&
            arrStyle[MainActivity.
                TEXT_STYLE_INDEX_ITALIC])
        {
            MainActivity.this.tvText.setTypeface(
                Typeface.DEFAULT, Typeface.BOLD_ITALIC);
        }
        else
        if (arrStyle[MainActivity.
            TEXT_STYLE_INDEX_BOLD] &&
            !arrStyle[MainActivity.
                TEXT_STYLE_INDEX_ITALIC])
```

```
{  
    MainActivity.this.tvText.setTypeface(  
        Typeface.DEFAULT, Typeface.BOLD);  
}  
else  
if (!arrStyle[MainActivity.  
    TEXT_STYLE_INDEX_BOLD] &&  
    arrStyle[MainActivity.  
    TEXT_STYLE_INDEX_ITALIC])  
{  
    MainActivity.this.tvText.setTypeface(  
        Typeface.DEFAULT, Typeface.  
        ITALIC);  
}  
else  
{  
    MainActivity.this.tvText.setTypeface(  
        Typeface.DEFAULT, Typeface.  
        NORMAL);  
}  
}  
});  
  
//--- Шаг 4. Создание объекта диалогового окна  
//--- android.app.AlertDialog -----  
AlertDialog dialog = builder.create();  
  
//--- Шаг 5. Показываем диалог -----  
dialog.show();  
}  
break;  
}  
}  
}
```

Внешний вид работы примера из Листинга 4.5 изображен на Рис. 4.7.

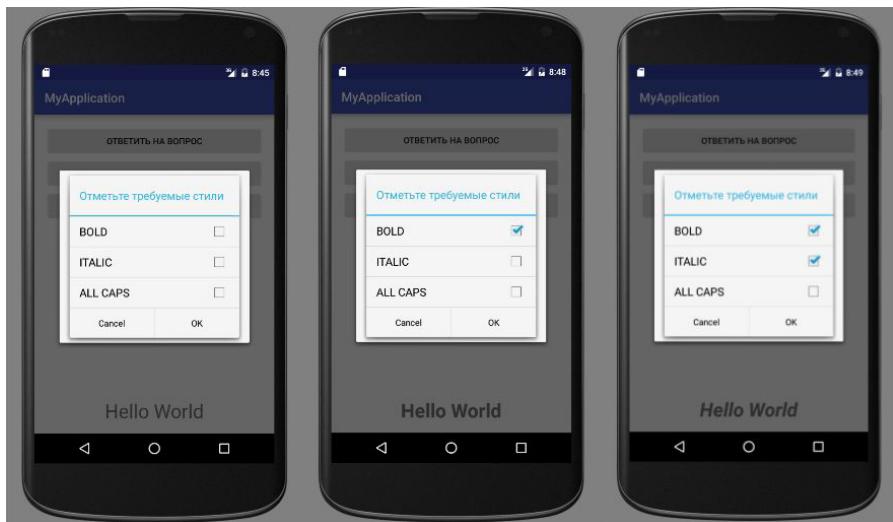


Рис. 4.7. Внешний вид работы Диалогового окна android.app.AlertDialog со списком множественного выбора из Листинга 4.5

Небольшие пояснения к работе примера из Листинга 4.5. Пользователь нажимает на кнопку «Стиль текста» и появляется Диалоговое окно `android.app.AlertDialog` со списком множественного выбора, с помощью которого пользователь может комбинировать стили текста (BOLD — текст жирный, ITALIC — текст курсивный, ALL CAPS — текст заглавными буквами) для текста «Hello World», отображенного в виджете `android.widget.TextView`. Как только пользователь устанавливает (или снимает) выделение любого элемента списка, соответствующие изменения в стиле текста сразу же отображаются на экране. Это достигается с помощью обработки события клика по элементу списка `DialogInterface.OnMultiChoiceClickListener`. При закрытии Диалогового окна с помощью кнопки «Cancel» комбинация стилей для теста «Hello World» откатывается

на комбинацию, которая была применена к тексту до показа Диалогового окна. При закрытии окна с помощью кнопки «ОК» выбранная пользователем комбинация стилей для текста остается примененной к тексту.

Исходный код рассматриваемого примера находится в модуле «app4» среди файлов исходного кода, прилагаемых к данному уроку.

4.4. AlertDialog с пользовательским набором виджетов

Если появляется необходимость предоставить пользователю возможность вводить информацию в Диалоговом окне и списки одиночного или множественного выбора не удается использовать, то можно разместить в Диалоговом окне **android.widget.AlertDialog** свой набор виджетов по своему усмотрению. Для этой цели? Необходимо вместо методов **setMessage** или **setSingleChoiceItems** или **setMultiChoiceItems** использовать метод класса **android.app.AlertDialog.Builder**:

```
AlertDialog.Builder.setView (int layoutResId);
```

или

```
AlertDialog.Builder.setView (View view);
```

Этот метод устанавливает в качестве содержимого Диалогового окна **android.app.AlertDialog** некоторый виджет, который передается в качестве параметра (или идентификатор файла ресурсов с раскладкой **layoutResId** или ссылка на виджет **view**). Внимание! Метод **setView**, который принимает идентификатор ресурса с макетом

раскладки работает для уровня API 23! В нашем примере (в целях совместимости с более ранними версиями API) будет использован второй метод `getView`. Виджет для содержимого Диалогового окна будет создан с помощью объекта `android.view.LayoutInflater` и затем размещен на Диалоговом окне.

Для примера этого раздела создадим в приложении еще одну кнопку с надписью «Приветствие» и идентификатором `R.id.btnFour`. При клике на эту кнопку, будет появляться Диалоговое окно с двумя текстовыми полями `android.widget.EditText`, в которые пользователь введет свои фамилию и имя и после нажатия на кнопку «OK» в Тосте выведется приветствие «Привет Фамилия Имя!».

Далее, создадим файл ресурсов с макетом раскладки виджетов, который и будет содержимым Диалогового окна для примера этого раздела. Созданный файл называется `/res/layout/dialog_content.xml`. Код примера находится в модуле «app4» среди файлов исходного кода прилагаемых к этому уроку. Xml код макета раскладки для файла `/res/layout/dialog_content.xml` приведен в Листинге 4.6.

Листинг 4.6. Xml верстка макета раскладки
для содержимого Диалогового окна
`android.app.AlertDialog` рассматриваемого примера

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

```
<TableRow>
    <TextView
        android:textSize="9pt"
        android:layout_gravity="center"
        android:text="Фамилия" />

    <EditText
        android:id="@+id/etLName" />
</TableRow>

<TableRow>
    <TextView
        android:textSize="9pt"
        android:layout_gravity="center"
        android:text="Имя" />

    <EditText
        android:id="@+id/etFName" />
</TableRow>
</TableLayout>
```

Внешний вид Диалогового окна **android.widget.Alert Dialog** с содержимым из Листинга 4.6 изображен на Рис. 4.8.

При нажатии на кнопку «Приветствие» появляется Диалоговое окно (Рис. 4.8), в котором пользователь должен ввести Фамилию и Имя для приветствия. При нажатии на кнопку «OK» Диалоговое окно закроется и появится всплывающее сообщение **Toast** с приветствием. При нажатии на кнопку «Cancel» Диалоговое окно закроется и ничего не произойдет. Создание и отображение Диалогового окна осуществляется в методе **btnClick** — обработчике события клика по кнопке «Приветствие» (идентификатор кнопки **R.id.btnFour**). Программный код приведен в Листинге 4.7.

4. Диалоговые окна. Классы AlertDialog, AlertDialog.Builder

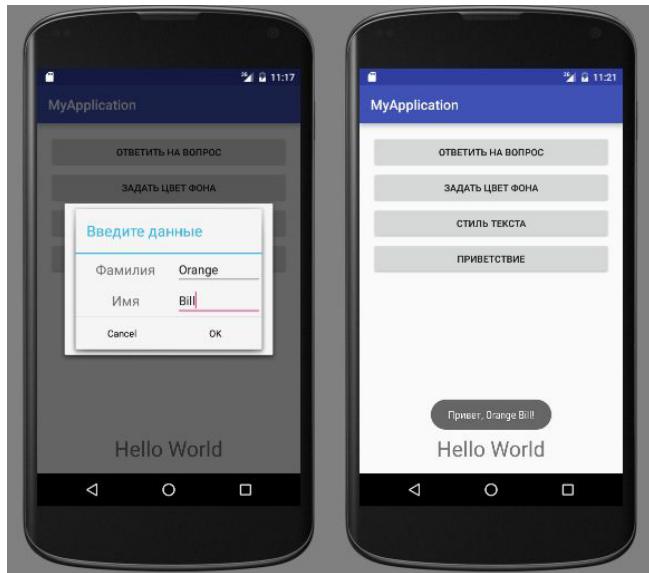


Рис. 4.8. Внешний вид работы примера из Листинга 4.7

Листинг 4.7. Формирование и отображение Диалогового окна android.app.AlertDialog с содержимым в виде макета из файла ресурсов /res/layout/dialog_content.xml

```
public void btnClick(View v)
{
    switch (v.getId())
    {
        /*
         * Обработка события нажатия на кнопку "Приветствие"
         * -----
        */
        case R.id.btnFour :
        {
            //-- Шаг 1. Создание объекта
            //-- android.app.AlertDialog Builder -----

```

```
AlertDialog.Builder builder =
        new AlertDialog.Builder(this,
                android.R.style.Theme_Holo_Light_Dialog
        );

//-- Шаг 2. Формирование заголовка окна и его
//-- содержимого -----
        builder.setTitle("Введите данные");

LayoutInflater inflater = this.getLayoutInflater();
final View view = inflater.inflate(
        R.layout.dialog_content,
        null, false);
builder.setView(view);

//-- Шаг 3. Назначение диалоговому окну кнопок ---
        builder.setPositiveButton("OK",
                new DialogInterface.OnClickListener()
        {
            public void onClick(DialogInterface
                    dialog, int id)
            {
//-- Чтение содержимого текстовых полей
//-- Диалогового окна -----
                EditText etLName = (EditText) view.
                        findViewById(R.id.etLName);
                EditText etFName = (EditText) view.
                        findViewById(R.id.etFName);

                String greeting = etLName.
                        getText().toString() + " " +
                        etFName.getText().toString();
//-- Вывод приветствия в Тосте -----
                Toast.makeText(MainActivity.this,
                        "Привет, " + greeting + "!",
                        Toast.LENGTH_SHORT).show();
            }
        });
    }
```

```
builder.setNegativeButton("Cancel",
    new DialogInterface.OnClickListener()
    {
        public void onClick(DialogInterface dialog, int id)
        {
        }
    });
}

//--- Шаг 4. Создание объекта диалогового окна
//--- android.app.AlertDialog -----
AlertDialog dialog = builder.create();

//--- Шаг 5. Показываем диалог -----
dialog.show();
}
break;
}
}
```

Внешний вид работы примера из Листинга 4.7 изображен на Рис. 4.8.

4.5. DatePickerDialog. Диалоговое окно для выбора даты

Существуют и специализированные Диалоговые окна — например, Диалоговые окна выбора даты или времени. В этом разделе рассмотрим Диалоговое окно выбора даты `android.app.DatePickerDialog` (<http://developer.android.com/reference/android/app/DatePickerDialog.html>). Иерархия классов для класса `android.app.DatePickerDialog`:

```
java.lang.Object  
|  
+--- android.app.Dialog  
|  
+--- android.app.AlertDialog  
|  
+--- android.app.DatePickerDialog
```

Чтобы создать объект экземпляр класса **android.app.DatePickerDialog**, необходимо воспользоваться конструктором:

```
DatePickerDialog(Context context,  
                  DatePickerDialog.OnDateSetListener callBack,  
                  int year, int monthOfYear, int dayOfMonth);
```

Конструктор, кроме ссылки на объект «Контекст приложения» (**context**), принимает ссылку на объект (**callback**), реализующий интерфейс **android.app.DatePickerDialog.OnDateSetListener** (<http://developer.android.com/reference/android/app/DatePickerDialog.OnDateSetListener.html>), который используется для получения уведомления о том, что пользователь завершил выбор даты, а так же начальные значения (**year**, **monthOfYear**, **dayOfMonth**) для даты, которая будет отображена в Диалоговом окне сразу же после его показа.

В интерфейсе **android.app.DatePickerDialog.OnDateSetListener** объявлен всего один метод:

```
void onDateSet (DatePicker view, int year,  
                int monthOfYear, int dayOfMonth);
```

который можно использовать для того, чтобы получить событие завершения выбора даты в Диалоговом окне. Метод

принимает ссылку на виджет `android.widget.DatePicker` (<http://developer.android.com/reference/android/widget/DatePicker.html>), который является источником события, а так же значения даты (`year`, `monthOfYear`, `dayOfMonth`), которые установил пользователь. Внимание, значения месяца здесь совпадают со значениями объектов класса `java.util.Calendar` (<http://developer.android.com/reference/java/util/Calendar.html>), то есть начинаются с 0.

Для получения информации о выборе даты можно так же переопределить метод класса `android.app.DatePickerDialog`:

```
void onDateChanged(DatePicker view,  
                   int year, int month, int day);
```

Этот метод будет вызываться всякий раз, как будет происходить смена даты в Диалоговом окне `android.app.DatePickerDialog`. Параметры, передаваемые в этот метод совпадают с параметрами метода `onDateSet` интерфейса `android.app.DatePickerDialog.OnDateSetListener`, который описывался чуть выше.

После создания объекта экземпляра класса `android.app.DatePickerDialog` необходимо назначить Диалоговому окну «позитивную» (метод `setButton` со значением `DialogInterface.BUTTON_POSITIVE`) и «негативную» (метод `setButton` со значением `DialogInterface.BUTTON_NEGATIVE`) кнопки и показать окно с помощью вызова метода `show()`.

В пример модуля «app4» добавим кнопку с надписью «Установить дату» с идентификатором `R.id.btnFive`, при нажатии на которую будет появляться Диалоговое

окно `android.app.DatePickerDialog`, в котором пользователь сможет выбрать требуемую дату. Для примера нам понадобятся дополнительные поля для объекта класса `MainActivity` (`year`, `month`, `day`), в которых будем запоминать выбранные значения года, месяца и дня, чтобы, когда пользователь нажмет на кнопку «Выбрать», наше приложение уже знало об этих значениях. Объявление этих полей приведено в Листинге 4.8:

Листинг 4.8. Объявление в классе `MainActivity` полей, необходимых для рассматриваемого в данном разделе примера из Листинга 4.9

```
class MainActivity extends AppCompatActivity
{
    /*
     * Поля, относящиеся к примеру на Диалоговое окно
     * выбора даты android.app.DatePickerDialog
     * -----
     */
    /**
     * Выбранный пользователем год
     */
    private int year;

    /**
     * Выбранный пользователем месяц (0..11)
     */
    private int month;
    /**
     * Выбранный пользователем день месяца
     */
    private int day;
    ...
}
```

Пример создания и отображения Диалогового окна `android.app.DatePickerDialog` осуществляется в методе обработчике события клика `btnDateClick` по кнопке «Установить дату», которая имеет идентификатор `R.id.btnFive`. Код метода `btnDateClick` (метод находится в классе `MainActivity` модуля «app4») приведен в Листинге 4.9:

Листинг 4.9. Создание и отображение Диалогового окна `android.app.DatePickerDialog`

```
/***
 * Обработчик события клика на кнопку
 * "Установить дату"
 * @param v - ссылка на объект
 * android.widget.Button который
 * является источником события.
 */
public void btnDateClick(View v)
{
    if (this.year == 0 || this.month == 0 ||
        this.day == 0)
    {
        //--- Получаем текущую дату с помощью
        //--- java.util.Calendar -----
        Calendar C      = Calendar.getInstance();

        this.year = C.get(Calendar.YEAR);
        this.month = C.get(Calendar.MONTH);
        this.day = C.get(Calendar.DAY_OF_MONTH);
    }
    //--- 1. Создание объекта android.app.
    //--- DatePickerDialog -----
    DatePickerDialog DPD = new DatePickerDialog(this,
                                              null, this.year,
                                              this.month, this.day)
{
```

```
    @Override
    public void onDateChanged(DatePicker view,
                                int year, int month,
                                int day)
    {
        //-- Сохраним выбор пользователя -----
        MainActivity.this.year = year;
        MainActivity.this.month = month;
        MainActivity.this.day = day;
    }
};

//-- 2. Назначение кнопок "OK", "Cancel"
//-- Диалоговому окну -----
DPD.setButton(DialogInterface.BUTTON_POSITIVE,
                "Выбрать", new DialogInterface.
                OnClickListener()
{
    @Override
    public void onClick(DialogInterface
                        dialog, int which)
    {
        //-- Вывод в Тосте выбранной даты -----
        String txt =
            ((MainActivity.this.day <
            10)?"0":"" + MainActivity.this.
            day + "/" + ((MainActivity.this.
            month < 9)? "0":"")) + (MainActivity.
            this.month + 1) + "/" +
            MainActivity.this.year;

        Toast.makeText(MainActivity.this,
                    "Выбранная дата дд/мм/гггг : " +
                    txt, Toast.LENGTH_SHORT).show();
    }
});
```

4. Диалоговые окна. Классы AlertDialog, AlertDialog.Builder

```
DPD.setButton(DialogInterface.BUTTON_NEGATIVE,  
                "Отменить", new DialogInterface.  
                OnClickListener()  
                {  
                    @Override  
                    public void onClick(DialogInterface  
                            dialog, int which)  
                    {  
                        //-- Пользователь отменил выбор - ничего не делаем --  
                    }  
                } );  
//-- 3. Показываем диалоговое окно  
//-- android.app.DatePickerDialog -----  
DPD.show();  
}
```

Внешний вид рассматриваемого в Листинге 4.9 примера изображен на Рис. 4.9.

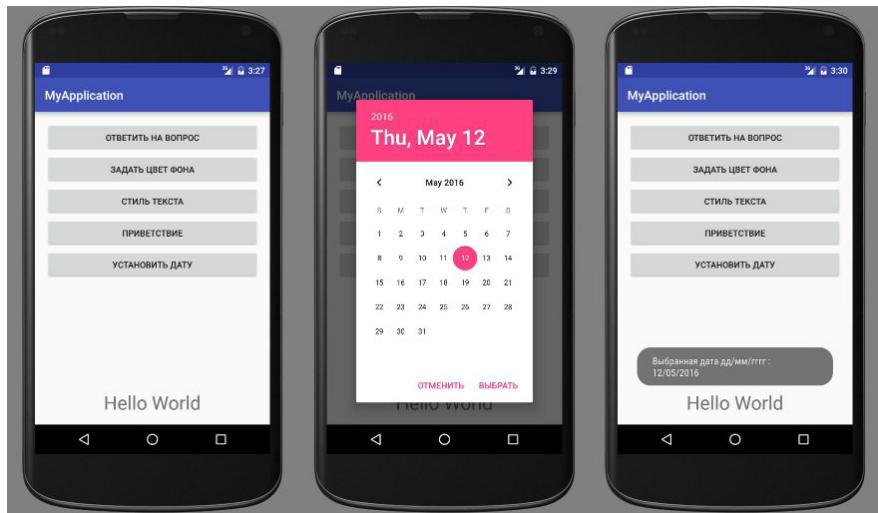


Рис. 4.9. Внешний вид работы диалога android.app.DatePickerDialog, код примера из Листинга 4.9

Исходный код примера из этого раздела находится в модуле «app4» файлов исходного кода, прилагаемых к данному уроку.

4.6. TimePickerDialog. Диалоговое окно для выбора времени

Рассмотрим также и Диалоговое окно для выбора времени `android.app.TimePickerDialog` (<http://developer.android.com/reference/android/app/TimePickerDialog.html>). Иерархия классов для класса `android.app.TimePickerDialog`:

```
java.lang.Object
|
+--- android.app.Dialog
|
+--- android.app.AlertDialog
|
+--- android.app.TimePickerDialog
```

Чтобы создать объект экземпляр класса `android.app.TimePickerDialog`, необходимо воспользоваться конструктором:

```
TimePickerDialog(Context context,
                  TimePickerDialog.OnTimeSetListener
                  listener, int hourOfDay,
                  int minute, boolean is24HourView);
```

Конструктор принимает ссылку на объект контекста приложения (`context`), ссылку на объект, реализующий интерфейс `android.app.TimePickerDialog.OnTimeSetListener`, который будет получать события выбора времени пользователем в Диалоговом окне, и начальные значения для

времени, которые необходимо отобразить в Диалоговом окне (**hourOfDay**, **minute**). Параметр **is24HourView** предназначен для указания в каком формате указывать значения часов — в 24-х часовом (true) или в 12-и часовом «ам/pm» формате (false).

В интерфейсе **android.app.TimePickerDialog.OnTimeSetListener** (<http://developer.android.com/reference/android/app/TimePickerDialog.OnTimeSetListener.html>) объявлен один метод:

```
void onTimeSet (TimePicker view, int hourOfDay,  
                int minute);
```

Этот метод будет вызываться всякий раз, когда пользователь будет изменять значения часов или минут в Диалоговом окне выбора времени **android.app.TimePickerDialog**. Метод принимает следующие параметры:

- **TimePicker view** — ссылка на виджет **android.widget.TimePicker** (<http://developer.android.com/reference/android/widget/TimePicker.html>), который отображается в Диалоговом окне выбора времени.
- **int hourOfDay, int minute** — текущие выбранные пользователем значения часов и минут.

Для получения информации о событии выбора пользователем времени можно не использовать этот интерфейс **android.app.TimePickerDialog.OnTimeSetListener**, а переопределить метод класса **android.app.TimePickerDialog**:

```
void onTimeChanged(TimePicker view,  
                   int hourOfDay, int minute);
```

Переопределение этого метода для объекта `android.app.TimePickerDialog` равнозначно использованию объекта, реализующего интерфейс `android.app.TimePickerDialog.OnTimeSetListener`. И по списку принимаемых параметров так же.

После создания объекта экземпляра класса `android.app.TimePickerDialog` необходимо назначить Диалоговому окну «позитивную» (метод `setButton` со значением `DialogInterface.BUTTON_POSITIVE`) и «негативную» (метод `setButton` со значением `DialogInterface.BUTTON_NEGATIVE`) кнопки и показать окно с помощью вызова метода `show()`.

В пример модуля «app4» добавим кнопку с надписью «Установить время» с идентификатором `R.id.btnExit`, при нажатии на которую будет появляться Диалоговое окно `android.app.TimePickerDialog`, в котором пользователь сможет выбрать требуемое время. Для примера нам понадобятся дополнительные поля для объекта класса `MainActivity` (`hour`, `minute`), в которых будем запоминать выбранные значения часов и минут, чтобы, когда пользователь нажмет на кнопку «Выбрать», наше приложение уже знало об этих значениях. Объявление этих полей приведено в Листинге 4.10:

Листинг 4.10. Объявление в классе `MainActivity` полей, необходимых для рассматриваемого в данном разделе примера из Листинга 4.11

```
class MainActivity extends AppCompatActivity
{
    /*
     * Поля, относящиеся к примеру на Диалоговое окно
     * выбора времени android.app.TimePickerDialog
    
```

```

    */
    /**
     * Выбранное пользователем значение часа
     */
    private int hour = -1;
    /**
     * Выбранные пользователем значения минут
     */
    private int minute = -1;
    ...
}

```

Пример создания и отображения Диалогового окна **android.app.TimePickerDialog** осуществляется в методе обработчике события клика **btnTimeClick** по кнопке «Установить время», которая имеет идентификатор **R.id.btnSix**. Код метода **btnTimeClick** (метод находится в классе **MainActivity** модуля «app4») приведен в Листинге 4.11:

Листинг 4.11. Создание и отображение Диалогового окна **android.app.TimePickerDialog**

```

    /**
     * Обработчик события клика на кнопку
     * "Установить время"
     * @param v- ссылка на объект android.widget.Button,
     * который является источником события.
     */
    public void btnTimeClick(View v)
    {
        if (this.hour == -1 || this.minute == -1)
        {
            //-- Значения не проинициализированы -
            //-- проинициализируем ----
            Calendar C = Calendar.getInstance();
            this.hour = C.get(Calendar.HOUR_OF_DAY);

```

```
        this.minute = C.get(Calendar.MINUTE);
    }

//-- 1. Создание объекта android.app.TimePickerDialog
TimePickerDialog TPD = new TimePickerDialog(this,
        null, this.hour, this.minute, true)
{
    @Override
    public void onTimeChanged(TimePicker view,
            int hour, int minute)
    {
//-- Запомним выбранные пользователем значения
//-- часов и минут -----
        MainActivity.this.hour = hour;
        MainActivity.this.minute = minute;
    }
};

//-- 2. Назначение диалоговому окну кнопок -----
TPD.setButton(DialogInterface.BUTTON_POSITIVE,
        "Выбрать", new DialogInterface.
        OnClickListener()
{
    @Override
    public void onClick(DialogInterface
            dialog, int which)
    {
        String txt = ((MainActivity.this.hour
                < 10)?"0":"" + MainActivity.
                this.hour + ":" +
                ((MainActivity.this.minute <
                10)?"0":"")) +
                MainActivity.this.minute;

//-- Вывод в Тосте выбранного времени -----
        Toast.makeText(MainActivity.this,
                "Выбранное время чч:мм : " + txt,
                Toast.LENGTH_LONG).show();
    }
});
```

```
TPD.setButton(DialogInterface.BUTTON_NEGATIVE,  
                "Отменить", new DialogInterface.  
                OnClickListener()  
    {  
        @Override  
        public void onClick(DialogInterface  
                            dialog, int which)  
        {  
            //-- Пользователь отменил выбор - ничего не делаем --  
        }  
    };  
//-- 3. Показываем диалоговое окно  
//-- android.app.TimePickerDialog -----  
TPD.show();  
}
```

Внешний вид рассматриваемого в Листинге 4.11 примера изображен на Рис. 4.10.

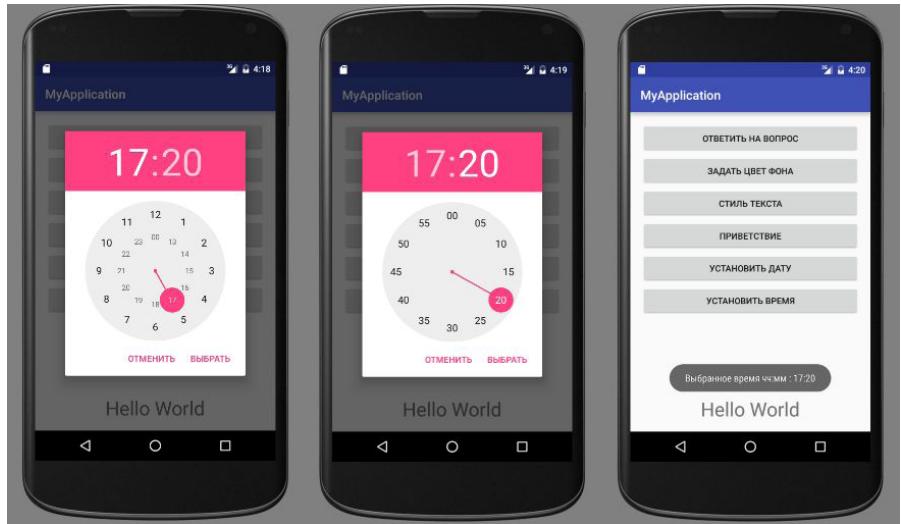


Рис. 4.10. Внешний вид работы диалога android.app.TimePickerDialog, код примера из Листинга 4.11

Исходный код примера из этого раздела находится в модуле «app4» файлов исходного кода, прилагаемых к данному уроку.

5. Работа с файлами на внешнем носителе

Внешним носителем принято считать флэш-карту памяти, которую можно вставлять в устройство мобильного телефона и использовать для сохранения файлов фотографий, музыкальных файлов, и файлов, загруженных из сети Интернет. Примером такой внешней карты памяти является SD карта (https://ru.wikipedia.org/wiki/Secure_Digital).

Для примера данного раздела создан модуль «app5», который можно найти среди файлов исходных кодов, прилагаемых к данному уроку.

Самый первый шаг, который необходимо сделать при создании приложения, работающего с внешним носителем, это указать в Манифесте приложения информацию о том, что приложению нужен доступ на чтение или на чтение/запись файлов на внешнем носителе. Без указания в Манифесте приложения такой информации, ваше приложение не сможет работать с файлами на внешнем носителе. Соответственно, пользователь, при установке вашего приложения на мобильный телефон будет проинформирован Операционной Системой Android о том, что приложение будет иметь доступ к файлам на внешнем носителе, и что пользователь должен дать на это разрешение (Рис. 5.4).

Давайте внесем необходимую информацию в Манифест приложения. Для этого необходимо открыть xml

файл с Манифестом приложения. На Рис. 5.1 изображено месторасположение файла Манифеста приложения.

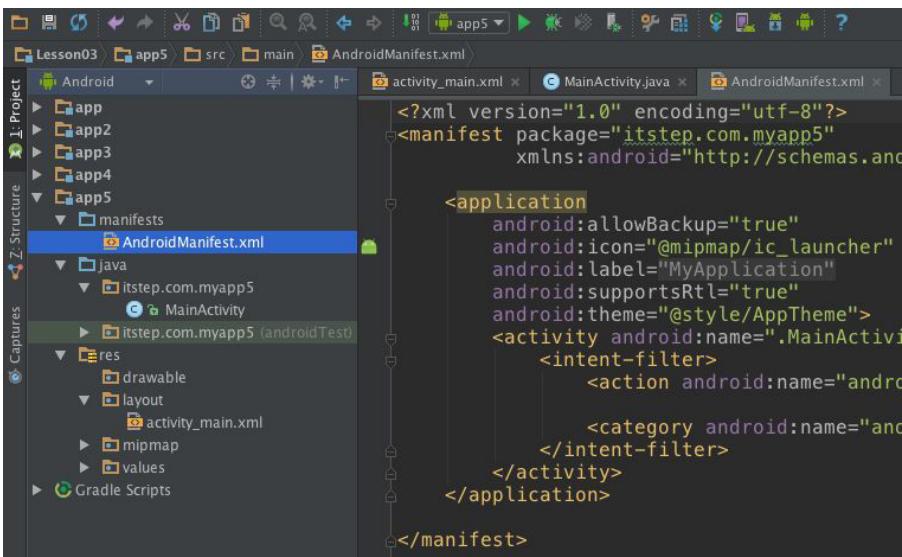


Рис. 5.1. Месторасположение файла Манифеста приложения в модуле Android Studio

Как видно из Рис. 5.1, Манифест приложения расположена в файле `имя_модуля/manifests/AndroidManifest.xml`. О файле Манифеста приложения мы уже упоминали в первом уроке данного курса. Напомним, что Манифест приложения содержит важную информацию о приложении, которая необходима Операционной Системе Android для запуска и исполнения любого приложения. Подробно о файле Манифеста приложения можно прочитать по ссылке <http://developer.android.com/intl/ru/guide/topics/manifest/manifest-intro.html>.

Для того, чтобы добавить в Манифест приложения разрешение на чтение/запись файлов на внешнем

носителе, необходимо внести в Манифест следующий код (Листинг 5.1):

Листинг 5.1. Размещение в Манифесте приложения разрешения на чтение/запись файлов на внешнем носителе

```
<manifest ...>
    <application>
        ...
    </application>
    <uses-permission android:name="android.permission.
        WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

Если приложению необходим доступ только на чтение файлов с внешнего носителя, то разрешение должно выглядеть следующим образом (Листинг 5.2):

Листинг 5.2. Размещение в Манифесте приложения разрешения на чтение файлов на внешнем носителе

```
<manifest ...>
    <application>
        ...
    </application>
    <uses-permission android:name="android.permission.
        READ_EXTERNAL_STORAGE" />
    ...
</manifest>
```

В наш рассматриваемый пример модуля «app5» добавляем разрешение на чтение/запись файлов, как показано в Листинге 5.1.

Далее. Поскольку речь идет о внешнем носителе, то ситуация с внешним носителем может быть абсолютно

разной: внешний носитель может отсутствовать, внешний носитель может быть поврежден, может быть занят в режиме подключения мобильного телефона к компьютеру в режиме USB-накопителя и т. д. Поэтому перед тем как выполнять операции над файлами, находящимися на внешнем носителе, необходимо проверить состояние внешнего носителя. Для получения информации о состоянии внешнего носителя и о другой информации о внешнем носителе, используется класс `android.os.Environment` (<http://developer.android.com/intl/ru/reference/android/os/Environment.html>). Этот класс предназначен для получения информации о переменных окружения Операционной Системы Android. Иерархия класса следующая:

```
java.lang.Object
|
+-- android.os.Environment
```

Для получения информации о состоянии внешнего носителя, предназначен статический метод класса `android.os.Environment`:

```
public static String android.os.Environment.
    getExternalStorageState ();
```

Метод возвращает строковое значение, содержащее состояние внешнего носителя. Значения представлены в виде констант:

- `Environment.MEDIA_UNKNOWN` — состояние носителя неизвестно (не определено).
- `Environment.MEDIA_REMOVED` — носитель отсутствует (отсоединен).

- **Environment.MEDIA_UNMOUNTED** — носитель физически подключен к мобильному устройству, но в Операционной Системе логически не подключен («не примонтирован» в терминологии ОС Unix).
- **Environment.MEDIA_CHECKING** — носитель существует и находится в состоянии «проверка диска».
- **Environment.MEDIA_NOFS** — носитель присутствует, но не отформатирован или отформатирован под неизвестную файловую систему.
- **Environment.MEDIA_MOUNTED** — носитель присутствует и «примонтирован» к Операционной Системе Anrdoid в режиме «чтение/запись». Это состояние готовности носителя для операций с файлами на чтение и запись.
- **Environment.MEDIA_MOUNTED_READ_ONLY** — носитель присутствует и «примонтирован» к Операционной Системе Anrdoid в режиме «только чтение». Это состояние готовности носителя для чтения файлов и их содержимого.
- **Environment.MEDIA_SHARED** — носитель физически присутствует но «не примонтирован», так как в настоящий момент мобильное устройство подключено к компьютеру как USB-накопитель.
- **Environment.MEDIA_BAD_REMOVAL** — носитель извлечен из мобильного устройства без предварительной операции «размонтирования».
- **Environment.MEDIA_UNMOUNTABLE** — носитель физически присутствует но не может быть «примонтирован» по причине неисправности самого носителя.

Из вышеперечисленных возможных состояний внешнего носителя интересующими нас состояниями,

то есть состояниями, при которых можно взаимодействовать с файлами, являются два состояния: `Environment.MEDIA_MOUNTED` и `Environment.MEDIA_MOUNTED_READ_ONLY`. Причем в состоянии `Environment.MEDIA_MOUNTED_READ_ONLY` у нас нет возможности выполнять операции записи файлов — можем только выполнять операции чтения.

Для удобства определения состояния готовности внешнего носителя, как правило, разработчики создают в классе Активности два метода, которые возвращают логическое значение `true` или `false`. Первый метод возвращает `true`, если внешний носитель готов для операций чтения/записи, а второй метод возвращает `true`, если внешний носитель готов только для операций чтения. Пример таких методов приведен в Листинге 5.3:

Листинг 5.3. Пример методов, определяющих состояние готовности внешнего носителя для операций «чтения/записи» и «чтения»

```
/*
 * Метод проверяет готовность внешнего носителя
 * для операций чтения/записи.-----
 */
public boolean isExternalStorageWritable()
{
    String state = Environment.getExternalStorageState();
    return (state.equals(Environment.MEDIA_MOUNTED));
}

/*
 * Метод проверяет готовность внешнего носителя
 * для операций чтения.
 * -----
 */
```

```
public boolean isExternalStorageReadable()
{
    String state = Environment.getExternalStorageState();
    return (state.equals(Environment.MEDIA_MOUNTED) ||
            state.equals(Environment.
                           MEDIA_MOUNTED_READ_ONLY));
}
```

Получив состояние готовности внешнего носителя нужно узнать путь к каталогу, к которому он примонтирован. Путь к каталогу необходим для файловых операций — ведь для того чтобы, например, прочитать файл, нужно указать к нему путь. Для определения пути к каталогу внешнего носителя предназначен метод класса **android.os.Environment**:

```
public static File android.os.Environment.
    getExternalStorageDirectory();
```

Этот метод возвращает объект **java.io.File**, содержащий путь к каталогу. С классом **java.io.File** вы должны быть знакомы из курса программирования на языке Java.

Давайте сразу попробуем узнать состояние готовности внешнего носителя и путь к каталогу, к которому он примонтирован. Для этого в методе **onCreate** класса Активности напишем следующий код (Листинг 5.4):

Листинг 5.4. Пример определения корневого файлового каталога для внешнего носителя

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

```
    setContentView(R.layout.activity_main);
    //--- Определение готовности внешнего носителя
    //--- для чтения/записи -----
    if (this.getExternalStorageWritable())
    {
        //--- Получение пути к каталогу внешнего носителя --
        File esMainDir = Environment.
            getExternalStorageDirectory();
        Log.d(MainActivity.TAG, "Каталог внешнего
            носителя : " + esMainDir.
            getAbsolutePath());
    }
    else
    {
        Log.d(MainActivity.TAG,
            "Внешний носитель не готов!");
    }
}
```

Результат исполнения примера из Листинга 5.4 для эмулятора «Nexus-4 API 23» (intel emulator64_x86) выдал такой результат:

```
/storage/emulated/0
```

Сразу оговоримся, что для разных эмуляторов и реальных мобильных устройств этот путь будет различаться.

Разработчикам приложений для ОС Android предоставляется удобный инструмент для получения информации об эмулируемом устройстве. Этот инструмент — специальное приложение, прилагаемое к среде разработки Android Studio, называется «Android Device Monitor» (или сокращенно ADM). Запустить его можно непосредственно из Android Studio или с помощью клика по иконке «Android

Device Monitor» на Панели инструментов или через выбор опции Меню «Tools / Android / Android Device Monitor».

Запустив приложение «Android Device Monitor», можно узнать много интересной информации об эмулируемом устройстве. Например, распределение динамической памяти, статистику сетевого обмена и т. д. И конечно же, в «Android Device Monitor» есть Файловый Менеджер, с помощью которого можно выполнять файловые операции на устройстве и даже загружать в эмулируемое устройство файлы с компьютера и выгружать файлы с эмулируемого устройства в компьютер. Это очень полезная возможность, которая пригодится нам чуть позже. Внешний

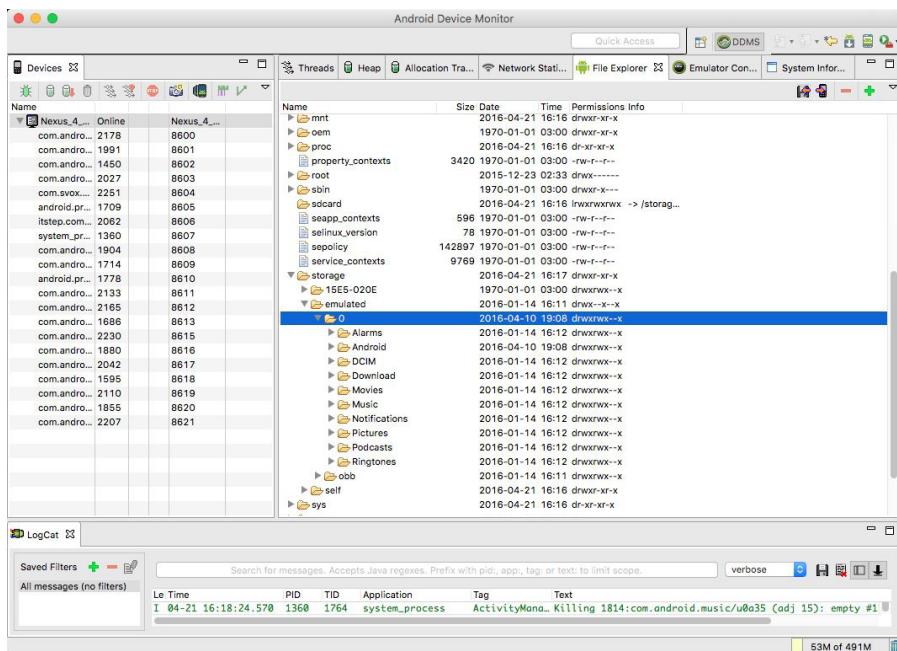


Рис. 5.2. Приложение «Android Device Monitor» с открытым Файловым Менеджером. Выделен каталог внешнего носителя

вид приложения «Android Device Monitor» с открытым Файловым Менеджером изображен на Рис. 5.2.

Как видно из Рис. 5.2, в каталоге внешнего носителя находятся какие-то файловые объекты (подкаталоги и файлы). В примере для текущего раздела (модуль «app5») добавим кнопку с надписью «Список файлов» и идентификатором **R.id.btnAddFiles**. По клику на эту кнопку приложение будет показывать в Диалоговом окне список файлов и подкаталогов, расположенных на внешнем носителе. Программный код обработчика события клика на кнопку «Список файлов» (идентификатор **R.id.btnAddFiles**) находится в методе класса Активности **btnListFilesClick** и отображен в Листинге 5.5:

Листинг 5.5. Отображение списка каталогов и файлов, находящихся на внешнем носителе

```
/*
 * Обработчик события клика на кнопку "Список файлов".
 * Метод получает список файлов и подкаталогов
 * внешнего носителя и выводит их в Диалоговом окне.
 * @param v- Ссылка на виджет, который является
 * источником события.
 */
public void btnListFilesClick(View v)
{
    if (this.getExternalStorageReadable())
    {
        //-- Получение пути к каталогу внешнего носителя --
        File esMainDir = Environment.
            getExternalStorageDirectory();
        Log.d(MainActivity.TAG, "Путь к каталогу
            внешнего носителя : " +
            esMainDir.getAbsolutePath());
    }
}
```

```
//--- Получение списка файлов и подкаталогов -----
ArrayList<String> listFiles = new ArrayList<>();
File[] arrFiles = esMainDir.listFiles();
if (arrFiles != null)
{
    for (File f : arrFiles)
    {
        if (f.isDirectory())
        {
            listFiles.add("[" + f.getName() + "]");
        }
        else
        {
            listFiles.add(f.getName());
        }
    }
}
else
{
    Toast.makeText(this, "Каталог внешнего
носителя пуст!", Toast.LENGTH_SHORT).
show();
}

//--- Вывод в Log найденных на внешнем носителе
//--- файлов и каталогов -----
for (int i = 0; i < listFiles.size(); i++)
{
    Log.d(TAG, listFiles.get(i));
}

//--- Создание объекта AlertDialog.Builder
//--- для формирования Диалогового окна -----
AlertDialog.Builder builder = new AlertDialog.
Builder(this, android.R.style.
Theme_Holo_Light_Dialog);
```

```
builder.setTitle("Список файлов и каталогов  
внешнего носителя");  
  
//--- Содержимым Диалогового окна будет список  
//--- android.widget.ListView -----  
    ListView LV = new ListView(this);  
    ArrayAdapter<String> adapter = new ArrayAdapter<>(  
        this, android.R.layout.  
            simple_list_item_1, listFiles);  
    LV.setAdapter(adapter);  
    builder.setView(LV);  
  
//--- Одна кнопка «Закрыть» - без обработчика события  
    builder.setNegativeButton("Закрыть", null);  
  
//--- Формируем Диалоговое окно и показываем его --  
    AlertDialog dialog = builder.create();  
    dialog.show();  
}  
else  
{  
    Toast.makeText(this, "Ошибка: Внешний носитель  
не готов!", Toast.LENGTH_SHORT).show();  
}  
}
```

Внешний вид работы примера из Листинга 5.5 изображен на Рис. 5.3.

Как видно из Листинга 5.5 и Рис. 5.3, названия каталогов помещены в квадратные скобки «[]», чтобы их можно было отличить от названий файлов.

Важно! В процессе работы над примером из Листинга 5.5 выяснилась одна неожиданно неприятная особенность! Начиная с версии API 23 и выше, недостаточно указывать

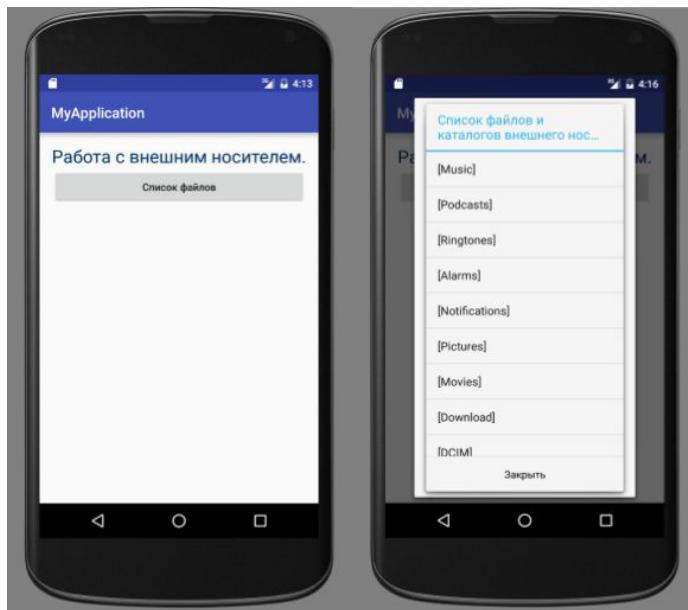


Рис. 5.3. Внешний вид работы примера из Листинга 5.5

разрешения на операции «чтение/запись» или «только чтение» в Манифесте приложения (см. Листинг 5.1 и 5.2). Необходимо делать еще дополнительные действия программно! А именно: необходимо программно определять, что приложение имеет необходимые разрешения на файловые операции с внешним носителем. Если таких разрешений у приложения нет, то необходимо запросить у пользователя подтверждения этих разрешений при первом запуске приложения. Если пользователь соглашается, то необходимо программно сообщить Операционной Системе Android, что приложению требуются разрешения на файловые операции с внешним носителем. Такой программный код необходимо выполнить в методе **onCreate** класса Активности. Пример этого кода приведен в Листинге 5.6.

Листинг 5.6. Необходимая для API 23+ программная проверка наличия у приложения разрешений на файловые операции с внешним носителем, и в случае их отсутствия, установка этих разрешений в Операционной Системе при подтверждении пользователем

```
@Override
protected void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    /*
     * Проверка, что у приложения есть разрешения
     * на чтение/запись файлов на внешнем носителе!
     * -----
     */
    int permission = ActivityCompat.
        checkSelfPermission(this,
            Manifest.permission.
                WRITE_EXTERNAL_STORAGE);
    if (permission != PackageManager.PERMISSION_GRANTED)
    {
        //-- Разрешения нет, запросим это разрешение
        //-- у пользователя -----
        ActivityCompat.requestPermissions(
            this,
            new String[]
            {
                Manifest.permission.
                    READ_EXTERNAL_STORAGE,
                Manifest.permission.
                    WRITE_EXTERNAL_STORAGE
            }, 1);
    }
}
```

Результат первого запуска приложения, определяющего отсутствие разрешений на файловые операции

с внешним носителем и запрашивающий подтверждения пользователя на эти разрешения (Листинг 5.6) изображен на Рис. 5.4.

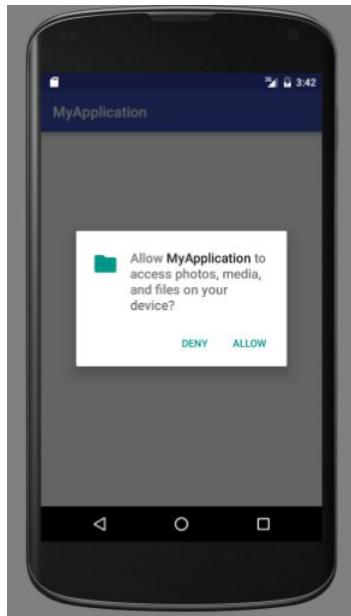


Рис. 5.4. Первый запуск приложения, использующего программный код из Листинга 5.6

5.1. Общедоступные каталоги внешнего носителя

Вернемся к нашему примеру из Листинга 5.5. Исполнение этого кода вывело следующий список каталогов:

- [Music]
- [Podcasts]
- [Ringtones]

- [Alarms]
- [Notifications]
- [Pictures]
- [Movies]
- [Download]
- [DCIM]
- [Android]

Этот список каталогов считается стандартным набором каталогов находящихся на внешнем носителе. Это так называемые «Общедоступные Каталоги». Каждый из этих Общедоступных каталогов имеет свое предназначение и наше приложение, в большинстве случаев, будет работать с файлами, находящимися именно в этих стандартных общедоступных каталогах. Хотя, приложение может создать и свой специальный каталог на внешнем носителе, если это необходимо.

Получить путь (в виде объекта **java.io.File**) к Общедоступному каталогу можно с помощью статического метода класса **android.os.Environment**:

```
public static File getExternalStoragePublicDirectory  
    (String type);
```

Этот метод возвращает объект класса **java.io.File**, который содержит в себе путь к Общедоступному каталогу на внешнем носителе. Метод принимает строковое значение, содержащее код Общедоступного каталога. Коды Общедоступных каталогов объявлены в виде констант класса **android.os.Environment**:

- **Environment.DIRECTORY_MUSIC** — стандартный каталог, который предназначен для размещения любых аудиофайлов, которые отображаются в списке музыкальных композиций пользователя.
- **Environment.DIRECTORY_PODCASTS** — стандартный каталог, который предназначен для размещения любых аудиофайлов которые должны отображаться в списке подкастов для дальнейшего выбора их пользователем. Эти аудиофайлы не принадлежат списку «регулярных» музыкальных композиций пользователя.
- **Environment.DIRECTORY_RINGTONES** — стандартный каталог, который предназначен для размещения любых аудиофайлов, которые пользователь может выбрать в качестве рингтонов.
- **Environment.DIRECTORY_ALARMS** — стандартный каталог, который предназначен для размещения любых аудиофайлов, которые пользователь может выбирать в качестве сигналов будильника.
- **Environment.DIRECTORY_NOTIFICATIONS** — стандартный каталог, который предназначен для размещения любых аудиофайлов, которые пользователь может выбирать в качестве сигналов для различных уведомлений.
- **Environment.DIRECTORY_PICTURES** — стандартный каталог, в который размещаются файлы изображений которые доступны пользователю.
- **Environment.DIRECTORY_MOVIES** — стандартный каталог, в который размещаются видеофайлы доступные для пользователя.

- **Environment.DIRECTORY_DOWNLOADS** — стандартный каталог, в который помещаются загруженные пользователем файлы.
- **Environment.DIRECTORY_DCIM** — каталог для традиционного расположения фотографий и видеофайлов, которые отсняты камерой устройства.
- **Environment.DIRECTORY_DOCUMENTS** — стандартный каталог, который предназначен для размещения документов, созданных пользователем.

Внимание! Каталог на внешнем носителе, который вернет метод **getExternalStoragePublicDirectory**, может не существовать! Поэтому, перед началом работы с этим Общедоступным каталогом необходимо проверить его существование и в случае его отсутствия приложение должно его самостоятельно создать. Пример показан в Листинге 5.7:

Листинг 5.7. Получение пути к Общедоступному каталогу и создание этого каталога на внешнем носителе, если этот каталог ранее не был создан

```
File esPicData = Environment.  
        getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_PICTURES);  
if (esPicData.exists() == false)  
{  
    esPicData.mkdir();  
}
```

5.2. Создание файлов на внешнем носителе

В наш пример в модуле «app5» добавим кнопку «Создать файл» с идентификатором `R.id.btnCreateFile`. При нажатии на эту кнопку, будет появляться Диалоговое окно `android.app.AlertDialog` со списком одиночного выбора содержащим список Общедоступных каталогов внешнего носителя и двумя текстовыми полями `android.widget.EditText`. В списке одиночного выбора `android.widget.ListView` (идентификатор `R.id.lvDirList`) пользователь будет выбирать каталог, в котором он хочет создать файл. В первом текстовом поле `android.widget.EditText` (идентификатор `R.id.etFileName`) пользователь будет вводить имя создаваемого файла, а во втором текстовом поле `android.widget.EditText` (идентификатор `R.id.etFileContent`) пользователь будет вводить текст, который будет записан в создаваемый файл. Файлы в текущем примере раздела будут создаваться текстовые и размещаться будут в любом Общедоступном каталоге, который выберет пользователь. Это нарушает логику использования Общедоступных каталогов по своему назначению, но ради этого примера мы эту логику нарушим.

Программный код примера этого раздела размещен в методе `btnCreateFileClick` класса Активности. Этот метод является обработчиком события нажатия на кнопку «Создать файл» (идентификатор `R.id.btnCreateFile`). Код этого метода приведен в Листинге 5.9.

Поскольку содержимое Диалогового окна в нашем примере является сложным набором виджетов, то создадим файл ресурсов xml макета раскладки для этого набора. Файл называется `/res/layout/create_file_dialog_content.xml`. Содержимое этого xml макета представлено в Листинге 5.8:

Листинг 5.8. Xml верстка набора виджетов для содержимого Диалогового окна android.app.AlertDialog рассматриваемого примера из Листинга 5.9

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

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textStyle="italic"
        android:textSize="8pt"
        android:textColor="#003366"
        android:text="Выберите каталог:" />

    <ListView
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:id="@+id/lvDirList"
        android:background="#F0FFF0">
    </ListView>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textStyle="italic"
        android:textSize="8pt"
        android:textColor="#003366"
        android:text="Введите название файла:" />
```

```
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/etFileName" />  
  
<TextView  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_gravity="center_horizontal"  
    android:textStyle="italic"  
    android:textSize="8pt"  
    android:textColor="#003366"  
    android:text="Введите содержимое файла:" />  
  
<EditText  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:id="@+id/etFileContent" />  
  
</LinearLayout>
```

Внешний вид Диалогового окна с содержимым, которое формируется на основе xml макета из Листинга 5.8 изображено на Рис. 5.5.

Программный код создания Диалогового окна и обработки события нажатия на кнопку «Создать» Диалогового окна приведен в Листинге 5.9. Этот пример интересен еще и тем, что является полезной иллюстрацией к теме «Диалоговые окна» текущего урока. Полезность заключается в извлечении из Диалогового окна значений, которые ввел пользователь в сложный набор виджетов, который является контентом Диалогового окна. Причем набор виджетов создается с помощью объекта `android.view.LayoutInflater` (<http://developer.android.com/intl/>)

ru/reference/android/view/LayoutInflater.html), который рассматривался в наших прошлых уроках.

Листинг 5.9. Создание и отображение Диалогового окна для создания нового текстового файла в любом доступном каталоге на внешнем носителе

```
/**  
 * Обработчик события клика на кнопку "Создать файл".  
 * Метод отображает Диалоговое окно со списком  
 * одиночного выбора  
 * android.widget.ListView и двумя текстовыми полями  
 * android.widget.EditText.  
 * В списке ListView пользователь выбирает каталог,  
 * в котором он хочет создать файл.  
 * В текстовых полях пользователь вводит название  
 * создаваемого файла и текстовое  
 * содержимое создаваемого файла.  
 * @param v- Ссылка на виджет, который является  
 * источником события.  
 */  
public void btnCreateFileClick(View v)  
  
{  
    if (this.isExternalStorageWritable())  
    {  
        //-- Получение пути к каталогу внешнего носителя --  
        File esMainDir = Environment.  
            getExternalStorageDirectory();  
  
        //-- Получение списка файлов и подкаталогов -----  
        ArrayList<String> listFiles = this.  
            fillDirectory(esMainDir);  
        //-- Создание объекта AlertDialog.Builder  
        //-- для формирования Диалогового окна
```

```
AlertDialog.Builder builder =
        new AlertDialog.Builder(
            this, android.R.style.
            Theme_Holo_Light_Dialog);
builder.setTitle("Создать файл");

//--- Создание виджета для содержимого Диалогового
//--- окна с помощью Inflater -----
LayoutInflater inflater = this.getLayoutInflater();
View view = inflater.inflate(
    R.layout.create_file_dialog_content,
    null, false);

//--- Создание Адаптера данных для списка
//--- ListView из контента для Диалога -----
ArrayAdapter<String> A = new ArrayAdapter<>(
    this,
    android.R.layout.
    simple_list_item_single_choice,
    listFiles);

ListView lvDirList = (ListView) view.
    findViewById(R.id.lvDirList);
lvDirList.setAdapter(A);

//--- Сообщаем, что ListView это список одиночного
//--- выбора -----
lvDirList.setChoiceMode(ListView.
    CHOICE_MODE_SINGLE);

//--- Назначаем контент Диалоговому окну -----
builder.setView(view);

//--- Кнопки "Создать" и "Отменить"
//--- для Диалогового окна -----
builder.setPositiveButton("Создать",
    new DialogInterface.OnClickListener()
```

```

{
    @Override
    public void onClick(DialogInterface dialog,
    int which)
    {
        File esMainDir = Environment.
            getExternalStorageDirectory();

    //--- Определяем выбранный пользователем каталог ---
        ListView lvDirList = (ListView)
            ((AlertDialog) dialog).
            findViewById(R.id.lvDirList);

        int index = lvDirList.
            getCheckedItemPosition();
        String strDir= "";
        if (index != -1)
        {
            strDir = lvDirList.getAdapter().
                getItem(index).toString();
            strDir = strDir.replaceAll("[\\\"
                [\\\"]]", "");
        }
        File dir = (strDir.isEmpty())?esMainDir:
            (new File(esMainDir, strDir));

    //--- Читаем из текстового поля название
    //--- создаваемого файла -----
        EditText etFileName = (EditText)
            ((AlertDialog) dialog).
            findViewById(R.id.etFileName);
        String strFileName = etFileName.
            getText().toString();
        if (strFileName.isEmpty())
        {

    //--- Если пользователь не ввел имя файла -
    //--- создадим случайное название -----

```

```
        strFileName = "noname" +
        ((int)(Math.random() * 100))
        +".txt";
    }

//-- Читаем из текстового поля содержимое
//-- для записи в создаваемый файл -----
EditText etFileCont = (EditText)
        ((AlertDialog) dialog).
        findViewById(R.id.etFileContent);
String strContent = etFileCont.getText().
        toString();
if (strContent.isEmpty())
{
//-- Если пользователь не ввел содержимое файла,
//-- то будет строка "no content" -----
        strContent = "no content";
}

//-- Создаем файл и записываем в него содержимое --
try
{
    File fileName = new File(dir, strFileName);

    FileWriter f = new FileWriter(fileName);
    f.write(strContent + "\r\n");
    f.flush();
    f.close();

    Toast.makeText(MainActivity.this,
    "Файл создан успешно: \n" + strFileName,
    Toast.LENGTH_LONG).show();
}
catch (IOException ioe)
{
    Toast.makeText(MainActivity.this,
    "Ошибка записи в файл: \n" +
    ioe.getMessage(),
    Toast.LENGTH_SHORT).show();
}
```

```
        Log.d(MainActivity.TAG,
            "Ошибка записи в файл: " +
            ioe.getMessage());
    }
}
});
builder.setNegativeButton("Отменить", null);

//-- Создание объекта android.app.AlertDialog ----
AlertDialog dialog = builder.create();

//-- Отображение Диалогового окна -----
dialog.show();
}

else
{
    Toast.makeText(this, "Ошибка: Внешний носитель
        не готов!", Toast.LENGTH_SHORT).show();
}
}
```

Давайте рассмотрим, что происходит в Листинге 5.9. Сама последовательность создания Диалогового окна **android.app.AlertDialog** вам уже известна и здесь объяснять ее еще раз не будем. Особенностью примера является то, что содержимое для Диалогового окна, которое является набором из нескольких виджетов, формируется с помощью объекта **android.view.LayoutInflater**. Это делается для того, чтобы назначить списку **android.widget.ListView** (идентификатор **R.id.lvDirList**) Адаптер данных **ArrayAdapter<T>**, который содержит список обнаруженных каталогов на внешнем носителе, а так же назначить виджет для отображения элементов списка **android.R.simple_list_item_single_choice**. Кроме этого,

списку **android.widget.ListView** (**R.id.lvDirList**) необходимо сообщить, что он является списком одиночного выбора, чтобы пользователь смог выбрать интересующий его каталог.

В обработчике события нажатия на кнопку «Создать», которая расположена на Диалоговом окне, происходит считывание введенной пользователем информации. Для этой цели необходимо получить ссылки на виджеты (список **android.widget.ListView** и два текстовых поля **android.widget.EditText**), которые были размещены в качестве содержимого Диалогового окна. Ссылки были получены с помощью метода **findViewById** который был применен к Диалоговому окну:

```
EditText etFileName = (EditText)
    ((AlertDialog) dialog).findViewById(R.id.etFileName);
```

После получения ссылок на виджеты, извлечение из них информации, которую ввел пользователь, а так же проверка правильности ввода пользователем данных, не составляет труда. Сам программный код создания файла и запись в файла текста, выделен в Листинге 5.9 жирным шрифтом. Как видно, для записи содержимого в файл используются обычные файловые потоки Java.

Внешний вид работы примера из Листинга 5.9 изображен на Рис. 5.5.

На Рис. 5.5 изображено, что пользователь создает файл с именем **test_file.txt** в Общедоступном каталоге **Notifications**. После успешного создания приложением этого файла в эмуляторе, в «Android Device Monitor» можно увидеть (и при желании скачать оттуда) созданный

файл **test_file.txt**. Фрагмент приложения «Android Device Monitor» с созданным файлом изображен на Рис. 5.6.

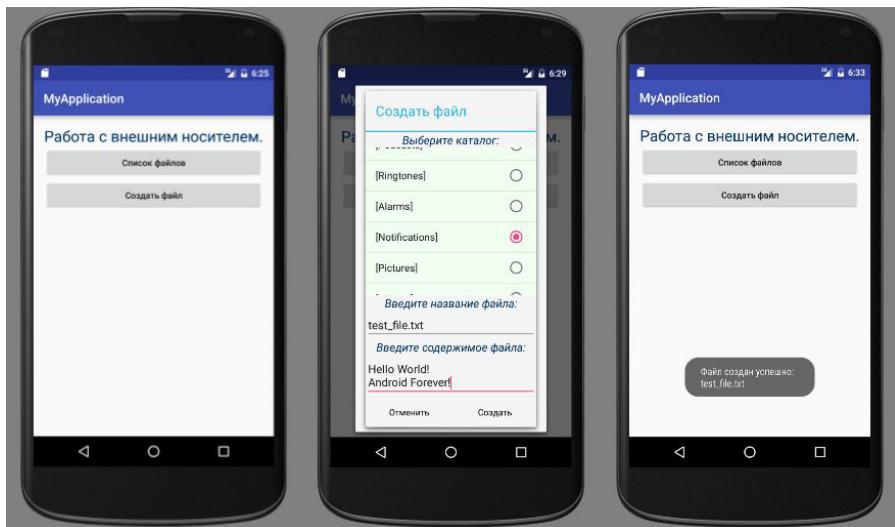


Рис. 5.5. Внешний вид работы примера из Листинга 5.9

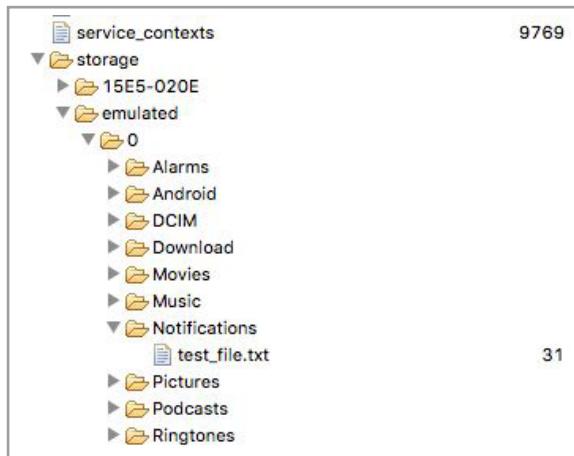


Рис. 5.6. В «Android Device Monitor» отображается файл, созданный приложением из Листинга 5.9

Конечно же, в данном примере (Листинг 5.9) пользователь выбирал интересующий его каталог из списка уже существующих на внешнем носителе Общедоступных каталогов, который отображался в Диалоговом окне. При этом, во время создания файла не делалось никаких проверок на то, что Общедоступный каталог существует на диске — ведь он уже был найден и показан пользователю. Но, как уже упоминалось выше в данном уроке, бывают ситуации, что при попытке записать файл в какой-то Общедоступный каталог оказывается, что такой каталог не создан. Поэтому, необходимо обязательно осуществлять проверку на существование требуемого Общедоступного каталога на внешнем носителе. И если каталог не существует — то его необходимо создать. В Листинге 5.10 приведен пример создания и записи файла с проверкой на существование Общедоступного каталога и попыткой его создания в случае, если каталог отсутствует.

Листинг 5.10. Обращение к Общедоступному каталогу для создания в нем файла с предварительной проверкой на существование этого Общедоступного каталога

```
File esDocDir = Environment.  
    getExternalStoragePublicDirectory(  
        Environment.DIRECTORY_DOCUMENTS);  
  
if (esDocDir.exists() == false)  
{  
    esDocDir.mkdir();  
}  
  
try  
{  
    File fileName = new File(esDocDir, "example.txt");  
}
```

```
FileWriter f = new FileWriter(fileName);
f.write("Hello World!\r\n");
f.flush();
f.close();

Toast.makeText(MainActivity.this, "Файл создан
успешно!",
Toast.LENGTH_LONG).show();
}

catch (IOException ioe)
{
    Toast.makeText(MainActivity.this,
    "Ошибка записи в файл: \n" + ioe.
getMessage(),
    Toast.LENGTH_SHORT).show();
    Log.d(MainActivity.TAG,
    "Ошибка записи в файл: " + ioe.
getMessage());
}
```

Исходный пример данного раздела можно найти в модуле «app5» файлов с исходным кодом, прилагаемых к данному уроку.

5.3. Практический пример: Диалоговое окно для выбора и открытия файлов, находящихся на внешнем носителе

Завершим тему работы с файлами на внешнем носителе еще одним примером, который будет еще и по совместительству примером на Диалоговое окно. В предыдущем разделе так же был пример с Диалоговым окном, в котором отображались каталоги, однако, пользователь не имел возможности заходить в эти каталоги и подкаталоги каталогов. Так же пользователь в примере текущего раздела сможет

выбирать файлы с помощью окна, которое будет создано. И выбранные файлы будут открываться и отображаться в приложении. И еще одна важная особенность примера текущего раздела заключается в том, что Диалоговое окно `android.app.AlertDialog` будет создано один единственный раз! То есть не будет создаваться заново с помощью объекта `android.app.AlertDialog.Builder`. Во всех предыдущих примерах на Диалоговые окна, которые рассматривались в данном уроке, как только возникала необходимость отобразить Диалоговое окно `android.app.AlertDialog` происходило создание объекта `android.app.AlertDialog.Builder`, с помощью которого настраивался внешний вид создаваемого Диалогового окна, а затем создавалось само окно как объект класса `android.app.AlertDialog`. Другими словами, проделывалась одна и та же работа многократно. И возможно такая работа проделывалась напрасно, так как создавались одинаковые объекты `android.app.AlertDialog`. Так вот, для случая когда Диалоговое окно предназначено для частого отображения в приложении и внешний вид этого окна один и тот же, имеет смысл создать такое окно один раз (в обработчике события `onCreate`), а затем это окно необходимо будет только отображать. То есть, будет экономиться время центрального процессора, однако такое окно будет занимать место в оперативной памяти и следовательно, такой подход более ресурсоемкий, что не приветствуется при создании мобильных приложений. Выбирать ли разработчику при работе с Диалоговыми окнами такой подход или нет, должен решать сам разработчик, предварительно хорошо взвесив все плюсы и минусы такого подхода.

В нашем примере (модуль «app5») добавим еще одну кнопку «Открыть файл» с идентификатором **R.id.OpenFile** и текстовое поле **android.widget.EditText** (идентификатор **R.id.etEditFile**), в котором будет отображаться содержимое выбранного пользователем файла. Обработчик нажатия на кнопку «Открыть файл» называется **btnOpenFileClick**.

Для данного примера были созданы дополнительные поля в классе Активности. Объявление этих полей представлено в Листинге 5.11:

Листинг 5.11. Дополнительные поля, необходимые для рассматриваемого в данном разделе примера

```
/*
 * Поля, относящиеся к примеру на Диалоговое окно
 * android.app.AlertDialog для выбора и открытия
 * файлов, находящихся на внешнем носителе.
 *
 */
/***
 * Виджет android.widget.ListView, который будет
 * размещаться в Диалоговом окне выбора файлов
 * android.app.AlertDialog и содержать список
 * файлов и каталогов внешнего носителя.
 */
private ListView lvFiles;

/**
 * Адаптер данных для списка lvFiles.
 */
private ArrayAdapter<String> adapter;
/***
 * Текущий отображаемый в списке lvFiles каталог
 * внешнего носителя.
 */

```

```
private File esCurDir;  
  
/**  
 * Текущий открытый пользователем файл,  
 * который отображается в приложении.  
 */  
private File esCurFile;  
  
/**  
 * Ссылка на Диалоговое окно android.app.  
 * AlertDialog, которое будет предоставлять  
 * пользователю возможность перемещаться по  
 * каталогам внешнего носителя и выбирать файлы  
 * для отображения их содержимого в приложении.  
 */  
private AlertDialog dialog;
```

Итак, давайте рассмотрим, что же это за поля:

- **AlertDialog dialog** — это непосредственно Диалоговое окно текущего примера, которое будет создано один раз в методе **onCreate** и показываться многократно по необходимости. В этом окне будет находиться список **android.widget.ListView**, в котором будут отображаться каталоги и файлы текущего отображаемого каталога внешнего носителя.
- **ListView lvFiles** — это список который будет располагаться в качестве контента Диалогового окна и будет содержать файлы и каталоги, которые будет открывать пользователь.
- **ArrayAdapter<String> adapter** — адаптер данных для списка **lvFiles**.
- **File esCurDir** — текущий отображаемый в Диалоговом окне **dialog** каталог.

- **File esCurFile** — текущий открытый пользователем файл, содержимое которого отображается в приложении.

Создание Диалогового окна **android.app.AlertDialog** осуществляется в методе **onCreate** и приведено в Листинге 5.12:

Листинг 5.12. Создание Диалогового окна многоразового использования для выбора файлов, находящихся на внешнем носителе

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

    /*
     * Проверка, что у приложения есть разрешения на
     * чтение/запись файлов на внешнем носителе!
     * Если разрешения нет - программно его создать,
     * предварительно запросив подтверждение пользователя.
     * -----
     */
    int permission = ActivityCompat.checkSelfPermission(this,
            Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if (permission != PackageManager.PERMISSION_GRANTED)
    {
        //-- Разрешения нет, запросим это разрешение
        //-- у пользователя -----
        ActivityCompat.requestPermissions(
            this,new String[]
            {
                Manifest.permission.READ_EXTERNAL_STORAGE,
                Manifest.permission.WRITE_EXTERNAL_STORAGE
            }, 1);
    }
}
```

```
/*
 * Формирование Диалогового окна android.app.AlertDialog
 * для открытия файлов.
 * Это Диалоговое окно, в котором будет отображаться
 * список названий каталогов и файлов, находящихся
 * на внешнем носителе. Пользователь может
 * перемещаться по каталогам внешнего носителя.
 */
//-- Получение пути к каталогу внешнего носителя --
this.esCurDir = Environment.
    getExternalStorageDirectory();

//-- Получение списка каталогов и файлов
//-- внешнего носителя -----
ArrayList<String> listFiles = this.
    fillDirectory(this.
        esCurDir);

//-- Формирование android.app.AlertDialog -----
AlertDialog.Builder builder = new AlertDialog.
    Builder(this, android.R.style.
        Theme_Holo_Light_Dialog);
//-- В заголовке окна будет отображать каталог,
//-- в котором сейчас находимся -----
builder.setTitle(this.esCurDir.getAbsolutePath());

//-- Список с именами файлов и каталогов -----
//-- для размещения в Диалоговом окне -----
this.lvFiles = new ListView(this);
this.adapter = new ArrayAdapter<>(
    this, android.R.layout.
        simple_list_item_1, listFiles);
this.lvFiles.setAdapter(this.adapter);

//-- Обработчик события клика по элементу списка --

```

```
this.lvFiles.setOnItemClickListener(new
AdapterView.OnItemClickListener()
{
    @Override
    public void onItemClick(AdapterView<?> parent,
                        View view, int position, long id)
    {
        String item = MainActivity.this.adapter.
                        getItem(position);
        item = item.replaceAll("[\\\\[\\\\]]", " ");

        //-- Если пользователь кликнул по [...], то выходим
        //-- в родительский каталог -----
        File fileObj = (item.compareTo(..) == 0)?
                        (MainActivity.this.esCurDir.
                        getParentFile()):
                        (new File(MainActivity.this.
                        esCurDir, item));

        if (fileObj.isDirectory())
        {
            //-- Пользователь кликнул по каталогу -
            //-- зайдем в него -----
            MainActivity.this.adapter.clear();
            ArrayList<String> listFiles =
                MainActivity.this.fillDirectory(fileObj);

            //-- Для корневого каталога внешнего носителя
            //-- не добавляем [...] -----
            if (fileObj.compareTo(Environment.
                        getExternalStorageDirectory()) != 0)
            {
                MainActivity.this.adapter.add(..);
            }

            MainActivity.this.adapter.addAll(listFiles);
            MainActivity.this.esCurDir = fileObj;
        }
    }
})
```

```
>MainActivity.this.dialog.setTitle(  
        MainActivity.this.esCurDir.  
        getAbsolutePath());  
    }  
    else  
        if (fileObj.isFile())  
        {  
  
        //-- Пользователь выбрал файл - прочитаем этот файл --  
        try  
        {  
            LineNumberReader LR =  
                new LineNumberReader(  
                    new FileReader(fileObj));  
            String S = "";  
            while (true)  
            {  
                String z = LR.readLine();  
                if (z == null) break;  
                S += z + "\n";  
            }  
            LR.close();  
  
        //-- Отобразим содержимое файла в виджете  
        //-- android.widget.EditText -----  
            EditText etEditFile = (EditText)  
                MainActivity.this.findViewById(R.  
                    id.etEditFile);  
            etEditFile.setText(S);  
  
        //-- Запомним путь к файлу -----  
            MainActivity.this.esCurFile = fileObj;  
        }  
        catch (Exception e)  
        {
```

```
        Toast.makeText(MainActivity.this,
        "Ошибка открытия файла : \n" +
        e.getMessage(), Toast.LENGTH_LONG).show();

        Log.d(MainActivity.TAG,
        "Ошибка открытия файла : " +
        e.getMessage());
    }

//--- Программно закроем Диалоговое окно -----
MainActivity.this.dialog.dismiss();

        Toast.makeText(MainActivity.this,
        fileObj.getAbsolutePath(),
        Toast.LENGTH_LONG).show();
    }
}
});

//--- Делаем содержимым Диалогового окна список
//--- android.widget.ListView -----
builder.setView(this.lvFiles);

//--- Кнопки Диалогового окна -----
builder.setNegativeButton("Закрыть", null);

//--- Создание Диалогового окна - один раз
//--- в приложении -----
this.dialog = builder.create();
}
```

Созданное в методе **onCreate** Диалоговое окно **dialog** предназначено для многоразового использования в приложении. Это окно отображается в методе **btnOpenFileDialog**, который обрабатывает событие нажатия на кнопку «Открыть файл». Программный код этого метода см. в Листинге 5.13:

Листинг 5.13. Отображение Диалогового окна для выбора файлов

```

/**
 * Обработчик события клика на кнопку "Открыть файл".
 * Метод отображает Диалоговое окно для перемещения
 * по каталогам внешнего носителя и выбора файла
 * для отображения его содержимого в приложении.
 * Диалоговое окно, отображаемое в этом методе,
 * создается один раз в методе onCreate.
 * @param v - Ссылка на виджет, который является
 * источником события.
 */
public void btnOpenFileClick(View v)
{
    if (this.getExternalStorageWritable())
    {
        this.dialog.show();
    }
    else
    {
        Toast.makeText(this,
                      "Внешний носитель не готов",
                      Toast.LENGTH_SHORT).show();
    }
}

```

Как видно из Листинга 5.13, ранее созданное Диалоговое окно **android.app.AlertDialog**, отображается всего лишь вызовом одного метода **show()**.

Открытие и чтение файла (Листинг 5.12), который выбрал пользователь, осуществляется в методе **onItemClickListener**, который является обработчиком клика по элементу списка **android.widget.ListView**. Как только пользователь кликает по элементу списка **android.widget.ListView**, происходит определение — выбран каталог или

файл. Если пользователь кликнул по каталогу, то ему в этом же списке отображается содержимое выбранного каталога. Если пользователь выбрал файл, то этот файл открывается, читается его содержимое и затем отображается на Активности приложения (Рис. 5.7). В Листинге 5.12 открытие файла и чтение содержимого файла выделено жирным шрифтом. Механизм чтения текстового файла вам должен быть знаком из курса Java.

Внешний вид работы примера из Листинга 5.12 и 5.13 на Рис. 5.7.

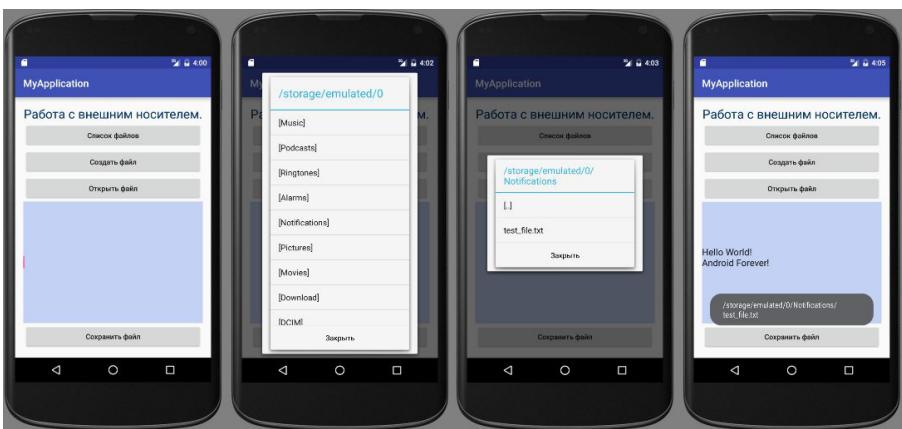


Рис. 5.7. Работа примера из Листинга 5.12

Пользователь нажимает на кнопку «Открыть файл», в появившемся окне выбирает каталог «Notifications», и затем в этом каталоге выбирает файл «test_file.txt». Содержимое файла отображается в текстовом поле `android.widget.EditText`.

В Листингах 5.12 и 5.9 используется метод `fillDirectory`. Этот метод предназначен для заполнения коллекции `ArrayList<String>` списком файлов и подкаталогов, которые

находятся в передаваемом в метод каталоге. Исходный код метода **fillDirectory** приведен в Листинге 5.14:

Листинг 5.14. Метод, получающий список файлов и подкаталогов для требуемого каталога.
Используется в Листингах 5.9 и 5.12

```
/**  
 * Метод возвращает коллекцию названий файлов  
 * и каталогов внешнего носителя, которые находятся  
 * в каталоге dir.@param dir - Каталог внешнего  
 * носителя, список файлов и подкаталогов которого  
 * необходимо получить.  
 * @return - Коллекцию любого размера строк.  
 * Имена каталогов  
 * заключены в [квадратные скобки].  
 */  
private ArrayList<String> fillDirectory(File dir)  
{  
    ArrayList<String> listFiles = new ArrayList<>();  
  
    if (this.isExternalStorageReadable())  
    {  
        File[] arrFiles = dir.listFiles();  
  
        if (arrFiles != null)  
        {  
            for (File f : arrFiles)  
            {  
                if (f.isDirectory())  
                {  
                    listFiles.add("[" + f.getName() + "]");  
                }  
                else  
                {  
                    listFiles.add(f.getName());  
                }  
            }  
        }  
    }  
    return listFiles;  
}
```

```
        }
    }
}
return listFiles;
}
```

Поскольку пользователь открывает содержимое файла в виджет **android.widget.EditText**, в котором он может модифицировать содержимое файла, то логично было бы предоставить ему возможность сохранить внесенные в файл изменения. Поэтому закончим пример текущего раздела функциональностью сохранения открытого файла. Для этого на Активности приложения добавлена кнопка с надписью «Сохранить файл» и идентификатором **R.id.btnSaveFile**. Метод, который обрабатывает события клика по этой кнопке называется **btnSaveFileClick**. Исходный код этого метода приведен в Листинге 5.15:

Листинг 5.15. Код метода, который обрабатывает событие нажатия на кнопку «Сохранить файл»

```
/**
 * Обработчик события клика на кнопку "Сохранить файл".
 * Метод выполняет сохранение содержимого ранее
 * открытого файла из виджета android.widget.EditText
 * (идентификатор R.id.etEditFile)
 * обратно в этот же файл на внешнем носителе.
 * @param v - Ссылка на виджет,
 * который является источником события.
 */
public void btnSaveFileClick(View v)
{
```

```
//-- Проверка, что файл был открыт -----
if (this.esCurFile == null)
{
    Toast.makeText(this, "Для сохранения необходимо
предварительно выбрать и открыть файл",
    Toast.LENGTH_LONG).show();
}

//-- Если внешний носитель доступен для записи ---
if (this.isExternalStorageWritable())
{
    //-- Получаем текст из текстового поля
    //-- android.widget.EditText -----
    EditText etEditFile = (EditText) this.
        findViewById(R.id.etEditFile);
    String content = etEditFile.getText().
        toString();

    //-- Пересоздаем файл и записываем в него содержимое --
    try
    {
        FileWriter f = new FileWriter(this.esCurFile);
        f.write(content + "\r\n");
        f.flush();
        f.close();
        Toast.makeText(MainActivity.this,
            "Файл сохранен успешно: \n" +
            this.esCurFile.getAbsolutePath(),
            Toast.LENGTH_LONG).show();
    }
    catch (IOException ioe)
    {
        Toast.makeText(MainActivity.this,
            "Ошибка записи в файл:
\n" + ioe.getMessage(),
            Toast.LENGTH_SHORT).show();
    }
}
```

```
        Log.d(MainActivity.TAG,
            "Ошибка записи в файл: " + ioe.getMessage());
    }
}
else
{
    Toast.makeText(this,
        "Внешний носитель не готов",
        Toast.LENGTH_SHORT).show();
}
}
```

Внешний вид работы примера из Листинга 5.15 представлен на Рис. 5.8.

Исходный код примера, рассматриваемого в этом разделе, находится в модуле «app5» среди файлов исходного кода, прилагаемых к данному уроку.

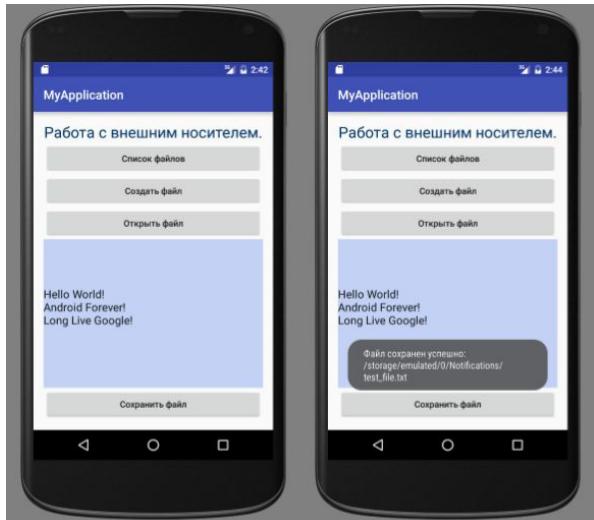


Рис. 5.8. Сохранение содержимого файла.
Внешний вид работы примера из Листинга 5.15

5.4. Еще несколько слов о работе с внешним носителем

В предыдущих разделах, посвященных работе с файлами на внешнем носителе, речь шла о каталогах, которые являются доступными для всех приложений, работающих на мобильном устройстве. Если какое-то приложение сохранит в этих каталогах на внешнем носителе некоторые свои файлы, а затем это приложение будет удалено с мобильного устройства, то сохраненные этим приложением файлы останутся на внешнем носителе. Это приводит к там называемому «загрязнению» внешнего носителя. Этот факт нужно помнить. Однако, Операционная Система Android предоставляет разработчику в пользование специальные каталоги на внешнем носителе, в которые приложение может сохранять свои файлы и при удалении приложения с мобильного устройства, эти файлы с внешнего носителя будут также удалены. Причем, эти файлы считаются собственностью того приложения, которое их создало и не видны пользователю как медиа файлы (далее в этом разделе эти каталоги будут называться «скрытыми каталогами»). Однако, любые другие приложения, имеющие права на запись файлов на внешний носитель (привилегия `WRITE_EXTERNAL_STORAGE`) могут записывать данные в эти файлы, находящиеся в скрытых каталогах нашего приложения.

Получить путь к каталогу, в котором приложение может сохранять свои файлы на внешнем носителе и при этом файлы не будут видны пользователю как медиа файлы, можно с помощью метода класса `android.content.Context`:

```
public File getExternalFilesDir (String type);
```

Параметр **type**, который принимает этот метод, это строковое значение для типа каталога. Это значение может быть `null`, что означает, что нам необходимо получить путь к корневому для нашего приложения скрытому каталогу на внешнем носителе. В Листинге 5.16 приведен пример получения пути к такому каталогу на эмулируемом устройстве.

Листинг 5.16. Получение пути к скрытому каталогу для нашего приложения на внешнем носителе

```
File rootExtFileDir = this.  
                    getExternalFilesDir(null);  
Log.d(TAG, rootExtFileDir.getAbsolutePath());
```

Результат исполнения примера из Листинга 5.16 выдал такой результат:

```
/storage/emulated/0/Android/data/itstep.com.myapp5/files
```

Как видите, каталог «Android» (о котором ранее не упоминалось в уроке, но мы его видели) на внешнем носителе предназначен для скрытых каталогов нашего приложения.

Остальные значения для параметра **type** существуют в виде строковых констант в классе `android.os.Environment` и уже упоминались в этом уроке:

- `Environment.DIRECTORY_MUSIC` — каталог `/storage/emulated/0/Android/data/itstep.com.myapp5/files/Music`

- **Environment.DIRECTORY_PODCASTS** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Podcasts
- **Environment.DIRECTORY_RINGTONES** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Ringtones
- **Environment.DIRECTORY_ALARMS** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Alarms
- **Environment.DIRECTORY_NOTIFICATIONS** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Notifications
- **Environment.DIRECTORY_PICTURES** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Pictures
- **Environment.DIRECTORY_MOVIES** — каталог /storage/emulated/0/Android/data/itstep.com.myapp5/files/Movies

Листинг 5.17. Получение пути к скрытому каталогу DIRECTORY_MUSIC для нашего приложения

```
File rootExtFileDir = this.  
        getExternalFilesDir(Environment.  
        DIRECTORY_MUSIC);  
Log.d(TAG, rootExtFileDir.getAbsolutePath());
```

Например, Листинг 5.17 выдаст такой путь к скрытому каталогу:

```
/storage/emulated/0/Android/data/itstep.com.myapp5/  
files/Music
```

Из любопытства, мы можем посмотреть на содержимое скрытых каталогов с помощью нашего Диалогового окна открытия файлов из модуля «app5» (Листинг 5.12). На Рис. 5.9 изображена последовательность перемещения по скрытым каталогам.

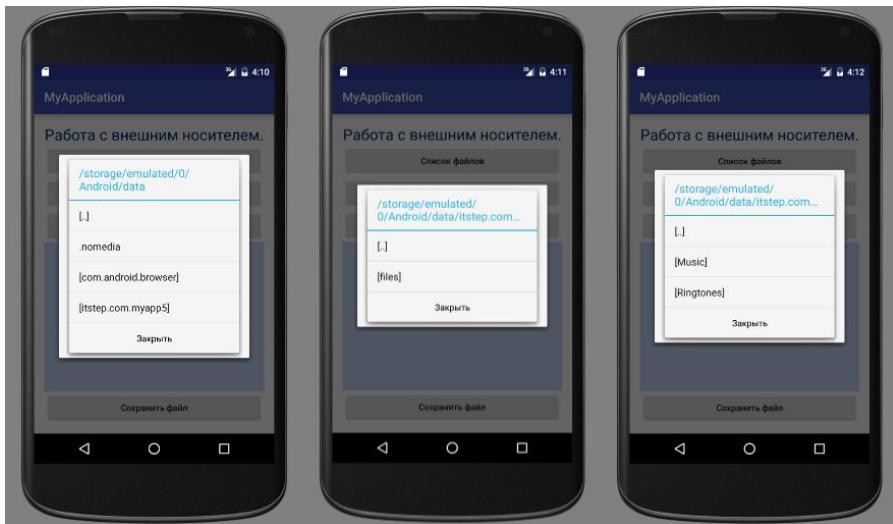


Рис. 5.9. Скрытые каталоги, на внешнем носителе, которые предназначены для нашего приложения

На Рис. 5.9 (самый левый скриншот) вы можете увидеть файл «.nomedia». Именно размещение в каталоге файла с таким именем делает каталог скрытым от пользователя. Судя по скриншоту из Рис. 5.9, каталог /storage/0/emulated/Android/data скрыт от пользователя. То есть, пользователь зайдя в каталог /storage/0/emulated/Android, не увидит там никаких файлов.

6. Работа с файлами на внутреннем носителе

Кроме внешнего носителя, приложению доступен еще и внутренний носитель. На внутреннем носителе приложение может создавать и сохранять файлы, которые доступны только этому приложению и никакому другому приложению эти файлы доступны не будут! Соответственно, при удалении приложения с мобильного устройства, эти файлы также будут удалены. Область внутреннего носителя, которая доступна приложению для создания собственных (приватных) файлов, для каждого приложения своя. Когда-то такая область называлась песочницей (sandbox). Однако, уже какое-то время на сайте для разработчиков Android приложений этот термин практически не встречается. Мы его тоже постараемся не использовать. Так вот, при работе с внутренним носителем, приложение имеет доступ только к своей области внутреннего носителя и все.

Для того чтобы создать файл для записи, необходимо воспользоваться функцией класса `android.content.Context` (<http://developer.android.com/intl/ru/reference/android/content/Context.html>):

```
public FileOutputStream openFileOutput (String name,  
                                         int mode);
```

Функция принимает следующие параметры:

- **String name** — название создаваемого файла.

- int mode — режим открытия файла. Может быть или Context.MODE_PRIVATE (файл создается, если отсутствует или пересоздается, если файл уже существует на внутреннем носителе) или Context.MODE_APPEND (файл создается, если отсутствует или файл открывается для добавления данных, если файл уже существует на внутреннем носителе).

Метод возвращает ссылку на объект байтового файлового потока для записи `java.io.FileOutputStream` (<http://developer.android.com/reference/java/io/FileOutputStream.html>), который должен быть уже вам знаком по курсу «Программирование на Java».

После открытия файла, для записи в него байт, используется функция класса `java.io.FileOutputStream`:

```
void write(byte[] buffer, int byteOffset,
          int byteCount);
```

Записав байты, файл необходимо закрыть с помощью функции класса `java.io.FileOutputStream`:

```
void close();
```

Для того, чтобы открыть файл для чтения из него данных, необходимо воспользоваться функцией класса `android.conent.Context`:

```
public FileInputStream openFileInput (String name);
```

Метод принимает название открываемого файла (`name`) и возвращает ссылку на объект байтового файлового потока для чтения `java.io.FileInputStream` (<http://developer>.

android.com/reference/java/io/FileInputStream.html), который должен быть уже вам знаком по курсу «Программирование на Java».

После открытия файла, для чтения из него байт, используется функция класса `java.io.FileInputStream`:

```
int read(byte[] buffer, int byteOffset,  
        int byteCount);
```

Прочитав байты, файл необходимо закрыть с помощью функции класса `java.io.FileInputStream`:

```
void close();
```

Для работы со своими собственными приватными файлами приложению не нужны никакие дополнительные привилегии и ничего в Манифесте приложения прописывать не нужно.

Пример создания файла и записи в него данных, а затем чтения данных из этого файла, представлен в Листинге 6.1.

Листинг 6.1. Запись данных в приватный файл приложения на внутреннем носителе и чтение данных из этого же файла

```
/*  
 * Запись данных в приватный файл приложения  
 * -----  
 */  
try  
{  
//-- Создание приватного файла приложения -----  
FileOutputStream FOS = this.openFileOutput(  
        "test.txt", Context.MODE_PRIVATE);
```

```
//-- Запись данных -----
String str = "Hello World!\nAndroid Forever!";
byte[] a = str.getBytes("UTF8");
FOS.write(a, 0, a.length);
FOS.close();
}
catch (FileNotFoundException fnfe)
{
//-- Генерируется методом openFileOutput -----
Log.d(TAG, "FileNotFoundException : " +
fnfe.getMessage());
}
catch (UnsupportedEncodingException uee)
{
//-- Генерируется методом getBytes -----
Log.d(TAG, "UnsupportedEncodingException : " +
uee.getMessage());
}
catch (IOException ioe)
{
//-- Генерируется методом write -----
Log.d(TAG, "IOException : " + ioe.getMessage());
}

/*
 * Чтение данных из приватного файла приложения
 */
try
{

//-- Открытие приватного файла приложения -----
FileInputStream FIS = this.openFileInput("test.txt");

//-- Чтение данных из файла -----
byte[] b = new byte[1024];
```

```

        int cnt = FIS.read(b, 0, b.length);
        FIS.close();

        String str = new String(b, 0, cnt, "UTF8");
        Log.d(TAG, "Прочитано из файла : " + str);

    }

    catch (FileNotFoundException fnfe)
    {
        //--- Генерируется методом openFileInput -----
        Log.d(TAG, "FileNotFoundException : " +
               fnfe.getMessage());
    }

    catch (IOException ioe)
    {
        //--- Генерируется методом read -----
        Log.d(TAG, "IOException : " + ioe.getMessage());
    }
}

```

В Листинге 6.1 подробно расписаны Исключительные ситуации, которые могут произойти при работе с файлами на внутреннем носителе.

Есть еще несколько методов класса **android.content.Context**, которые полезны при работе с приватными файлами на внутреннем носителе:

- **File getFilesDir()** — возвращает абсолютный файловый путь к каталогу приложения на внутреннем носителе, где хранятся приватные файлы приложения.
- **File getDir(String name, int mode)** — возвращает путь к каталогу на внутреннем носителе. При необходимости, этот метод может создать новый каталог.
- **boolean deleteFile(String name)** — удаляет приватный файл приложения на внутреннем носителе.

- `String[] fileList()` — получает список файлов и подкаталогов на внутреннем носителе.

Применение этих методов не представляет никакой сложности, поэтому примеры использования этих методов приводиться не будут. Вам рекомендуется самостоятельно ознакомиться с работой этих методов.

7. Каталог assets приложения

В данном уроке достаточно большое место занимает тема работы с файлами на мобильном устройстве. Дан- ный раздел урока дополнит тему файлов с точки зрения ресурсов приложения (подробно о ресурсах приложения будет рассказано в следующих уроках).

Каталог assets приложения предназначен для раз- мещения настоящих файлов внутри приложения. Это могут быть файлы изображений или звуковые файлы, которые необходимы приложению для своей полно- ценной работы. В этом разделе мы научимся размещать файлы в каталоге assets. А также извлекать содержимое этих файлов из каталога assets.

Для начала необходимо добавить каталог assets в про- ект (в модуль) приложения в Android Studio. Это делается с помощью клика правой кнопкой мыши по названию модуля в проекте Android Studio. В появившемся контекст- ном меню нужно выбрать пункты «New / Folder / Assets Folder» (см. Рис. 7.1). В появившемся диалоговом окне нажимаем «Finish». Каталог assets создан (см. Рис. 7.2).

Теперь с помощью обычного файлового менеджера необходи-мо скопировать в этот каталог файлы, содер- жимое которых после компиляции необходи-мо разместить в нашем приложении. Путь к каталогу assets будет сле- дующим: **имя_модуля/srs/main/assets**. В нашем примере (модуль «аррб») мы разместим в этот каталог обычный тек- стовый файл, назовем его test_assets.txt. В файле раз- местим строку «This is text file from assets directory!».

Урок №3

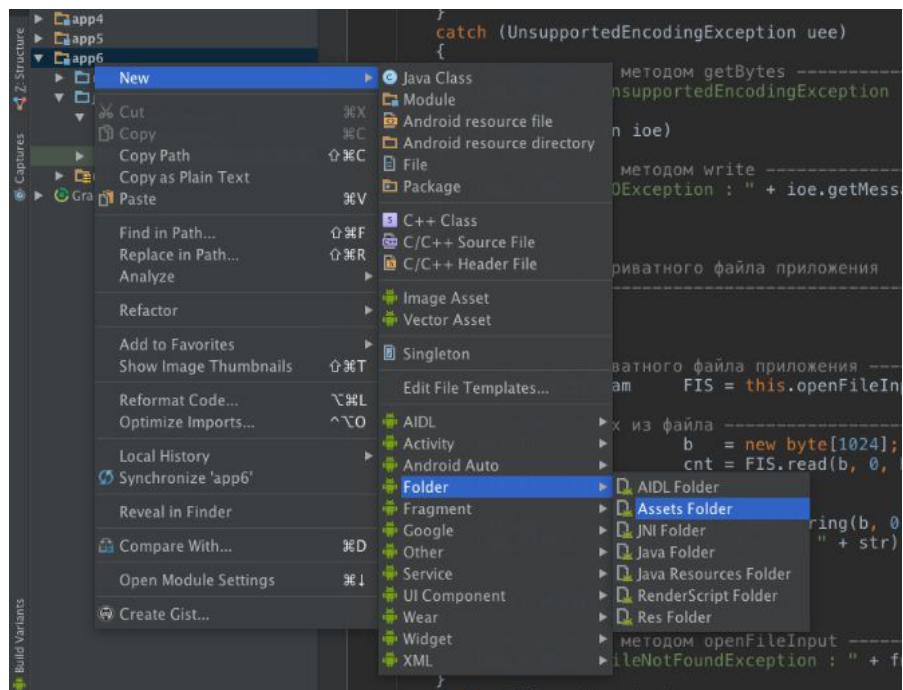


Рис. 7.1. Добавление каталога assets в модуль проекта приложения в Android Studio

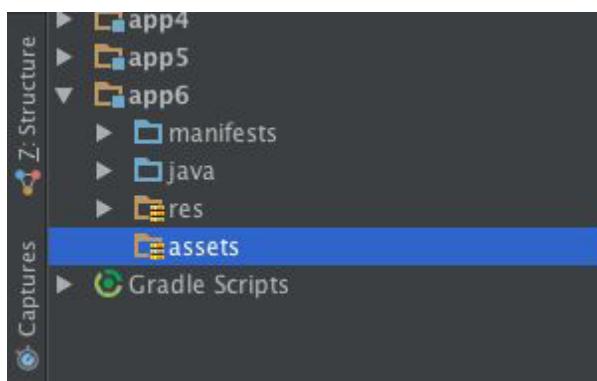


Рис. 7.2. Добавленный каталог assets в модуль приложения

Это необходимо для того, чтобы в нашем примере открыть файл, прочитать его содержимое и затем вывести это содержимое на экран (Рис. 7.3).

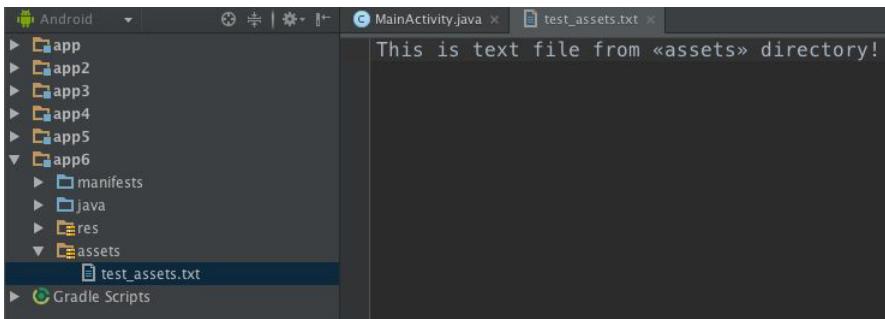


Рис. 7.3. Добавленный с помощью файлового менеджера в каталог assets файл test_assets.txt

Для того, чтобы получить доступ к файлам приложения, расположенных в каталоге assets предназначен класс `android.content.res.AssetManager` (<http://developer.android.com/intl/ru/reference/android/content/res/AssetManager.html>).

Для того, чтобы получить ссылку на объект `android.content.res.AssetManager` необходимо воспользоваться методом класса Активности:

```
AssetManager getAssets();
```

Далее, в классе `android.content.res.AssetManager` есть следующие методы:

- `void close()` — закрывает объект `android.content.res.AssetManager`. Закрыв объект `android.content.res.AssetManager`, вы потеряете к нему доступ и не получите его в следующий раз! Поэтому не нужно пользоваться этим методом!

- `final String[] list(String path)` — получает список файлов, находящихся в каталоге assets.
- `final InputStream open(String fileName)` — открывает файл из каталога assets в режиме `AssetManager.ACCESS_STREAMING`.
- `final InputStream open(String fileName, int accessMode)` — открывает файл из каталога assets с указанием режима открытия (см. ниже описание режимов).

Режимы, которые доступны для открываемых файлов представлены в виде целочисленных (`int`) констант класса `android.content.res.AssetManager`:

- `AssetManager.ACCESS_BUFFER` — предпринять попытку загрузить все содержимое файла в память для быстрых операций чтения.
- `AssetManager.ACCESS_RANDOM` — чтение файлов осуществлять частями, с возможностью перемещения по файлу как вперед, так и назад.
- `AssetManager.ACCESS_STREAMING` — чтение файлов осуществлять последовательно с возможностью перемещения вперед.
- `AssetManager.ACCESS_UNKNOWN` — никакая специфическая информация о том, какой доступ к данным из файла необходим, не указывается.

В примере для этого раздела, добавлена кнопка с надписью «Прочитать файл из assets» и идентификатором `R.id.btnAssetsReadFile`. Для обработки события клика по кнопке создан метод `btnAssetsReadFileClick`, содержимое которого представлено в Листинге 7.1.

Листинг 7.1. Чтение файла test_assets.txt из каталога assets рассматриваемого примера

```
/***
 * Обработчик клика на кнопку "Прочитать файл из assets"
 * @param v - ссылка на виджет, являющийся
 * источником события.
 */
public void btnAssetsReadFileClick(View v)
{
//-- Получение объекта android.content.res.AssetsManager
AssetManager AM = this.getAssets();
try
{
//-- Открытие файла из каталога assets -----
InputStream IS = AM.open("test_assets.txt");
//-- Чтение байтов из файла -----
byte[] b = new byte[1024];
ByteArrayOutputStream BAOS =
                    new ByteArrayOutputStream();
do
{
    int cnt = IS.read(b, 0, b.length);
    if (cnt == -1) break;
    BAOS.write(b, 0, cnt);
}
while (IS.available() > 0);
//-- Преобразование прочитанных байтов в строку -----
byte[] a = BAOS.toByteArray();
String str = new String(a, 0, a.length, "UTF8");
BAOS.close();
// ----- Вывод содержимого файла -----
Log.d(TAG, "Содержимое файла : \n" + str);
Toast.makeText(this, str, Toast.LENGTH_LONG).
        show();
}
catch (Exception e)
```

```
{  
    Log.d(TAG, e.getMessage());  
    Toast.makeText(this, e.getMessage(),  
        Toast.LENGTH_SHORT).show();  
}  
}
```

Результат работы примера из Листинга 7.1 изображен на Рис. 7.4. Содержимое файла test_assets.txt, которое прочитано с помощью метода **btnAssetsReadFileClick**, отображается в объекте **Toast**.

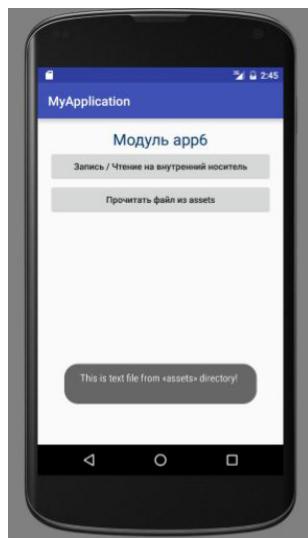


Рис. 7.4. Внешний вид работы примера из Листинга 7.1

8. Взаимодействие с файлами растровых изображений. Класс Bitmap

Достаточно часто в каталог assets приложения размещают файлы растровых изображений (форматы gif, jpeg, png, bmp), чтобы затем использовать эти изображения в приложении — например, в виде пиктограмм или в виде изображений на кнопках или в списках.

Для отображения растровых изображений предназначен класс android.graphics.Bitmap (<http://developer.android.com/intl/ru/reference/android/graphics/Bitmap.html>). Иерархия класса android.graphics.Bitmap достаточно проста:

```
java.lang.Object
 |
 +-- android.graphics.Bitmap
```

Непосредственно сам объект android.graphics.Bitmap только хранит растровое изображение. Но не отображает его! Для отображения растрового изображения предназначен виджет android.widget.ImageView, который описывается ниже в данном уроке.

В классе android.graphics.Bitmap достаточно много методов и в основном эти методы выходят за тематику данного урока, так как имеют отношение к теме графического рисования, которая будет описана в дальнейших

уроках. Однако, несколько методов все же приведем, для ознакомления:

- **static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height)** — создает объект `android.graphics.Bitmap` на основе другого изображения `Bitmap`, при этом указываются координаты и размеры области из которой будут скопированы пиксели для создаваемого изображения.
- **static Bitmap createBitmap(int width, int height, Bitmap.Config config)** — создается объект `android.graphics.Bitmap` указанных размеров и глубиной цвета (параметр `config`).
- **int getHeight()** — возвращает высоту изображения.
- **int getPixel(int x, int y)** — возвращает цвет пикселя по указанным координатам.
- **int getWidth()** — возвращает ширину изображения.
- **boolean hasAlpha()** — возвращает признак поддержки изображением alpha-канала (прозрачность).
- **void setPixel(int x, int y, int color)** — устанавливает пиксель в указанных координатах в заданный цвет.
- **boolean compress(Bitmap.CompressFormat format, int quality, OutputStream stream)** — сохраняет содержимое растрового изображения `android.graphics.Bitmap` в поток вывода с указанием формата сжатия (параметр `format`)

Как видно из перечисленных методов или из справочной информации о классе `android.graphics.Bitmap`

с сайта разработчиков (<http://developer.android.com/intl/ru/reference/android/graphics/Bitmap.html>) у класса **android.graphics.Bitmap** нет методов для создания объекта изображения из файла. Для создания объектов **android.graphics.Bitmap** из файла предназначен класс **android.graphics.BitmapFactory** (<http://developer.android.com/intl/ru/reference/android/graphics/BitmapFactory.html>). В этом классе есть методы для всевозможных вариантов (из массива байтов, из файла, из потока) создания объектов **android.graphics.Bitmap**. Нам интересны следующие два метода:

```
static Bitmap decodeFile(String pathName);
```

и

```
static Bitmap decodeStream(InputStream is);
```

Первый метод принимает путь к файлу, а второй метод принимает ссылку на объект потока, в котором находятся байты растрового изображения.

Пример создания объекта **android.graphics.Bitmap** из файла представлен в Листинге 8.1:

Листинг 8.1. Пример создания объекта
android.graphics.Bitmap из файла

```
Bitmap bmp = BitmapFactory.decodeFile(  
    "Имя_файла_растрового_изображения");
```

Пример создания объекта **android.graphics.Bitmap** из графического файла, находящегося в каталоге assets приложения, приведен в Листинге 8.2:

Листинг 8.2. Создание объекта android.graphics.Bitmap из файла изображения, который находится в каталоге assets

```

try
{
    AssetManager AM = this.getAssets();
    InputStream IS = AM.open("имя_файла_
                           растрового_изображения");
    Bitmap bmp = BitmapFactory.decodeStream(IS);
    IS.close();
    ...
}
catch (IOException ioe)
{
    ...
}

```

И в качестве шутливого, но интересного примера, можно создать объект **android.widget.Bitmap** с помощью прорисовки каждого пикселя. Пример такого создания объекта **android.graphics.Bitmap** представлен в Листинге 8.3.

Листинг 8.3. Создание объекта android.graphics.Bitmap с помощью прорисовки каждого пикселя

```

//-- Создание изображения android.graphics.Bitmap --
Bitmap bmp = Bitmap.createBitmap(100, 100,
                                Bitmap.Config.ARGB_8888);
//-- Заливаем красным фоном -----
for (int y = 0; y < 100; y++)
{
    for (int x = 0; x < 100; x++)
    {
        bmp.setPixel(x, y, Color.rgb(0xFF, 0x00, 0x00));
    }
}

```

```
//-- Рисуем посередине желтый прямоугольник -----
for (int y = 0; y < 50; y++)
{
    for (int x = 0; x < 50; x++)
    {
        bmp.setPixel(x + 25, y + 25,
                     Color.rgb(0xFF, 0xFF, 0x00));
    }
}
//-- Назначим созданный Bitmap виджету ImageView --
ImageView iv1 = (ImageView) this.
    findViewById(R.id.iv1);
iv1.setImageBitmap(bmp);
```

Внешний вид работы примера из Листинга 8.3 изображен на Рис. 8.1.

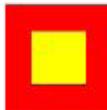


Рис. 8.1. Внешний вид работы
примера из Листинга 8.3

Листинги данного раздела являются короткими и поэтому не находятся в файлах исходного кода, прилагаемых к данному уроку. Указанный в этих листингах код будет повторно использован в следующих разделах.

9. Отображение изображений в приложении. Виджет android.widget.ImageView

Виджет `android.widget.ImageView` (<http://developer.android.com/intl/ru/reference/android/widget/ImageView.html>) предназначен для отображения растровых изображений `android.graphics.Bitmap`. Иерархия класса `android.widget.ImageView` следующая:

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

Как видно из иерархии классов, виджет `android.widget.ImageView` является таким же виджетом что и кнопка или текстовое поле. А значит многие возможности по взаимодействию с этим виджетом нам уже знакомы.

Два самых главных метода этого класса, предназначенные для отображения растровых изображений, следующие:

```
void setImageBitmap(Bitmap bm);
```

и

```
void setImageURI(Uri uri);
```

Первый метод принимает ссылку на объект `android.graphics.Bitmap` (как создавать эти объекты рассказывалось

в предыдущем разделе). Второй метод принимает ссылку на путь к файлу в виде объекта `android.net.Uri` (<http://developer.android.com/intl/ru/reference/android/net/Uri.html>).

Рассмотрим варианты назначения объекта `android.graphics.Bitmap` для отображения его в виджете `android.widget.ImageView`. Для всех примеров используется виджеты `android.widget.ImageView`, xml верстка которых представлена в Листинге 9.1.

Листинг 9.1. Xml верстка для виджетов `android.widget.ImageView`, которые используются в рассматриваемых в разделе примерах

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:id="@+id/iv1"
    />
```

В Листинге 9.2 представлен вариант отображения в виджете `android.widget.ImageView` объекта `android.graphics.Bitmap`, который создан из файла на внешнем носителе. Удобный случай повторить пройденные темы текущего урока.

Листинг 9.2. Отображение изображения из общедоступного каталога внешнего носителя Pictures в виджете android.widget.ImageView

```
/*
 * Пример №1.
 * Отображение изображения из общедоступного
 * каталога внешнего носителя Pictures в виджете
 * android.widget.ImageView
 */
```

```
ImageView iv1 = (ImageView) this.  
        findViewById(R.id.iv1);  
if (this.getExternalStorageReadable())  
{  
    File picDir = Environment.  
        getExternalStoragePublicDirectory(  
            Environment.DIRECTORY_PICTURES);  
    File imgFile = new File(picDir, "auto_09.png");  
    if(imgFile.exists())  
    {  
        Bitmap bmp1 = BitmapFactory.  
            decodeFile(imgFile.  
                getAbsolutePath());  
        iv1.setImageBitmap(bmp1);  
    }  
    else  
    {  
        Log.d(TAG, "Файл не найден : " +  
            imgFile.getAbsolutePath());  
    }  
}  
else  
{  
    Toast.makeText(this, "Внешний носитель  
        не готов", Toast.LENGTH_SHORT).  
    show();  
}
```

Внимание! Для примера из Листинга 9.2, предварительно с помощью приложения Android Device Monitor был скопирован файл auto_09.png в общедоступный каталог внешнего носителя Pictures. Не забудьте сделать тоже самое при тестировании этого примера в своем эмуляторе!

Внешний вид работы примера из Листинга 9.2 изображен на Рис. 9.1 слева.

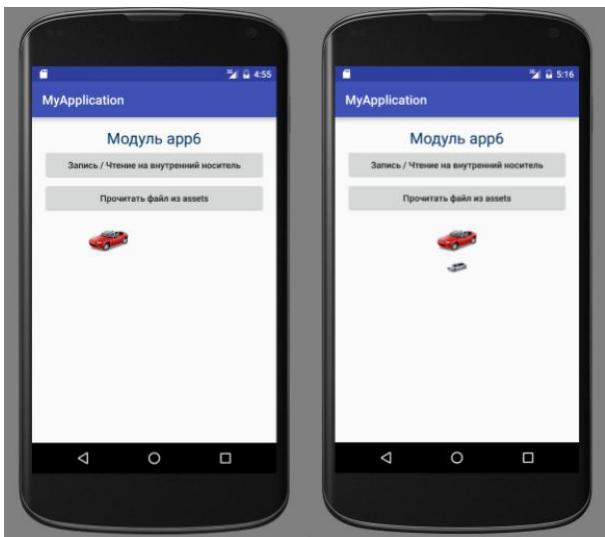


Рис. 9.1. Внешний вид работы примеров из Листингов 9.2 и 9.3

Следующим примером (Листинг 9.3) отображения изображения в виджете `android.widget.ImageView` будет отображение файла растрового изображения также находящегося на внешнем носителе в общедоступном каталоге Pictures, с помощью метода `setImageURI`:

Листинг 9.3. Отображение изображения из общедоступного каталога Pictures внешнего носителя в виджете `android.widget.ImageView` с помощью метода `setImageURI`

```
/*
 * Пример №2.
 * Отображение изображения из общедоступного
 * каталога Pictures внешнего носителя в виджете
 * android.widget.ImageView
 * с помощью метода setImageURI
 * -----
 */
```

```
ImageView iv2 = (ImageView) this.  
        findViewById(R.id.iv2);  
  
if (this.getExternalStorageReadable())  
{  
    File picDir = Environment.  
            getExternalStoragePublicDirectory(  
                Environment.DIRECTORY_PICTURES);  
    File imgFile = new File(picDir, "auto_08.png");  
  
    if(imgFile.exists())  
    {  
        iv2.setImageURI(Uri.fromFile(imgFile));  
    }  
    else  
    {  
        Log.d(TAG, "Файл не найден : " +  
            imgFile.getAbsolutePath());  
    }  
}  
else  
{  
    Toast.makeText(this, "Внешний носитель не  
        готов", Toast.LENGTH_SHORT).show();  
}
```

Внимание! Для примера из Листинга 9.3, предварительно с помощью приложения Android Device Monitor был скопирован файл auto_08.png в общедоступный каталог внешнего носителя Pictures. Не забудьте сделать тоже самое при тестировании этого примера в своем эмуляторе!

Внешний вид работы примера из Листинга 9.3 изображен на Рис. 9.1 справа. Обратите внимание, что размеры обоих изображений для примеров из Листингов 9.2 и 9.3

одинаковы (128×128 пикселей). Однако, изображение, отображаемое с помощью метода `setImageURI` отображается в меньших размерах. То есть для этого случая не происходит перерасчет размеров изображения с учетом плотности количества точек на дюйм в устройстве и изображение отображается в виде 1 пиксель = 1dpi. Возможно, этот недочет разработчики компании Google исправят в следующих версиях API.

Следующим примером будет пример на отображение изображения из каталога assets приложения. Предварительно в каталог assets был скопирован файл auto_05.png. Код примера представлен в Листинге 9.4:

Листинг 9.4. Отображение изображения из каталога assets в виджете `android.widget.ImageView`

```
/*
 * Пример №3.
 * Отображение изображения из каталога
 * assets приложения в виджете
 * android.widget.ImageView
 * путем создания android.graphics.Bitmap из потока
 * с помощью BitmapFactory.decodeStream
 * -----
 */
ImageView iv3 = (ImageView) this.findViewById(R.id.iv3);
AssetManager AM = this.getAssets();
try
{
    InputStream IS = AM.open("auto_05.png");
    Bitmap bmp = BitmapFactory.decodeStream(IS);
    iv3.setImageBitmap(bmp);
}
catch (IOException ioe)
{
```

```
    IS.close();
    Log.d(TAG, "IOException : " +
        ioe.getMessage());
}
```

Внешний вид примера из Листинга 9.4 изображен на Рис. 9.2 слева (третья картинка сверху).

И последний пример отображения изображения в виджете `android.widget.ImageView` затронет тему ресурсов приложения. Поскольку тема ресурсов приложения будет подробно рассматриваться в следующих уроках, в текущем уроке будет только пример извлечения изображения из ресурсов и отображение этого изображения в виджете `android.widget.ImageView`. Для этого примера предварительно с помощью файлового менеджера в каталог имени_модуля/res/drawable был скопирован файл с изображением `auto_04.png`. Программный пример приведен в Листинге 9.5:

Листинг 9.5. Отображение в виджете `android.widget.ImageView` изображения из ресурсов приложения

```
/*
 * Пример №4 .
 * Отображение изображения из ресурсов приложения
 * в виджете android.widget.ImageView
 * -----
 */
ImageView iv4 = (ImageView) this.findViewById(R.id.iv4);
iv4.setBackgroundResource(R.drawable.auto_04);
```

Как видно из Листинга 9.5, файл, скопированный в каталог для графических ресурсов `/res/drawable`

доступен в приложении через класс **R** (т. е. **R.drawable.имя_файла_без_расширения**), как, например, идентификаторы виджетов (т. е. **R.id.идентификатор_виджета**).

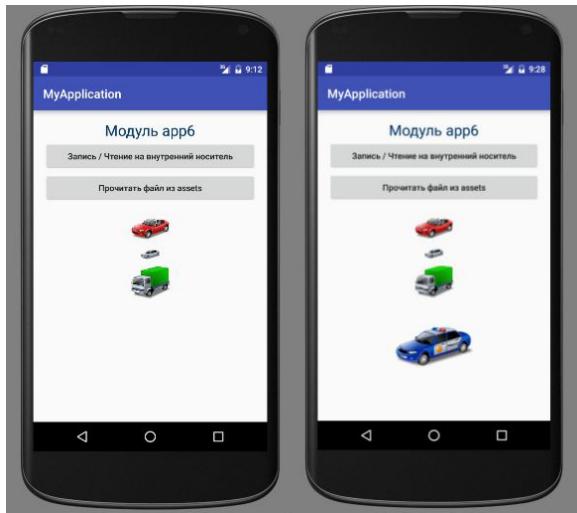


Рис. 9.2. Внешний вид работы примеров из Листинга 9.4 (слева) и Листинга 9.5 (справа)

Внешний вид работы примера из Листинга 9.5 представлен на Рис. 9.2 справа (четвертая картинка сверху). Обратите внимание, что размер отображаемого изображения больше остальных изображений, хотя сам графический файл имеет такое же разрешение, что и другие графические файлы из примеров этого раздела (128×128 пикселей). Не будем делать выводы о причинах, просто примем этот факт к сведению, чтобы как-нибудь на досуге поразмышлять над ним.

Исходные коды примеров этого раздела можно найти в модуле «аррб» среди файлов исходных кодов. Прилагаемых к этому уроку. Все изображения, используемые в примерах раздела находятся в папке `аррб/res/drawable`.

10. Виджет android.widget.GridView

Виджет `android.widget.GridView` (<http://developer.android.com/intl/ru/reference/android/widget/Gridview.html>) это виджет, похожий на список `android.widget.ListView`, только `GridView` отображает элементы списка в прокручиваемой сетке. Данные для элементов списка `android.widget.GridView` берутся из Адаптера Данных `android.widget.ListAdapter` (<http://developer.android.com/reference/android/widget/ListAdapter.html>). Таким Адаптером данных является хорошо знакомый вам `android.widget.ArrayAdapter<T>` (<http://developer.android.com/reference/android/widget/ArrayAdapter.html>).

Иерархия классов для `android.widget.GridView` имеет следующий вид:

```
java.lang.Object
|
+--- android.view.View
|
+--- android.view.ViewGroup
|
+--- android.widget.
        AdapterView<android.
        widget.ListAdapter>
|
+--- android.widget.AbsListView
|
+--- android.widget.GridView
```

Как видно из иерархии классов, класс `android.widget.GridView` имеет общего со списком `android.widget.ListView` родителя `android.widget.AbsListView`. Следовательно, многое о классе `android.widget.GridView` мы уже знаем!

Xml верстка виджета `android.widget.GridView` имеет следующий вид (Листинг 10.1):

Листинг 10.1. Xml верстка виджета android.widget.GridView

```
<GridView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center" android:id="@+id/gvOne"
    />
```

В Листинге 10.1 жирным шрифтом выделены атрибуты, которые относятся именно к классу `android.widget.GridView`. Давайте рассмотрим эти атрибуты подробнее.

- **`android:columnWidth`** — задает фиксированную ширину для каждого столбца (не забывайте об `dp`) для виджета `android.widget.GridView`. Программно задать ширину столбцов можно с помощью метода `setColumnWidth(int)`.
- **`android:horizontalSpacing`** — задает величину размера горизонтального расстояния между столбцами. Программно задать величину размера горизонтального расстояния можно с помощью метода `setHorizontalSpacing(int)`.

- **android:numColumns** — задает количество столбцов в виджете `android.widget.GridView`. Если атрибуту назначить значение `auto_fit` (т. е. `-1`), то количество столбцов будет таким, сколько поместится по ширине экрана устройства. Программно задать количество столбцов можно с помощью метода `setNumColumns(int)`.
- **android:stretchMode** — задает, как будут столбцы растягиваться для заполнения всего доступного пустого пространства (если таковое имеется). Значения для атрибута следующие: `none` (растяжения нет), `spacingWith` (растягивается пространство между столбцами), `spacingWithUniform` (растягивается пространство между столбцами, при этом пространство имеет одинаковый размер), `columnWidth` (растягиваются столбцы). Программно задать режим растяжения столбцов можно с помощью метода `setStretchMode(int)`. Для метода в классе `android.widget.GridView` объявлены целочисленные константы: `NO_STRETCH` (растяжения нет), `STRETCH_SPACING` (растягивается пространство между столбцами), `STRETCH_SPACING_UNIFORM` (растягивается пространство между столбцами, при этом пространство имеет одинаковый размер), `STRETCH_COLUMN_WIDTH` (растягиваются столбцы).
- **android:verticalSpacing** — задает величину размера вертикального расстояния между столбцами. Программно задать величину размера вертикального расстояния можно с помощью метода `setVerticalSpacing(int)`.

Назначим виджету `android.widget.GridView` из Листинга 10.1 Адаптер данных, как это представлено в Листинге 10.2

и посмотрим, что получится. Программный код примера из Листинга 10.2 можно выполнить в методе **onCreate** класса Активности.

Листинг 10.2. Назначение адаптера данных android.widget.ArrayAdapter<T> списку android.widget.GridView

```
/*
 * Назначение виджету android.widget.GridView
 * Адаптера данных
 */
//-- Получаем ссылку на виджет -----
GridView gvOne = (GridView) this.
    findViewById(R.id.gvOne);
//-- Формируем коллекцию со строками для элементов
//-- списка -----
String[] fruits =
{
    "Lemon", "Kiwi", "Orange", "Banana",
    "Apple", "Peach",
};
ArrayList<String> listFruits = new ArrayList<>();
for (int i = 0; i < 10; i++)
{
    for (int j = 0; j < fruits.length; j++)
    {
        listFruits.add(fruits[j] + " " + i);
    }
}
//-- Создаем Адаптер данных ArrayAdapter<String> --
ArrayAdapter<String> adapter = new ArrayAdapter<>(
    this,
    android.R.layout.simple_list_item_1,
    listFruits);
//-- Назначаем Адаптер данных списку -----
gvOne.setAdapter(adapter);
```

В Листинге 10.2 жирным шрифтом выделен идентификатор ресурса xml макета, который используется для представления элементов списка. Как видите, был взят стандартный идентификатор `android.R.layout.simple_list_item_1`, который мы брали для первых примеров по виджету `android.widget.ListView` в этом уроке. То есть подтверждается очень сильная схожесть виджетов `ListView` и `GridView`.

Внешний вид работы примера из Листинга 10.2 изображен на Рис. 10.1.

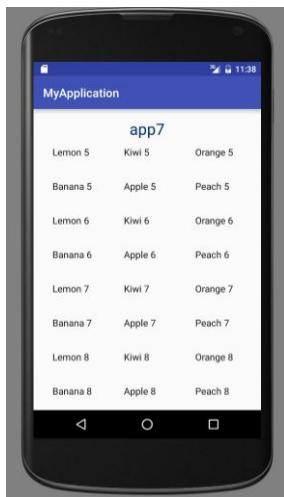


Рис. 10.1. Внешний вид работы примера из Листинга 10.2

Для того, чтобы определять по какому элементу списка осуществил нажатие пользователь, необходимо создать объект, который должен реализовать уже известный вам интерфейс `AdapterView.OnItemClickListener` (<http://developer.android.com/intl/ru/reference/android/widget/AdapterView.OnItemClickListener.html>).

Пример назначения списку **android.widget.GridView** обработчика клика по элементу **AdapterView.OnItemClickListener** **ClickListener** приведен в Листинге 10.3:

Листинг 10.3. Назначение обработчика события нажатия по элементу списка **android.widget.GridView**

```
//-- Назначаем виджету android.widget.GridView
//-- обработчик клика по элементу
    gvOne.setOnItemClickListener(new
        AdapterView.OnItemClickListener()
    {
        @Override
        public void onItemClick(AdapterView<?> parent,
                            View view, int position, long id)
        {
            String item = parent.getAdapter().
                getItem(position).toString();
            Toast.makeText(MainActivity.this,
                "Выбран : " +
                item, Toast.LENGTH_SHORT).show();
        }
    });
});
```

Результат работы примера из Листинга 10.3 изображен на Рис. 10.2.

В общем, виджет **android.widget.GridView** можно считать уже изученным, так как он очень похож на виджет **android.widget.ListView**. В завершении знакомства с виджетом **android.widget.GridView** рассмотрим пример, в котором совместим полученные в этом уроке знания. Пример будет на создание оригинального макета виджета для элементов списка **android.widget.GridView**, в котором будут отображаться изображения, взятые из файлов,

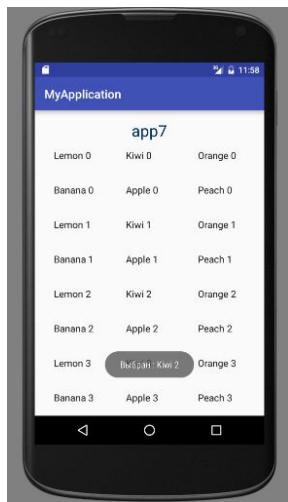


Рис. 10.2. Результат работы примера из Листинга 10.3

и под каждым изображением будет выводиться название файла этого изображения.

Для начала, создадим класс, объекты которого будут представлять элементы списка. Класс назовем **GridViewItem** и будет он состоять из двух полей: **bmp** — ссылка на изображение **android.graphics.Bitmap** и **title** — название файла (подпись под изображением). Программный код класса **GridViewItem** представлен в Листинге 10.4.

Листинг 10.4. Класс **GridViewItem**, объекты которого будут представлять элементы списка **android.widget.GridView**

```
/**
 * Class GridViewItem - объект, представляющий собой
 * элемент данных для списка android.widget.GridView
 */
class GridViewItem
```

```
/**  
 * Bitmap изображение элемента  
 */  
public Bitmap bmp;  
  
/**  
 * Подпись элемента  
 */  
public String title;  
  
//-- Class members -----  
public GridViewItem(Bitmap bmp, String title)  
{  
    this.bmp = bmp;  
    this.title = title;  
}  
  
@Override  
public String toString()  
{  
    return this.title;  
}  
}
```

Для рассматриваемого примера файлы изображений будем брать из каталога assets приложения. Поэтому предварительно создадим этот каталог в модуле и скопируем в него файлы с изображениями (Рис. 10.3).

Следующий шаг — создадим файл ресурсов с xml макетом раскладки набора виджетов, который будет представлять элементы списка **android.widget.GridView**. Файл с макетом раскладки называется /res/layout/my_gridview_item.xml и его содержимое приведено в Листинге 10.5:

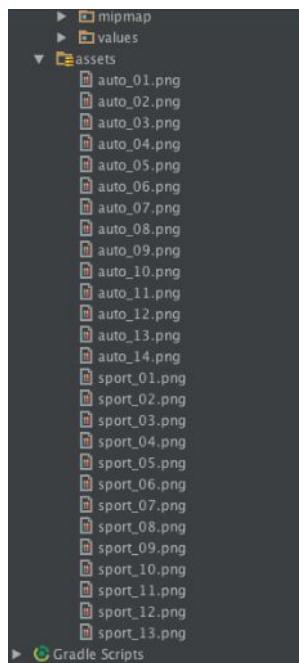


Рис. 10.3. Файлы изображений, помещенные для рассматриваемого примера в каталог assets приложения

Листинг 10.5. Макет набора виджетов из файла ресурсов /res/layout/my_gridview_item.xml для элемента списка android.widget.GridView

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

    <ImageView
        android:layout_width="80dp"
```

```

        android:layout_height="80dp"
        android:layout_gravity="center"
        android:id="@+id/ivImg"
    />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:textColor="#003366"
    android:id="@+id/tvTitle"
    />

</LinearLayout>

```

И последним шагом будет извлечение изображений из каталога assets и помещение их в объекты **GridViewItem**, которые, в свою очередь, будут помещены в коллекцию для Адаптера данных **android.widget.ArrayAdapter<T>**, который будет назначен списку **android.widget.GridView**. Программный код этого шага выполняется в методе **onCreate** класса активности и приведен в Листинге 10.6:

Листинг 10.6. Формирование Адаптера данных **android.widget.ArrayAdapter<T>** и назначение его виджету **android.widget.GridView**

```

/*
 * Наполнение списка android.widget.GridView
 * данными (изображения с подписью).
 * -----
 */
//-- Список для объектов, которые будут отображаться
//-- в списке -----
ArrayList<GridViewItem> items = new ArrayList<>();

```

```
//-- Получаем ссылку на виджет android.widget.GridView
GridView gvOne = (GridView) this.
    findViewById(R.id.gvOne);

//-- Получение AssetManager -----
AssetManager AM = this.getAssets();
try
{
    String[] F = AM.list("");
    for (String f : F)
    {

//-- Загружаем изображение -----
try
{
    InputStream IS = AM.open(f);
    Bitmap bmp = BitmapFactory.decodeStream(IS);
    IS.close();

//-- Заполняем коллекцию для адаптера данных -----
    items.add(new GridViewItem(bmp, f));
}
catch (Exception e)
{
}

//-- Это не графический файл или это каталог -----
    Log.d(TAG, " Exception : " + e.getMessage());
}
}
}

catch (IOException ioe)
{
    Log.d(TAG, "IOException : " + ioe.getMessage());
    Toast.makeText(this,
        "IOException : \n" + ioe.getMessage(),
        Toast.LENGTH_SHORT).show();
}
```

```
//-- Создание Адаптера данных -----
ArrayAdapter<GridViewItem> adapter =
        new ArrayAdapter<GridViewItem>(this,
            R.layout.my_gridview_item,
            R.id.tvTitle, items)
{
    @Override
    public View getView(int position,
        View convertView, ViewGroup parent)
    {
//-- Виджет для элемента списка подготовит метод
//-- родительского класса -----
        View view = super.getView(
            position, convertView, parent);

//-- Получение ссылки на отображаемый в виджете
//-- объект GridViewItem -----
        GridViewItem GVI = this.
            getItem(position);

//-- Размести данные элемента в виджете -----
        ImageView ivImg = (ImageView) view.
            findViewById(R.id.ivImg);
        ivImg.setImageBitmap(GVI.bmp);

        TextView tvTitle = (TextView) view.
            findViewById(R.id.tvTitle);
        tvTitle.setText(GVI.title);

        return view;
    }
};

//-- Назначаем Адаптер данных виджету
//-- android.widget.GridView -----
gvOne.setAdapter(adapter);
```

```
//--- Назначаем виджету android.widget.GridView  
//--- обработчик клика по элементу -----  
gvOne.setOnItemClickListener(new AdapterView.  
    OnItemClickListener()  
{  
    @Override  
    public void onItemClick(AdapterView<?> parent,  
                            View view, int position,  
                            long id)  
    {  
        GridViewItem item = (GridViewItem) parent.  
            getAdapter().getItem(position);  
        Toast.makeText(MainActivity.this,  
                    "Выбран : " + item.title,  
                    Toast.LENGTH_LONG).show();  
    }  
});
```

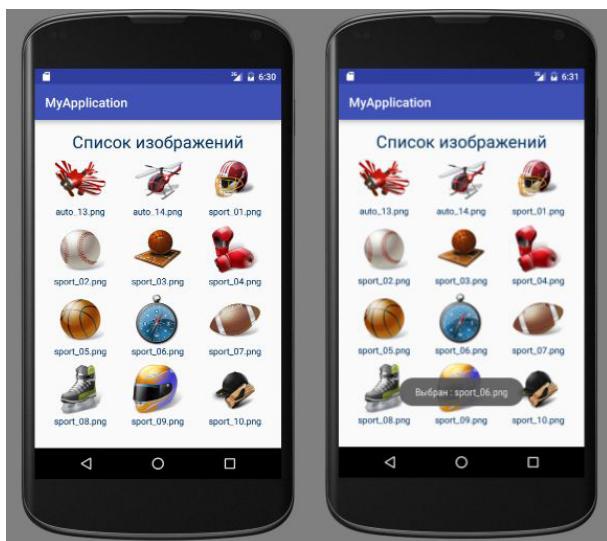


Рис. 10.4. Внешний вид работы примера из Листингов 10.4, 10.5, 10.6

Обратите внимание, в Листинге 10.6 каждый элемент списка является сложным элементом, т.е. элементом, состоящим из более чем одного значения. Для таких элементов нужно или использовать Адаптер данных **android.widget.SimpleAdapter ()** или переопределять метод **getView** Адаптера данных. Оба эти варианта уже описывались в наших уроках этого курса. В Листинге 10.6 используется способ в котором осуществляется переопределение метода **getView**.

Внешний вид работы примера из Листинга 10.6 изображен на Рис. 10.4.

Верстка макета Активности текущего примера приведена в Листинге 10.7.

Листинг 10.7. Верстка макета Активности для примера этого раздела

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/
        android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="itstep.com.myapp7.MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textSize="12pt"
        android:textColor="#003366"
        android:text="Список изображений" />
```

```
<GridView  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:columnWidth="90dp"  
    android:numColumns="auto_fit"  
    android:verticalSpacing="10dp"  
    android:horizontalSpacing="10dp"  
    android:stretchMode="columnWidth"  
    android:gravity="center"  
    android:id="@+id/gvOne" />  
  
</LinearLayout>
```

Исходные коды примера находятся в модуле «app7» среди файлов исходных кодов, прилагаемых к этому уроку.

11. Домашнее задание

1. Необходимо разработать приложение, работающее со списком сотрудников. Внешний вид приложения изображен на Рис. 11.1.

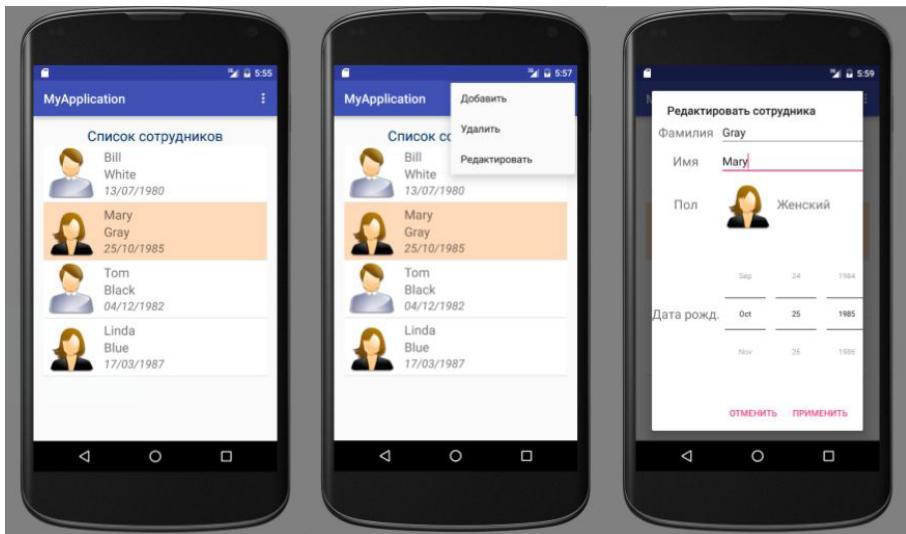


Рис. 11.1. Внешний вид приложения из домашнего задания №1

Список сотрудников отображается в виджете `android.widget.ListView` с custom-подсветкой выбранного пользователем элемента списка. Изображения «Мужчина» и «Женщина» должны находиться в каталоге assets приложения. Рекомендуется сделать в классе Активности два поля типа `android.graphics.Bitmap` (одно поле — ссылка на `Bitmap` «Мужчина», второе — на «Женщина»), в которые выполнить загрузку изображений из каталога assets

(в методе **onCreate**). Изображения загружаются всего один раз, а далее они назначаются виджетам **android.widget.ImageView** с помощью метода **setImageBitmap**.

Далее, в приложении пользователь имеет возможность добавлять/удалять/редактировать сотрудников. Для выбора соответствующего действия предназначено меню приложения (см. Рис. 11.1 второе изображение). Для добавления или редактирования информации о сотруднике нужно использовать Диалоговое окно (см. Рис. 11.1 изображение справа). Диалоговое окно можно создать один раз в методе **onCreate** и затем только отображать с помощью вызова метода **show()**.

Информация о сотрудниках содержится в объектах класса **Human**, который представлен в Листинге 11.1:

Листинг 11.1. Класс Human, объекты которого содержат данные о сотрудниках

```
/**  
 * Class Human  
 * -----  
 */  
class Human  
{  
    //--- Class members -----  
    /**  
     * Фамилия  
     */  
    public String firstName;  
  
    /**  
     * Имя  
     */  
    public String lastName;
```

```
/**  
 * Пол. true - мужской  
 */  
public boolean gender;  
  
/**  
 * День рождения  
 */  
public Calendar birthDay;  
  
//--- Class methods -----  
public Human(String firstName, String lastName,  
             boolean gender, Calendar birthDay)  
{  
    this.firstName      = firstName;  
    this.lastName       = lastName;  
    this.gender         = gender;  
    this.birthDay       = birthDay;  
}  
  
/**  
 * Возвращает дату рождения в виде строки dd/mm/yyyy  
 */  
public String getBirthDayString()  
{  
    String      str      = "";  
  
    int day = this.birthDay.get(Calendar.DAY_OF_MONTH);  
    str += ((day < 10)?"0":"") + day + "/";  
  
    int mon = this.birthDay.get(Calendar.MONTH) + 1;  
  
    str += ((mon < 10)?"0":"") + mon + "/";  
  
    str += this.birthDay.get(Calendar.YEAR);  
    return str;  
}
```

```
public static Calendar makeCalendar(int day,  
                                    int month, int year)  
{  
    Calendar C = Calendar.getInstance();  
    C.setTimeInMillis(0);  
    C.set(Calendar.YEAR, year);  
    C.set(Calendar.MONTH, month - 1);  
    C.set(Calendar.DAY_OF_MONTH, day);  
    return C;  
}  
}
```

Далее. Для списка `android.widget.ListView` использовать Адаптер данных `android.widget.ArrayAdapter<Human>`. Список сотрудников автоматически сохраняется приложением (например с помощью сериализации) в файл на внутреннем носителе. При старте приложения содержимое этого файла считывается для отображения списка сотрудников из сохраненного файла.

Не забудьте о поворотах устройства! Для этой цели реализуйте всю необходимую функциональность.

2. Необходимо создать приложение «Файловый менеджер». Внешний вид приложения изображен на Рис. 11.2.

Приложение «Файловый менеджер» работает только с файлами и каталогами, находящимися на внешнем носителе. Файлы и каталоги отображаются в списке `android.widget.GridView`. Попробуйте реализовать custom-подсветку выбранного пользователем элемента.

При длительном нажатии на каталог, приложение «Файловый менеджер» «заходит» в этот каталог и отображает содержимое этого каталога. Для обработки события

длительного нажатия необходимо использовать метод класса `android.widget.GridView`:

```
void setOnItemLongClickListener(AdapterView.  
    OnItemLongClickListener listener);
```

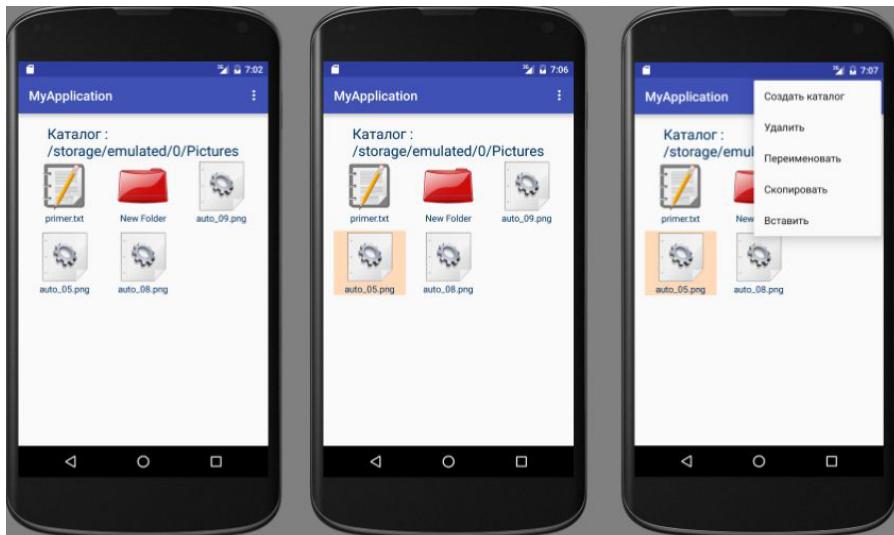


Рис. 11.2. Внешний вид приложения «Файловый менеджер» из домашнего задания №2

В приложении «Файловый менеджер» используется три вида изображений: для каталогов, для текстовых файлов и для всех остальных типов файлов.

Пользователь может совершать с помощью приложения «Файловый менеджер» следующие действия: создание каталога, удаление файла или каталога (каталог удаляется рекурсивно), переименование файла или каталога, копирование и вставка файла или каталога (каталог копируется рекурсивно). Для выбора соответствующего действия

необходимо использовать меню или контекстное меню (на ваш выбор).

Перед удалением файла или каталога необходимо с помощью Диалогового окна уточнить у пользователя подтверждение операции удаления.

Приложение «Файловый менеджер» не удаляет и не переименовывает общедоступные каталоги.



Урок №3

Layout, Views

© Бояршинов Юрий

© Компьютерная Академия «Шаг»

www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объеме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объем и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.