# Списки. Адаптеры. Активности.

***RelativeLayout***

Располагает дочерние элементы относительно позиции других дочерних элементов разметки или относительно области самой разметки RelativeLayout.

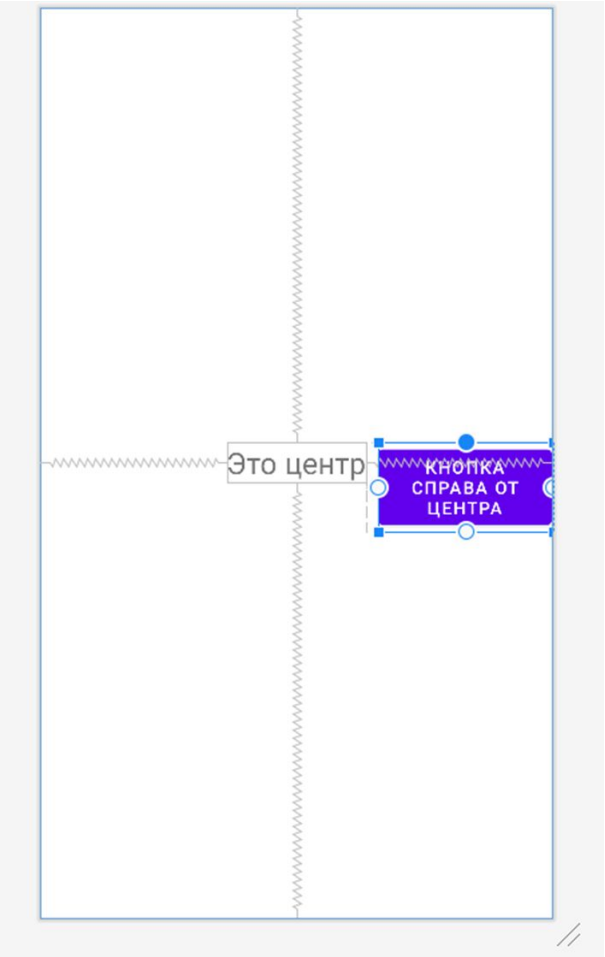
Атрибуты позиционирования элементов в файле xml:

- **layout_above**: располагает элемент над элементом с указанным Id
- **layout_below**: располагает элемент под элементом с указанным Id
- **layout_toLeftOf**: располагается слева от элемента с указанным Id
- **layout_toRightOf**: располагается справа от элемента с указанным Id
- **layout_toStartOf**: располагает начало текущего элемента, где начинается элемент с указанным Id
- **layout_toEndOf**: располагает начало текущего элемента, где завершается элемент с указанным Id

- **layout_alignBottom**: выравнивает элемент по нижней границе другого элемента с указанным Id
- **layout_alignLeft**: выравнивает элемент по левой границе другого элемента с указанным Id
- **layout_alignRight**: выравнивает элемент по правой границе другого элемента с указанным Id
- **layout_alignStart**: выравнивает элемент по линии, у которой начинается другой элемент с указанным Id
- **layout_alignEnd**: выравнивает элемент по линии, у которой завершается другой элемент с указанным Id
- **layout_alignTop**: выравнивает элемент по верхней границе другого элемента с указанным Id
- **layout_alignBaseline**: выравнивает базовую линию элемента по базовой линии другого элемента с указанным Id

**layout_alignParentBottom**: если атрибут имеет значение true, то элемент прижимается к нижней границе контейнера
**layout_alignParentRight**: если атрибут имеет значение true, то элемент прижимается к правому краю контейнера
**layout_alignParentLeft**: если атрибут имеет значение true, то элемент прижимается к левому краю контейнера
**layout_alignParentStart**: если атрибут имеет значение true, то элемент прижимается к начальному краю контейнера (при левосторонней ориентации текста - левый край)
**layout_alignParentEnd**: если атрибут имеет значение true, то элемент прижимается к конечному краю контейнера (при левосторонней ориентации текста - правый край)
**layout_alignParentTop**: если атрибут имеет значение true, то элемент прижимается к верхней границе контейнера
**layout_centerInParent**: если атрибут имеет значение true, то элемент располагается по центру родительского контейнера
**layout_centerHorizontal**: при значении true выравнивает элемент по центру по горизонтали
**layout_centerVertical**: при значении true выравнивает элемент по центру по вертикали

```
android:layout_width="wrap_content"

android:layout_height="wrap_content
"


android:layout_centerInParent="true
"
    android:text="Это центр"
    android:textSize="24sp" />

<Button

android:layout_width="wrap_content"

android:layout_height="wrap_content
"


android:layout_alignTop="@id/txtVie
w"

android:layout_marginLeft="10dp"

android:layout_toRightOf="@id/txtVi
ew"
```



```
<Button

android:layout_width="wrap_conten
t"

android:layout_height="wrap_conte
nt"

android:layout_alignLeft="@id/txt
View"

android:layout_marginLeft="10dp"

android:layout_alignParentBottom=
"true"
    android:text="Кнопка Внизу"
/>
```

Создание элемента RelativeLayout в коде:

```
RelativeLayout relativeLayout = new RelativeLayout(this);

 TextView txtView = new TextView(this);
 txtView.setId(TextView.generateViewId());
 txtView.setText("Это центр!");
 RelativeLayout.LayoutParams txtViewParams = new
RelativeLayout.LayoutParams(
                RelativeLayout.LayoutParams.WRAP_CONTENT,
                RelativeLayout.LayoutParams.WRAP_CONTENT);
// выравнивание по центру родительского контейнера
 txtViewParams.addRule(RelativeLayout.CENTER_IN_PARENT);
// добавление в RelativeLayout
  relativeLayout.addView(txtView, txtViewParams);
  setContentView(relativeLayout);
```

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

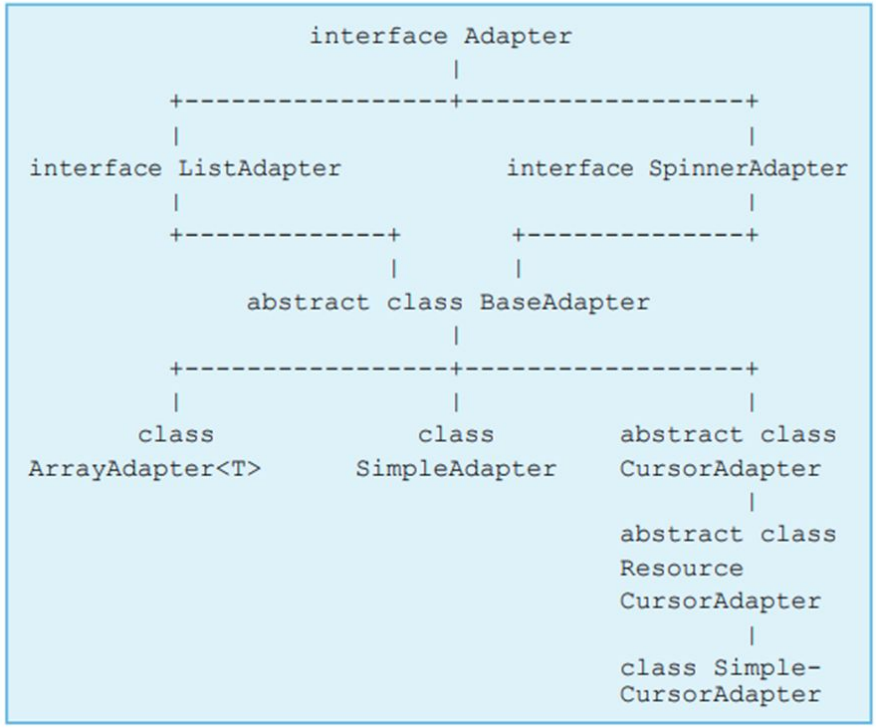
Адаптер данных служит мостом между элементом управления и набором данных, предназначенным для отображения в этом виджете.

Адаптер предоставляет доступ к этому набору данных и также отвечает за создание виджетов для каждого элемента данных из этого набора.

Где-бы не происходили изменения данных — в виджетах в результате действий пользователя или в наборе данных в результате работы программ — синхронизировать виджеты и набор данных не нужно! Адаптер данных это сделает автоматически!

В Android существует множество классов Адаптеров данных, каждый из которых ориентирован на соответствующий виджет списка.

Родительским классом для всех классов Адаптеров является класс **android.widget.BaseAdapter**
(http://developer.android.com/intl/ru/reference/ android/widget/BaseAdapter.html)



## *Класс ArrayAdapter<T>*

http://developer.android.com/ intl/ru/reference/android/widget/ArrayAdapter.html

Это готовый Адаптер данных, который предназначен для использования в списках как **Spinner** и **ListView**.

Принимает в качестве набора данных список (коллекцию, реализующую интерфейс List — например коллекцию ArrayList) или массив объектов.

По умолчанию, ArrayAdapter перебирает каждый элемент набора данных и вставляет строковое значение, которое формируется с помощью вызова метода **toString()** для отображаемого объекта данных, в указанный виджет TextView (виджет TextView в виде идентификатора ресурсов передается в конструктор класса ArrayAdapter).

Если нужно использовать для отображения элементов списка другой виджет или виджеты вместо предлагаемого по умолчанию виджета TextView, то необходимо создать класс, производный от ArrayAdapter и переопределить в нем метод **getView(int, View, ViewGroup)**, в котором или с помощью оператора new или с помощью объекта LayoutInflater реализовать создание нужного виджета, отображающего элемент данных из набора данных Адаптера.

Класс ArrayAdapter содержит методы по работе с коллекцией данных — **add, insert, remove, sort, clear** и метод **setDropDownViewResource** для задания макета из ресурсов для отображения пунктов выпадающего списка

*Конструкторы класса ArrayAdapter*:

- **ArrayAdapter(Context context, int resource, T[] objects)** — Принимает идентификатор ресурса содержащего макет, представляющий внешний вид каждого элемента списка и массив объектов который является набором данных для Адаптера. Т.к. в качестве набора данных выступает массив, то такой набор данных является *не модифицируемым* с точки зрения добавления и удаления элементов, но менять содержимое в массиве объектов можно.

- **ArrayAdapter(Context context, int resource, List objects)** — данный конструктор предназначен для работы с модифицируемым с точки зрения добавления и удаления набором данных. Во всем остальном, этот конструктор совпадает с предыдущим конструктором.

*Методы класса ArrayAdapter*:

■ **public void add(T object)** — добавляет новый объект object в набор данных Адаптера. Объект добавляется в конец списка.
■ **public void clear()** — удаляет все объекты из набора данных Адаптера.
■ **public int getCount()** — возвращает количество объектов в наборе данных Адаптера (количество элементов списка)

■ **public T getItem (int position)** — возвращает ссылку на объект из набора данных Адаптера по индексу position.
■ **public int getPosition (T item)** — возвращает индекс объекта item из набора данных

■ **public void insert (T object, int index)** — вставляет новый объект object в набор данных Адаптера в указанную позицию. Позиция, в которую осуществляется вставка, указывается параметром index.

■ **public void notifyDataSetChanged ()** — метод предназначен для оповещения виджета-списка о том, что в назначенном им Адаптере данных произошли изменения и виджету-списку необходимо обновить свое представление (выполнить перерисовку). Обычно, этот метод не нужно вызывать явно — по умолчанию любые изменения в Адаптере данных (методы add, insert, remove) приводят к изменению внешнего вида виджета-списка и наоборот — изменения данных пользователем с помощью виджета приводит к обновлению данных в Адаптере данных.

■ **public void sort (Comparator comparator)** — сортирует набор данных Адаптера, используя в качестве критерия сортировки объект comparator.

***Классы SimpleAdapter, CursorAdapter, SimpleCursorAdapter***
(http://developer. android.com/intl/ru/reference/android/widget/SimpleAdapter. html)
(http:// developer.android.com/intl/ru/reference/android/
widget/ SimpleCursorAdapter.html).

Класс SimpleAdapter является готовым к использованию Адаптером данных. Но в отличии от ArrayAdapter, который предназначен для отображения элементов списка, содержащих одно значение, класс SimpleAdapter предназначен для отображения элементов списка, содержащих множество значений.

Класс SimpleCursorAdapter похож на android.widget.SimpleAdapter, только использует не набор объектов Map, а android.database.Cursor, т.е. набор строк с полями.

***Выпадающий список Spinner***
(http://developer.android.com/intl/ru/ reference/android/widget/Spinner.html).

Представляет собой выпадающий список, позволяет отображать только один элемент списка, который является выбранным.

Элементы, которые отображаются в выпадающем списке Spinner берутся из Адаптера данных.

Для получения мгновенной информации о смене выбранного элемента предназначено событие **onItemSelected**. Для обработки этого события, необходимо создать класс, реализующий интерфейс **AdapterView.OnItemSelectedListener,** в котором объявлены два метода:

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

public void onNothingSelected (AdapterViewparent);
```

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

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

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

Для получения информации о выбранном элементе используются методы:

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

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

**void setSelection(int position)**

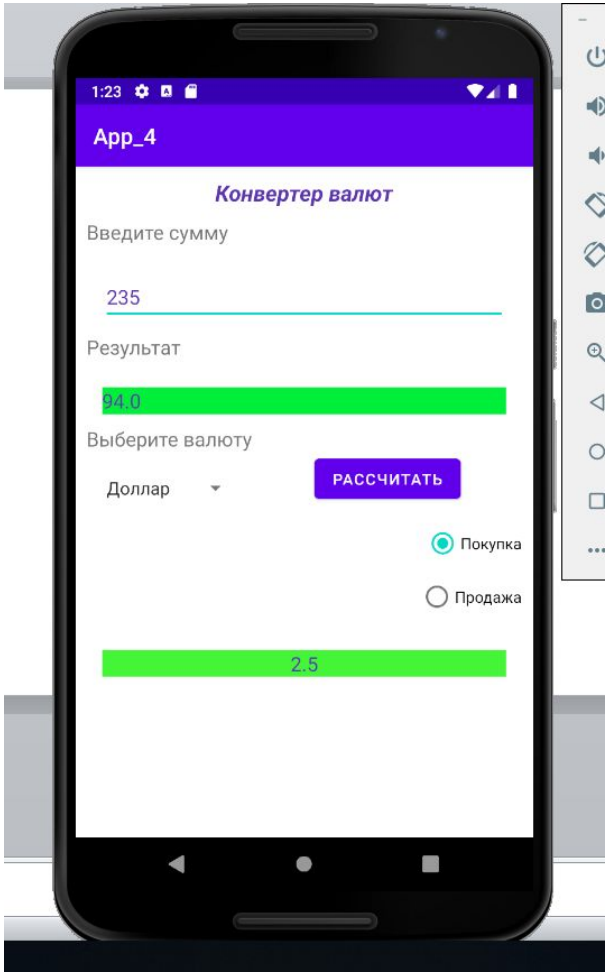
```
public class Currency {
    private String name;
    private double
purchaseRate;
    private double
sellingRte;
…
}
```

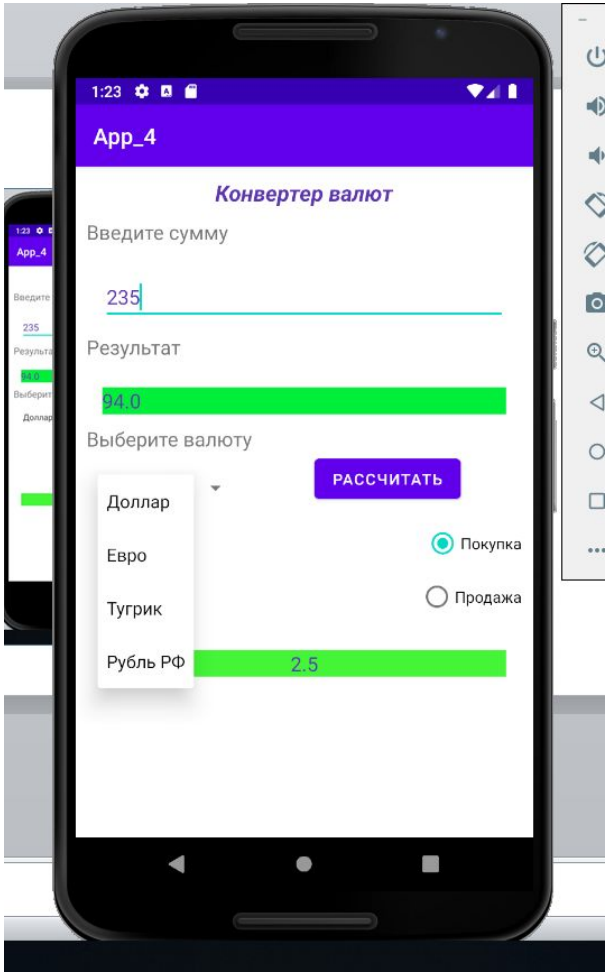
В методе onCreate:

```
spinner = (Spinner) findViewById(R.id.currencyList);
// Создаем адаптер ArrayAdapter с помощью массива строк и стандартной разметки
элемета spinner
ArrayAdapter<Currency> adapter = new ArrayAdapter<Currency>(this,
                                        android.R.layout.simple_spinner_item, currencyList);
// Определяем разметку для использования при выборе элемента
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

// Применяем адаптер к элементу spinner
spinner.setAdapter(adapter);
spinner.setSelection(0);
```

```
AdapterView.OnItemSelectedListener itemSelectedListener = new
AdapterView.OnItemSelectedListener() {
    @RequiresApi(api = Build.VERSION_CODES.N)
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

        // Получаем выбранный объект
        currentCurrency = (Currency) parent.getItemAtPosition(position);
        currentRate = getCurrency.get().doubleValue();
        rate.setText(String.valueOf(currentRate));
    }

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

```
spinner.setOnItemSelectedListener(itemSelec
tedListener);
```


1:23
App_4
Конвертер валют
Введите сумму
235
Результат
94.0
Выберите валюту
Доллар
РАССЧИТАТЬ
Покупка
Продажа
2.5


1:23
App_4
Конвертер валют
Введите сумму
235
Результат
94.0
Выберите валюту
Доллар
РАССЧИТАТЬ
Доллар
Евро
Тугрик
Рубль РФ
Покупка
Продажа
2.5

В примере не создавался макет виджета для списка. Вместо этого использовался стандартный макет (представляет собой обычный TextView), идентификатор которого задается с помощью константы **android.R.layout.simple_spinner_item**.

Если есть необходимость использовать отличные от стандартных макетов макеты для элементов списка, то необходимо :
создать файл ресурсов макета.

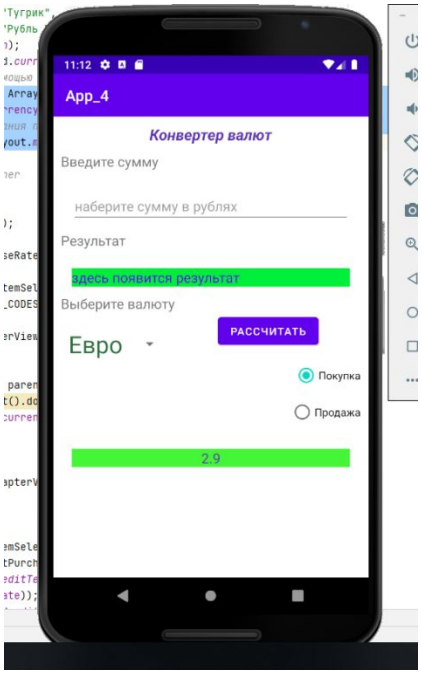
В нашем примере добавим два макета — один для обычного вида списка (файл ресурсов /res/layout/my_spinner_item.xml), второй для представления выпадающих элементов списка (файл ресурсов /res/layout/my_dropdown_spinner_item.xml).

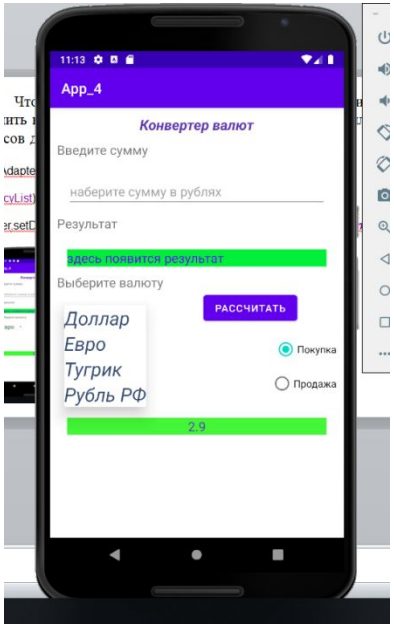
```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android=

"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#2A6F3A"
    android:textSize="14pt">
</TextView>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android=

"http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#364B6F"
    android:textSize="12pt"
    android:textStyle="italic">
</TextView>
```

Чтобы назначить эти макеты списку, необходимо заменить в нашем примере значения идентификаторов файлов ресурсов для макетов.

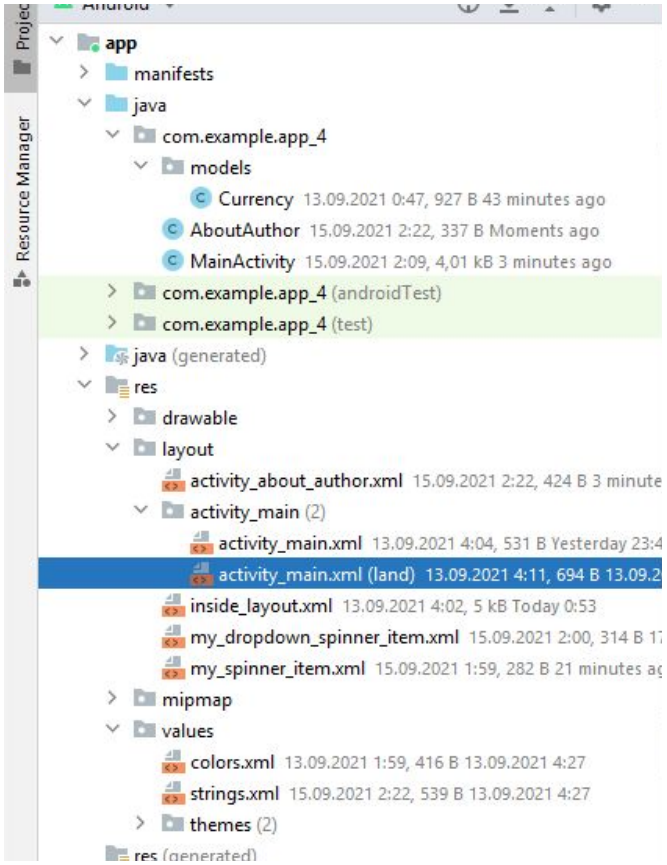
```
ArrayAdapter<Currency> adapter = new
ArrayAdapter<Currency>(this,

R.layout.my_spinner_item, currencyList);

adapter.setDropDownViewResource(R.layout.my_dropdown_spinner_
item);
```



## *Создание и запуск нескольких активностей*

Для добавления новой активности в контекстном меню папки , в которой находится класс MainActivity, нужно выбрать **New->Activity->Empty Activity**




Будут созданы два файла: активность и макет.

Затем нужно отредактировать макет активности. Например,

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".AboutAuthor">
    <TextView
        android:id="@+id/messageText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Автор - крутой Андроид разработчик!
        Присоединение телефона к компьютеру - его конек!"
        android:textSize="18sp"
        android:layout_margin="1dp"
        android:textAlignment="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ImageView
        android:layout_width="445dp"
        android:layout_height="574dp"
        android:scaleType="centerCrop"
        android:src="@drawable/IMG_0297"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Для запуска второй активности необходим объект **Intent**. Объект **Intent** представляет некоторую задачу приложения, которую надо выполнить (например, запуск activity)

Конструктор этого объекта принимает два параметра:

- Первый параметр представляет контекст - объект Context
- Вторым параметром идет класс компонента, которому мы передаем объект Intent.

Для запуска активности нужно вызвать метод **startActivity()** и передать ему в качестве параметра объект Intent.

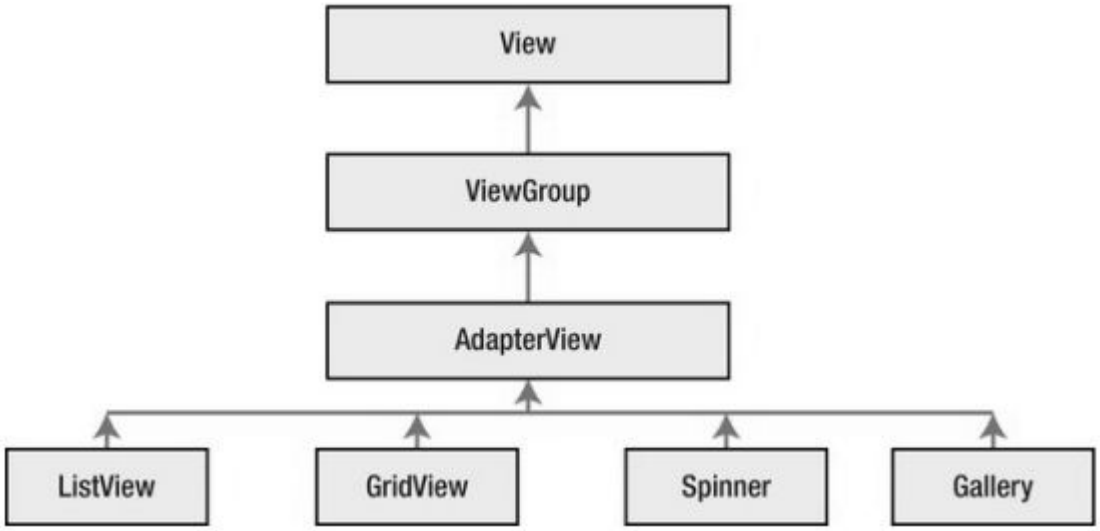
```
Intent intent = new Intent(this,
AboutAuthor.class);
startActivity(intent);
```

***ListView***

http://developer.android.com/reference/android/widget/ListView.html.

Представляет собой список элементов с вертикальной прокруткой. Данные для элементов списка поставляются из Адаптера Данных, который должен наследовать интерфейс ListAdapter

Иерархия классов для виджета ListView выглядит следующим образом



```
<ListView
   android:id="@+id/pigsList"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   app:layout_constraintTop_toBottomOf="@id/titleTextview"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   >
</ListView>
```

Создание Адаптера Данных с данными для элементов списка ListView ничем не отличается от создания Адаптера Данных для списка Spinner.

```
pigsList = (ListView) findViewById(R.id.pigsList);
// создаем адаптер
ArrayAdapter<Pig> adapter = new ArrayAdapter(this,
simple_list_item_1,  pigs);

// устанавливаем для списка адаптер
pigsList.setAdapter(adapter);
```

## *ListView одиночного выбора*

Существует возможность создавать ListView с возможностью выбора элементов.

Для этой цели необходимо, во-первых, для объекта ListView вызвать метод:

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

который принимает в качестве параметра тип выбора для списка.

- 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** или создать свой макет с радио-кнопкой и текстовым полем

## *ListView множественного выбора*

Имеется возможность создавать списки с множественным выбором элементов.

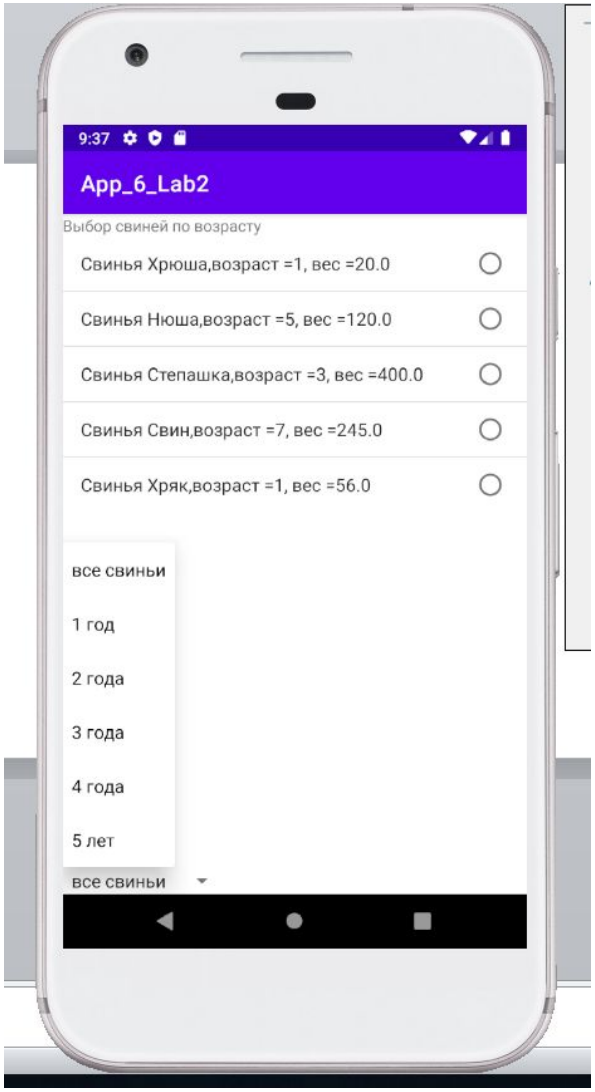
Для этого необходимо с помощью метода класса ListView:

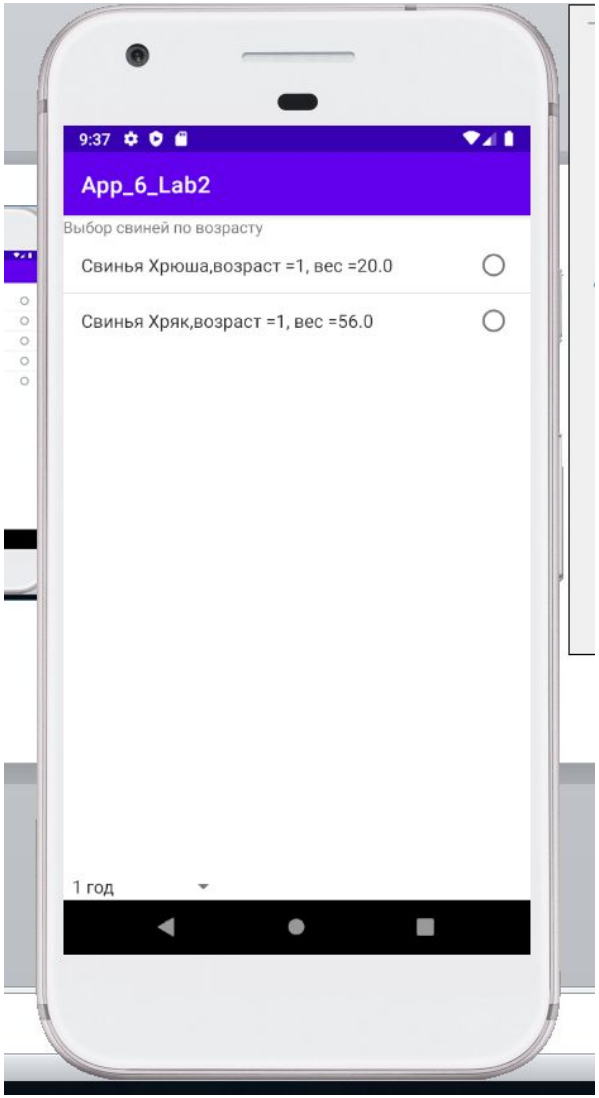
**public void setChoiceMode (int choiceMode);**

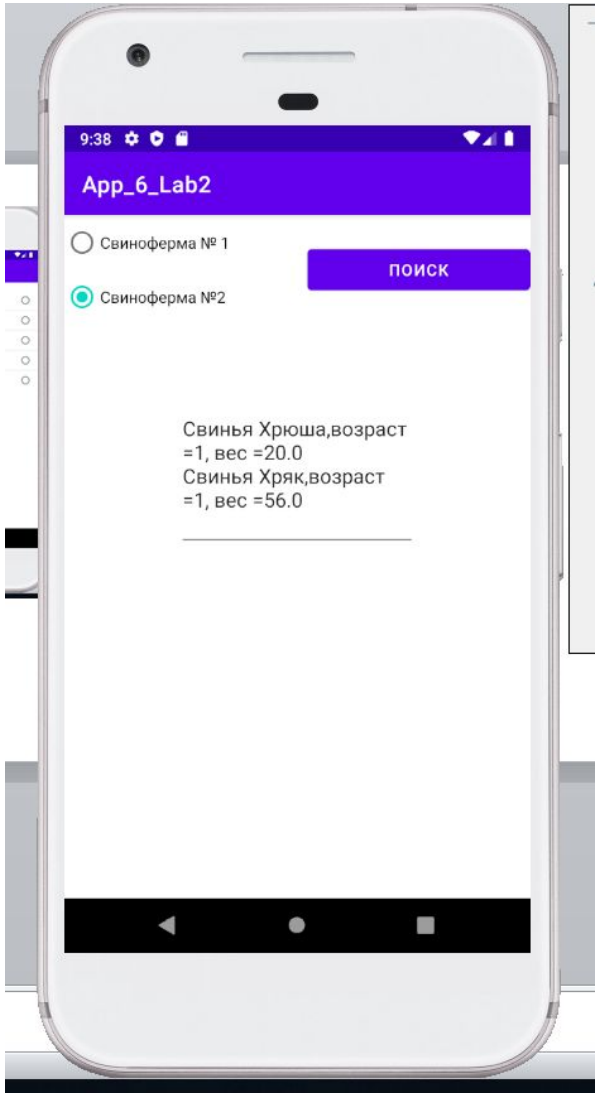
установить режим множественного выбора, передав в метод значение ListView.CHOICE_MODE_MULTIPLE.

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).

**Пример.** Разработать в Android Studio интерактивное приложение, работающее с несколькими активностями и интентами, которое должно: a) содержать текстовое поле, кнопку; b) при щелчке по кнопке приложение должно предложить выбрать активность, используемую для передачи данных. Разные активности поставляют разные данные.


9:37
App_6_Lab2
Выбор свиней по возрасту
Свинья Хрюша,возраст =1, вес =20.0
Свинья Нюша,возраст =5, вес =120.0
Свинья Степашка,возраст =3, вес =400.0
Свинья Свин,возраст =7, вес =245.0
Свинья Хряк,возраст =1, вес =56.0
все свиньи
1 год
2 года
3 года
4 года
5 лет
все свиньи


9:37
App_6_Lab2
Выбор свиней по возрасту
Свинья Хрюша,возраст =1, вес =20.0
Свинья Хряк,возраст =1, вес =56.0
1 год


9:38
App_6_Lab2
Свиноферма № 1
Свиноферма №2
поиск
Свинья Хрюша,возраст
=1, вес =20.0
Свинья Хряк,возраст
=1, вес =56.0

```
public class Pig {
   private String name;
   private int age;
   private double weight;

    public Pig(String name, int age, double weight) {
        this.name = name;
        this.age = age;
        this.weight = weight;
    }

    public String getName() {
        return name;
    }

…
    @Override
    public String toString() {
        return "Свинья " +
                name +
                ",возраст =" + age +
                ", вес =" + weight;
    }
}
```

Нужно создать две дополнительные активности, в каждой из которых будет свой набор данных. Например,

```
public class PigFarm1 extends AppCompatActivity {
    private static List<Pig> pigs = new ArrayList<Pig>();

    static {
        pigs.add(new Pig("Свинья 1", 4, 20));
        pigs.add(new Pig("Свинья 2", 1, 120));
        pigs.add(new Pig("Свинья 3", 2, 400));
        pigs.add(new Pig("Свинья 4", 5, 245));
        pigs.add(new Pig("Свинья 5", 2, 56));
    }
```

```
public class PigFarm2 extends AppCompatActivity
{
    private static List<Pig> pigs = new
ArrayList<Pig>();

    static {
        pigs.add(new Pig("Хрюша", 1, 20));
        pigs.add(new Pig("Нюша", 5, 120));
        pigs.add(new Pig("Степашка", 3, 400));
        pigs.add(new Pig("Свин", 7, 245));
        pigs.add(new Pig("Хряк", 1, 56));
    }
```

В макете первой активности для отображения информации используем виджет GridView

```
<TextView android:id="@+id/titleTextview"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">

</TextView>
<GridView
    android:id="@+id/gridview"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:numColumns="2"
    android:verticalSpacing="16dp"
    android:horizontalSpacing="16dp"
    android:stretchMode="columnWidth"

    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@id/titleTextview"
    app:layout_constraintBottom_toBottomOf="parent" />
```

В активности устанавливаем адаптер для привязки данных к виджету.

```
// получаем элемент GridView
GridView pigsList = (GridView) findViewById(R.id.gridview);
// создаем адаптер
ArrayAdapter<Pig> adapter = new ArrayAdapter<Pig>(this,
                                    android.R.layout.simple_list_item_1, pigs);
pigsList.setAdapter(adapter);
```

Устанавливаем обработчик события выбора элемента списка

```
AdapterView.OnItemClickListener itemListener = new AdapterView.OnItemClickListener() {

    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

        selectPig((Pig)parent.getItemAtPosition(position));
    }
};
pigsList.setOnItemClickListener(itemListener);
```

В макете второй активности для отображения информации используем виджет ListView, а также Spinner для выбора данных по заданному критерию

```
<TextView android:id="@+id/titleTextview"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">
</TextView>
<ListView
    android:id="@+id/pigsList"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintTop_toBottomOf="@id/titleTextview"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    >
</ListView>
<Spinner
    android:id="@+id/ageSpinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    tools:listitem="@layout/support_simple_spinner_dropdown_item"
    android:entries="@array/ages"
    />
```

Для выпадающего списка  создаем ресурс в виде массива строк:

```
<string-array name="ages">
   <item>все свиньи</item>
   <item>1 год     </item>
   <item>2 года    </item>
   <item>3 года    </item>
   <item>4 года    </item>
   <item>5 лет     </item>
</string-array>
```

Для элемента Spinner устанавливаем атрибут

```
android:entries="@array/ages"
```

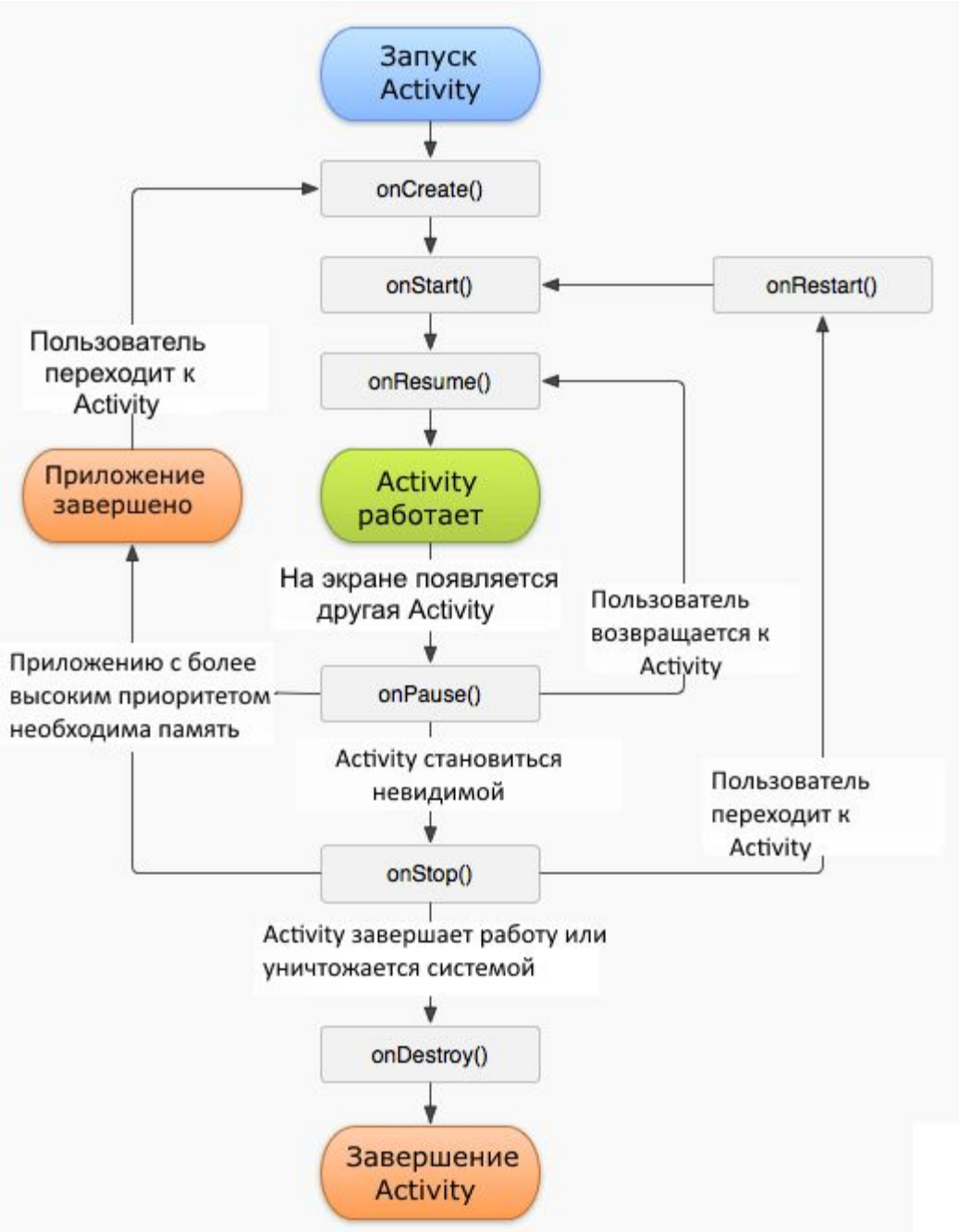
В активности устанавливаем адаптер для привязки данных к виджету.

```
pigsList = (ListView) findViewById(R.id.pigsList);
// создаем адаптер
ArrayAdapter<Pig> adapter = new ArrayAdapter(this,
       android.R.layout.simple_list_item_single_choice, pigs);

// устанавливаем для списка адаптер
pigsList.setAdapter(adapter);
```

## *Жизненный цикл активности*

Все объекты activity, которые есть в приложении, управляются системой в виде стека activity, который называется **back stack**. При запуске новой activity она помещается поверх стека и выводится на экран устройства, пока не появится новая activity.

Когда текущая activity заканчивает свою работу, она удаляется из стека, и возобновляет работу та activity, которая ранее была второй в стеке.


Запуск Activity
onCreate()
onStart()
onRestart()
Пользователь переходит к Activity
onResume()
Приложение завершено
Activity работает
На экране появляется другая Activity
Пользователь возвращается к Activity
Приложению с более высоким приоритетом необходима память
onPause()
Activity становиться невидимой
Пользователь переходит к Activity
onStop()
Activity завершает работу или уничтожается системой
onDestroy()
Завершение Activity

После запуска activity проходит через ряд событий, которые обрабатываются системой и для обработки которых существует ряд методов:

**onCreate** - метод, с которого начинается выполнение activity.

В этом методе activity переходит в состояние Created. *Этот метод обязательно должен быть определен в классе activity.*

Этот метод получает объект Bundle, который содержит прежнее состояние activity, если оно было сохранено. Если activity заново создается, то данный объект имеет значение null.

После того, как метод onCreate() завершил выполнение, activity переходит в состояние Started, и и система вызывает метод onStart()

**onStart**

В методе **onStart()** осуществляется подготовка к выводу activity на экран устройства. Как правило, этот метод не требует переопределения, а всю работу производит встроенный код.
После завершения работы метода activity отображается на экране, вызывается метод **onResume**, а activity переходит в состояние Resumed.

**onResume**

При вызове метода onResume activity переходит в состояние Resumed и отображается на экране устройства, и пользователь может с ней взаимодействовать. И activity остается в этом состоянии, пока она не потеряет фокус, например, вследствие переключения на другую activity или просто из-за выключения экрана устройства.

**onPause**

Если пользователь решит перейти к другой activity, то система вызывает метод **onPause**, а activity переходит в состояние Paused. В этом методе можно освобождать используемые ресурсы, приостанавливать процессы, например, воспроизведение аудио, анимаций, останавливать работу камеры (если она используется) и т.д., чтобы они меньше сказывались на производительность системы.

В этом состоянии activity остается *видимой* на экране, и на работу данного метода отводится очень мало времени, поэтому не стоит здесь сохранять какие-то данные, особенно если при этом требуется обращение к сети, например, отправка данных по интернету, или обращение к базе данных - подобные действия лучше выполнять в методе onStop().

После выполнения этого метода activity становится невидимой, не отображается на экране, но все еще активна. И если пользователь решит вернуться к этой activity, то система вызовет снова метод onResume.

**onStop**

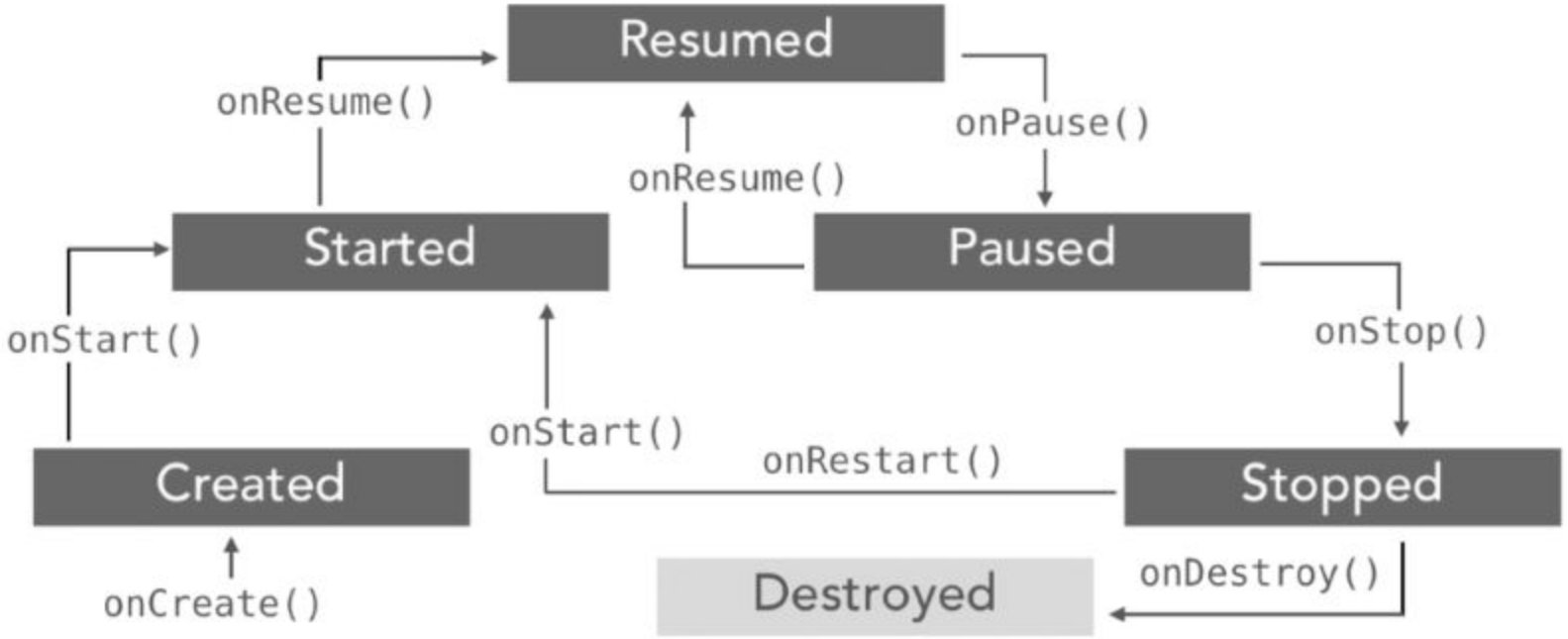
В этом методе активность переходит в состояние Stopped. И становится полностью невидима. В методе onStop следует освобождать используемые ресурсы, также можно сохранять данные, например, в базу данных.

При этом во время состояния Stopped активность остается в памяти устройства, сохраняется состояние всех элементов интерфейса.

Если после вызова метода **onStop** пользователь решит вернуться к прежней activity, система вызовет метод **onRestart**. Если же activity вовсе завершила свою работу, например, из-за закрытия приложения, то вызывается метод **onDestroy()**.

**onDestroy**

Завершается работа activity вызовом метода **onDestroy**, который возникает либо когда система решит «убить» activity в силу конфигурационных причин (например, поворот экрана или при многоконном режиме), либо при вызове метода **finish()**.


Resumed
onResume()
onPause()
onResume()
Started
Paused
onStart()
onStop()
onStart()
onRestart()
Created
Stopped
onCreate()
Destroyed
onDestroy()

Чтобы указать, какая активность является точкой входа в приложение, нужно в файле манифеста **AndroidManifest.xml** в элементе **intent-filter** установить для тега **action** атрибуту **android:name** соответствующее значение.

Элемент **category** с атрибутом android:name= "android.intent.category.LAUNCHER" указывает, что активность будет представлять стартовый экран, отображающийся при запуске приложения.

Элементы activity в файле манифеста вложены в узел application и определяют все используемые в приложении активности.

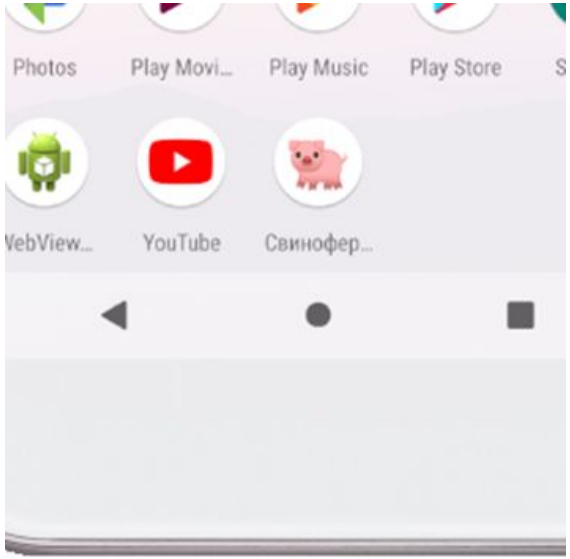
В теге application определяется большинство настроек уровня приложения

Более подробно про файл манифеста:
https://metanit.com/java/android/2.3.php
и в документации.

Атрибуты узла application:

- **android:allowBackup** указывает, будет ли для приложения создаваться резервная копия. Значение android:allowBackup="true" разрешает создание резервной копии.
- **android:icon** устанавливает иконку приложения. При значении android:icon="@mipmap/ic_launcher" иконка приложения берется из каталога **res/mipmap**
- **android:roundIcon** устанавливает круглую иконку приложения. Также берется из каталога **res/mipmap**
- **android:label** задает название приложение, которое будет отображаться на мобильном устройстве в списке приложений и в заголовке. **android:supportsRtl** указывает, могут ли использоваться специальные API для работы с правосторонней ориентацией текста (например, для таких языков как арабский или фарси).
- **android:theme** устанавливает тему приложения.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.app_6_lab2">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.App_6_Lab2">
        <activity
            android:name=".PigFarm2"
            android:exported="true" />
        <activity
            android:name=".PigFarm1"
            android:exported="true" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



## *Передача данных между Activity*

**I способ**. Для передачи данных между двумя активностями используется объект **Intent**.

Используя его метод **putExtra()** можно добавить значение и ключ для него.

В качестве значения можно передавать данные простейших типов - String, int, float, double, long, short, byte, char, массивы этих типов, либо объект интерфейса Serializable.

```
intent2 = new Intent(this, PigFarm2.class);
intent2.putExtra("title","Выбор свиней по возрасту");
```

В зависимости от типа отправляемых данных при их получении мы можем использовать ряд методов объекта Intent. Все они в качестве параметра принимают ключ объекта.

**getExtra()**: универсальный метод, который возвращает значение типа Object.

**getStringExtra()**: возвращает объект типа String

**getInt Extra()**: возвращает значение типа int и т.п.

Старт второй активности: *startActivity(intent2);*

Во второй активности:

```
((TextView)findViewById(R.id.titleTextview
))
        .setText(getIntent()
        .getStringExtra("title"));
```

Для передачи сложных данных используется механизм сериализации.

Передавать можно объекты только тех классов, которые реализуют интерфейс **Serializable**.

Пусть public class Pig implements Serializable …

```
Pig pig=new Pig("Свин",67,150);
intent2.putExtra("pig",pig);
```

В другой активности:

```
Pig mainPig=(Pig)
    getIntent().getSerializableExtra("pig");
    ((TextView)findViewById(R.id.titleTextview))
            .setText("Главная свинья:
    "+mainPig.toString());
```

Получить результат работы активности при возврате к предыдущей активности можно, переопределив ее метод **onActivityResult().**

Установить значение в качестве результата активности нужно с помощью метода **setResult().**

Например,

```
setResult(RESULT_OK,getIntent().putExtra("pigs",pigsString.toString()));
```

В **setResult** передаем константу **RESULT_OK**, означающую **успешное завершение** вызова. И именно она будет передаваться в параметр **resultCode** метода **onActivityResult. Второй параметр – объект Intent с записанными в него результирующими данными.**

Активность, результат которой нужно получить, запускается методом **startActivityForResult**.

```
startActivityForResult(intent2, 0);
```

В **startActivityForResult** в качестве параметров передается **Intent** и **requestCode (используются для идентификации запроса в методе onActivityResult)**.

Пример переопределения метода **onActivityResult:**

```
@Override
protected void onActivityResult(int requestCode, int
resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (resultCode == RESULT_OK && data != null) {
        String result = data.getStringExtra("pigs");
        // используем result
        ((EditText)
findViewById(R.id.FindResult)).setText(result);
    } else {
        // не удалось получить результат
        ((EditText)
findViewById(R.id.FindResult)).setText("Ничего не найдено");
    }
}
```

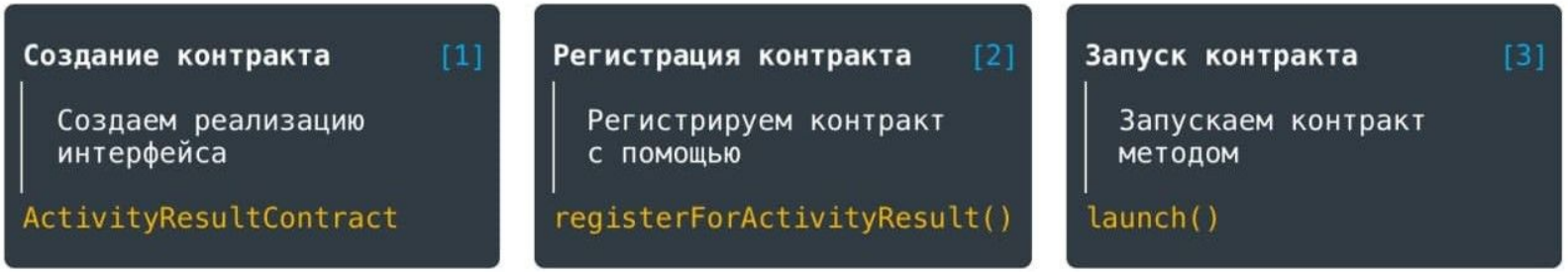
В **onActivityResult** следующие параметры:

- **requestCode** – тот же идентификатор, что и в startActivityForResult. По нему определяется, с какой Activity пришел результат.
- **resultCode** – код возврата. Определяет успешно прошел вызов или нет.
- **data** – Intent, в котором возвращаются данные

*Метод onActivityResult() нарушает принцип единственной ответственности!*

**II способ**. В 2020 году Google представила **Activity Result API**. Это инструмент для обмена данными между активностями. https://habr.com/ru/company/e-Legion/blog/545934/

Применение Activity Result состоит из трех шагов:




- **Контракт** — это класс, реализующий интерфейс **ActivityResultContract<I,O>**.

Где I определяет тип входных данных, необходимых для запуска Activity, а O — тип возвращаемого результата.

Существуют стандартные реализации этого интерфейса: PickContact, TakePicture, RequestPermission и др.

При создании контракта необходимо реализовать два его метода:

**createIntent()** — принимает входные данные и создает объект Intent, который будет в дальнейшем запущен вызовом launch()

**parseResult()** — отвечает за возврат результата, обработку resultCode и парсинг данных

Метод **getSynchronousResult()** можно переопределить в случае необходимости.

Он позволяет сразу же, без запуска Activity, вернуть результат, например, если получены неправильные входные данные. Если подобное поведение не требуется, метод по умолчанию возвращает null.

Например,

```
public class PigFarmContract extends
ActivityResultContract<String,String> {

    @NonNull
    @Override
    public Intent createIntent(@NonNull Context context, String input) {
        return new Intent(context, PigFarm1.class).putExtra("title",input);
    }

    @Override
    public String parseResult(int resultCode, @Nullable Intent intent) {
        if(resultCode == Activity.RESULT_OK)
            return intent.getStringExtra("pig");
        return null;
    }
}
```

- **Регистрация** контракта в активности (или фрагменте) осуществляется с помощью вызова метода **registerForActivityResult()**.

В параметры необходимо передать **ActivityResultContract** и **ActivityResultCallback**.

Callback сработает при получении результата.

Регистрация контракта не запускает новую активность, а лишь возвращает специальный объект **ActivityResultLauncher**, который нужен для запуска активности.

```
//регистрация контракта активности
pigFarm1StarterForResult = registerForActivityResult(new PigFarmContract(),
        new ActivityResultCallback<String>() {
            @Override
            public void onActivityResult(String result) {
                // обработка результата
                ((EditText) findViewById(R.id.FindResult)).setText(result);
            }
        });
```

Здесь pigFarm1StarterForResult - поле

```
ActivityResultLauncher<String>
pigFarm1StarterForResult;
```

- Для **запуска** активности нужно вызвать метод **launch()** на объекте **ActivityResultLauncher**, который получен при регистрации**.**

```
pigFarm1StarterForResult.launch("Выберите себе
свинью");
```

Регистрировать контракты можно в любой момент жизненного цикла активности, но запустить его до перехода в состояние CREATED нельзя. Общепринятый подход — регистрация контрактов как полей класса.

Не рекомендуется вызывать метод registerForActivityResult() внутри операторов if .

Во время ожидания результата процесс приложения может быть уничтожен системой (например, при открытии камеры, которая требовательна к оперативной памяти). И если при восстановлении процесса мы не зарегистрируем контракт заново, результат будет утерян.

***Объект намерения Intent***

***http://java-online.ru/android-intent.xhtml***

**Intent** представляет собой объект описания операции, которую необходимо выполнить через систему Android. Т.е. необходимо сначала описать некоторую операцию в виде объекта *Intent*, после чего отправить её на выполнение в систему вызовом одного из методов активности *Activity*.

Объекты Intent в основном используются в трёх операциях :

- **Старт операции**

  Например, переход от одной активности к другой выполняется с помощью объекта *Intent*, который передается методу *startActivity()* или методу *startActivityForResult()*, который вернет отдельный объект *Intent* в обратном вызове метода *onActivityResult()*.

- **Запуск сервиса**

Android-приложение может выполнять действия в фоновом режиме без пользовательского интерфейса с помощью определенного сервиса, в качестве которого используется компонент *Service*. Сервис можно стартовать для выполнения какого-либо действия, например, чтение файла. Для старта сервиса необходимо вызвать метод активности *startService()* и передать ему объект *Intent*.

Если сервис имеет интерфейс типа клиент-сервер, то можно с ним установить связь через вызов метода *bindService()*, с передачей ему в качестве параметра объекта *Intent*.

- **Рассылка широковещательных сообщений**

Широковещательное сообщение может принимать любое приложение android. Система генерирует различные широковещательные сообщения о системных событиях, например, зарядка устройства. Широковещательные сообщения отправляются другим приложениям передачей объекта *Intent* методам sendBroadcast(), sendOrderedBroadcast() или sendStickyBroadcast().

Существует два типа объектов *Intent* : **явные и неявные**.

Явные объекты *Intent* в основном используются для запуска компонента из собственного приложения, где известно наименование запускаемых классов и сервисов.

```
Intent intent2 = new Intent(this, PigFarm2.class);
```

Неявные объекты *Intent* не содержат имени конкретного класса. Вместо этого они включают действие (action), которое требуется выполнить. Неявный *Intent* может включать дополнительно наименование категории (category) и тип данных (data).

Это позволяет компоненту **из другого приложения** обработать этот запрос. Например, если необходимо пользователю показать место на карте, то с помощью неявного объекта *Intent* можно попросить это сделать другое приложение, в котором данная функция предусмотрена.

Действие *action* объекта *Intent* определяет, что нужно выполнить, например просмотр фото (view) или выбор фото (pick). В значительной степени действие определяет, каким образом описана остальная часть намерения *Intent*, в частности, что именно содержится в разделе данных. Действие для объекта *Intent* можно указать методом **setAction()** или определить в конструкторе *Intent*.

Простой пример формирования неявного намерения для открытия определенной страницы сайта

```
public void goToGSTU(View view) {
    String url = "http://gstu.by";
    Uri address = Uri.parse(url);
    Intent urlIntent = new
Intent(Intent.ACTION_VIEW, address);

    if
(urlIntent.resolveActivity(getPackageManager())≠nu
ll)
        startActivity(urlIntent);
    else
        Log.d("Intent", "Intent (ACTION_VIEW) не
обработан!");
}
```

/apk/res/

# Константные значения действий, используемые в java-коде:

| | |
|---|---|
| ACTION_ANSWER | Открывает активность, связанную с входящими звонками. |
| ACTION_CALL | Открывает активность, инициализирующую обращение к телефону. |
| ACTION_HEADSET_PLUG | Подключение наушников. |
| ACTION_SEND | Вызов активности для отправки данных, указанных в намерении. Получатель должен быть определен с помощью полученной активности. Для указания типа передаваемых данных MIME используйте метод setType. Данные передаются параметром намерения extras с ключами EXTRA_TEXT или EXTRA_STREAM в зависимости от типа. При использовании электронной почты стандартное приложение принимает дополнительные параметры по ключам EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC и EXTRA_SUBJECT. Действие ACTION_SEND следует использовать только в тех случаях, когда данные необходимо передать удаленному адресату, а не другой программе на том же устройстве. |
| ACTION_SENDTO | Открывает активность для отправки сообщений контакту, указанному в пути URI, который определяется в намерении. |
| ACTION_VIEW | Наиболее распространенное общее действие просмотра чего-либо. Выбор приложения зависит от схемы (протокола) данных : стандартные адреса http будут открывать браузер, адреса tel вызовут приложение для дозвона, geo откроет программу Google Maps, а данные с контактами откроют приложении для управления контактной информацией. |
| ACTION_WEB_SEARCH | Открывает активность поиска информации в интернете, основываясь на передаваемых данных с помощью пути URI; как правило, при этом запускается браузер. |

Действие, определяемое в манифесте, включает наименование пакета и не используют префикс "ACTION_", например, «android.intent.action.SEND».

Категория – это строка, содержащая дополнительные сведения о том, каким компонентом должна выполняться обработка объекта *Intent*. В объект *Intent* можно поместить любое количество категорий.

В файле манифеста используются стандартные значения, предоставляемые системой:

| | |
|---|---|
| BROWSABLE | Операция стартует веб-браузер для отображения данных, указанных по ссылке (url), например, изображение или сообщение электронной почты. |
| LAUNCHER | Операция (активность) является начальной операцией задачи. Активность с данной категорией помещается в окно для запуска приложений. |
| DEFAULT | Данная категория позволяет объявить компонент обработчиком по умолчанию для действия, выполняемого с указанным типом данных внутри фильтра намерений. |
| HOME | Активность с данной категорией отображает главный экран (Home Screen), который открывается после включения устройства и загрузки системы, или когда нажимается клавиша HOME. |

При создании объекта *Intent* в некоторых случая необходимо определить данные (data).

Например, для действия ACTION_EDIT, данные должны включать URI документа, который требуется отредактировать. Объект Uri ссылается на данные и/или тип этих данных MIME. Так, например, операция, которая выводит на экран изображения, скорее всего, не сможет воспроизвести аудио/видео файл, даже если и у тех, и у других данных будут одинаковые форматы URI. Поэтому указание типа данных MIME помогает системе Android найти наиболее подходящий компонент для выполнения объекта Intent.

Когда android получает неявный объект *Intent* для выполнения, то система ищет подходящие компоненты путем сравнения содержимого *Intent* с фильтрами *Intent* других приложений, зарегистрированных в файлах манифестов. Если параметры объекта *Intent* совпадают с параметрами одного из фильтров *Intent*, то система запускает этот компонент и передает ему объект *Intent*. При наличии нескольких подходящих фильтров система открывает диалоговое окно, где пользователь может выбрать подходящее приложение.

Фильтр *Intent* представляет собой секцию в файле манифеста приложения, описывающее типы объектов *Intent*, которые компонент мог бы выполнить. Наличие фильтра *Intent* в описании активности в манифесте позволяет другим приложениям напрямую запускать данную операцию с помощью некоторого объекта *Intent*. Если фильтр *Intent* не описан, то операцию можно будет запустить только с помощью явного объекта *Intent*.

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action   android:name="android.intent.action.SEND"/>
        <category android:name=
            "android.intent.category.DEFAULT"/>
        <data     android:mimeType="text/plain"/>
    </intent-filter>
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <action android:name=
            "android.intent.action.SEND_MULTIPLE"/>
        <category android:name=
            "android.intent.category.DEFAULT"/>
        <data android:mimeType=
            "application/vnd.google.panorama360+jpg"/>
        <data android:mimeType="image/*"/>
        <data android:mimeType="video/*"/>
    </intent-filter>
</activity>
```

Активность ShareActivity предназначена для упрощения обмена текстовым и мультимедийным контентом.

Несмотря на то, что пользователи могут входить в эту активность из MainActivity, они также могут открыть ShareActivity напрямую из другого приложения, которое создаст неявный объект *Intent*, соответствующий одному из двух фильтров *Intent*.

***Меню в Android приложениях***

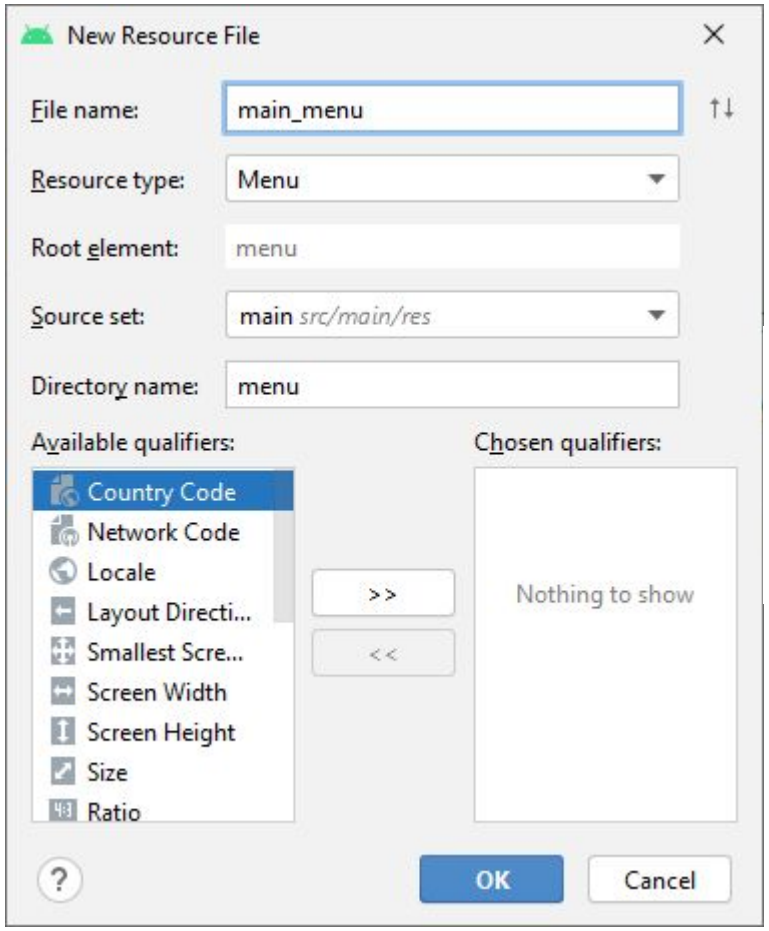
***http://java-online.ru/android-menu.xhtml***

Android поддерживает два типа меню : главное и контекстное

Меню представляет класс **android.view.Menu**, и каждая активность ассоциируется с объектом этого типа. Объект Menu может включать различное количество элементов, а те в свою очередь могут хранить другие элементы.

Меню представляет собой ресурс. При создании нового проекта с Empty Activity по умолчанию нет никаких ресурсов меню, при необходимости их нужно добавлять вручную.

Для определения ресурсов меню в контекстном меню папки **res** нужно выбрать пункт **New -> Android Resource File**:




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

</menu>
```

В директории **res/menu/** будет создан файл *main_menu.xml*.

Описание меню включает следующие секции :

**<menu>** корневой элемент в XML-структуре файла; может содержать один или несколько элементов <item> и <group>;

**<item>** непосредственный пункт меню; элемент может иметь вложенный элемент <menu> для создания подменю;

**<group>** невидимый контейнер для пунктов меню; позволяет выполнять группирование элементов (необязательная секция).

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto">
   <item
      android:id="@+id/bigPigs"
      android:title="Большие свиньи"
      app:showAsAction="always" />
   <item
      android:id="@+id/smallPigs"
      android:title="маленькие свиньи"
      app:showAsAction="never" />
</menu>
```

Каждый пункт меню включает следующие атрибуты :

• **id** идентификатор пункта меню, по которому приложение может распознать выделенный пользователем пункт меню;
• **title** отображаемый текст;
• **icon** отображаемый рядом с текстом иконка;
• **showAsAction** определяет поведение меню в *ActionBar*;
• **enabled** определение доступности пункта меню;
• **orderInCategory** порядок отображения в *ActionBar*
• **titleCondensed** атрибут применяется в том случае, если обычный заголовок слишком широкий и не «помещается» в выбранном элементе меню.

*MainActivity.java* может иметь только одно меню. Если приложение имеет несколько экранов, то у каждой активности должно быть отдельное меню со своими настройками. Для подключения меню к активности необходимо переписать метод активности ***onCreateOptionsMenu***.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.main_menu, menu);
    return true;
}
```

Метод getMenuInflater возвращает объект MenuInflater, у которого вызывается метод inflate(). Этот метод в качестве первого параметра принимает ресурс, представляющий описание меню в xml, и наполняет им объект menu, переданный в качестве второго параметра.