Activity и Layouts

Визуальную часть приложения можно назвать самой важной — если приложение не будет красивым и удобным, то пользователь вряд ли будет его использовать, какие крутые фичи в себе оно не содержало бы. Поэтому нужно изучить составляющие этой самой визуальной части, прежде чем переходить непосредственно к программированию.

Два самых важных понятия в интерфейсе Android — это Activity и View.

Activity — это та часть приложения, с которой взаимодействует пользователь. Можно назвать ее «окном» в терминологии десктопных ОС *(хотя фактически это не окно, Window в андроиде тоже есть, но с ним мало кто из разработчиков сталкивался)*. Внутри Activity расположены дочерние элементы интерфейса. К activity мы позже обязательно вернемся и рассмотрим это понятие в отдельном уроке.

View — элемент интерфейса. То же самое, что и в любой другой ОС. Это может быть кнопка, поле для ввода текста, контейнер для картинки, контейнер для других View и т.д.

Так же немаловажный элемент — ViewGroup. Напрямую начинающие с ним не сталкиваются, обычно с этим классом работают более опытные разработчики. Фактически, ViewGrop — это модифицированный View, созданный для того, чтобы служить контейнером для других View. Тут мы уже знакомимся с понятием Layout.

Layouts в Android

Layout — общее название для нескольких наследников ViewGroup. Лэйауты служат контейнерами для View, и созданы они для того, чтобы мы могли удобно располагать всяческие кнопочки, поля для ввода текста и прочие элементы интерфейса.

«Самых нужных» лэйаута всего 3:

- LinearLayout
- FrameLayout
- RelativeLayout

В большинстве приложений они используются в качестве layout'ов в, наверное, 90% случаев.

Конечно же, на самом деле их больше — у класса **ViewGroup** 44 прямых наследника.

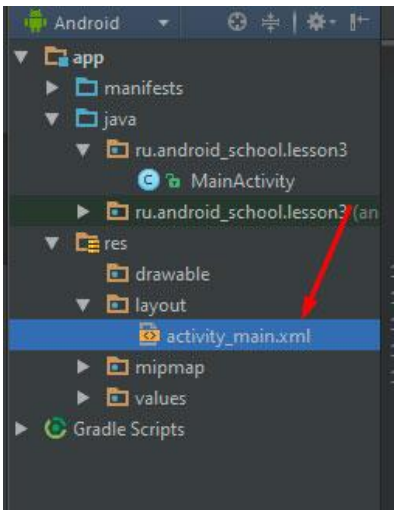
Посмотрите на следующий рисунок:

Кнопки, поле для ввода — это **View**. Зеленая рамочка вокруг кнопок — это границы **ViewGroup**, внутри которой находятся кнопки. В свою очередь, ViewGroup с кнопками и поле для ввода находятся внутри другого ViewGroup, границы которого обозначены красным.

А весь «фон» в виде клеточек — это Activity, внутри которого находятся все остальные элементы интерфейса.

Создание интерфейса в Android

В Android принято использовать декларативный подход к созданию интерфейса, когда это возможно. Под декларативным подходом подразумевается описание интерфейса в XML-файлах. Файлы находятся в директории res/layout/:



Так же есть особый подход к именованию файлов. В отличие от исходников на Java, в ресурсах не предусмотрено вложенности директорий, поэтому все файлы лежат в одной директории и чтобы не запутаться в них, когда их много, приняты следующие названия:

- **activity_**name.xml — для **Activity**
- **fragment_**name.xml — для **фрагментов** *(о них мы обязательно поговорим в следующих статьях)*
- **view_**name.xml — для **View**

Здесь «name» — имя элемента интерфейса. Например, для LoginActivity файл будет называться activity_login.xml, для MainFragment — fragment_main.xml и т. д.

При создании проекта с пустой Activity у нас по умолчанию создастся MainActivity и xml-файл с описанием этой активити. Давайте откроем этот файл:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      android:layout_width="match_parent"
5      android:layout_height="match_parent"
6      android:paddingBottom="@dimen/activity_vertical_margin"
7      android:paddingLeft="@dimen/activity_horizontal_margin"
8      android:paddingRight="@dimen/activity_horizontal_margin"
9      android:paddingTop="@dimen/activity_vertical_margin"
10     tools:context="ru.android_school.lesson2.MainActivity">
11
12     <TextView
13         <strong>android:layout_width</strong>="wrap_content"
14         <strong>android:layout_height</strong>="wrap_content"
15         android:text="Hello World!" />
16 </RelativeLayout>
```

Тут вы видите RelativeLayout — главный контейнер для всего контента в Activity, и TextView, который находится внутри этого контейнера.

Обратите внимание на атрибуты **layout_width** и **layout_height**. Этими атрибутами, как нетрудно догадаться, мы задаем ширину и высоту элемента. В абсолютных значениях они задаются редко, как правило используются две константы:

- **match_parent** — элемент будет занимать все доступное ему пространство.
- **wrap_content** — элемент будет использовать столько места, сколько требуется для отображения контента внутри. Кнопка, например, будет иметь размер текста + отступы.

LinearLayout

LinearLayout, как следует из названия, располагает дочерние элементы в «линейном» порядке, т.е. друг за другом. Линейный лэйаут может быть горизонтальным или вертикальным.

Давайте посмотрим на практике, что это такое.

Удалим весь шаблонный код, и вставим вместо него следующий код:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent"
5     android:orientation="horizontal">
6
7     <Button
8        android:layout_width="wrap_content"
9        android:layout_height="wrap_content"
10       android:text="Button 1" />
11
12    <Button
13       android:layout_width="wrap_content"
14       android:layout_height="wrap_content"
15       android:text="Button 2" />
16
17    <Button
18       android:layout_width="wrap_content"
19       android:layout_height="wrap_content"
20       android:text="Button 3" />
21
22 </LinearLayout>
```

Как видите, мы создали LinearLayout и внутри него поместили три кнопки. Обратите внимание на атрибут ***orientation*** у LinearLayout. Он обозначает «направление» контента в лэйауте. Атрибут orientation может принимать два значения — «*horizontal*» и «*vertical*«. Как нетрудно догадаться, в первом случае дочерние элементы будут расположены горизонтально слева направо, во втором — вертикально сверху вниз.

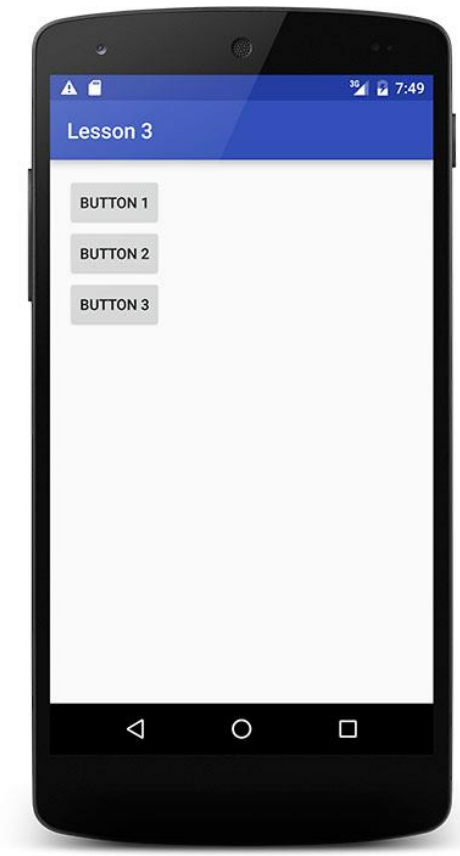
В этом примере мы расположили элементы горизонтально. Запустите приложение, и увидите следующую картину:

Теперь давайте изменим ориентацию с горизонтальной на вертикальную:

activity_main.xml

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

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 1" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 2" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button 3" />

</LinearLayout>
```

После запуска проекта мы увидим следующее:



Т.е. теперь элементы расположены вертикально.

У LinearLayout (а точнее, у его дочерних View) есть еще один интересный атрибут — **layout_weight**. Этим атрибутом мы говорим лэйауту, сколько пространства должен занимать элемент. В качестве значения можно использовать любое число. Например, если мы хотим равномерно распределить пространство между двумя кнопками, мы можем задать обеим кнопкам **layout_weight** = 1. Тогда они разделят имеющееся пространство на две равных части. Если мы зададим одной кнопке вес = 1, а второй = 2, то вторая кнопка будет занимать в 2 раза больше места, чем первая. Чтобы окончательно понять, как это работает, давайте посмотрим на примере:

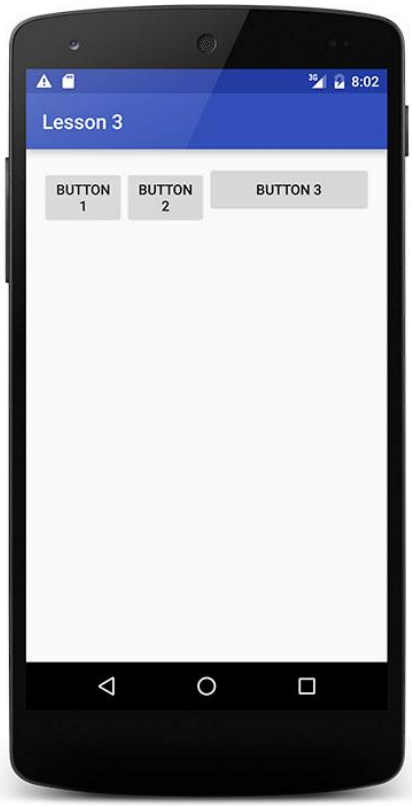
activity_main.xml

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

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 1" />

    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Button 2" />
```

```
18
19    <Button
20        android:layout_width="0dp"
21        android:layout_height="wrap_content"
22        android:layout_weight="2"
23        android:text="Button 3" />
24
25 </LinearLayout>
```

Также при использовании атрибута layout_weight рекомендуется заменить ширину (если лэйаут горизонтальный) или высоту (если лэйаут вертикальный) на 0dp. О том, что такое dp, мы поговорим в следующих уроках.

Как видите, первым двум кнопкам мы задали вес = 1, а третьей = 2. Сумма весов = 4, соответственно, первые две кнопки займут левую половину экрана, а третья — правую половину, т.е. ее ширина будет в два раза больше других кнопок. Посмотрим, что получилось:



Получилось не очень симпатично, потому что у первых двух кнопок текст не влез в заданную ширину и часть перенеслась на новую строку, поэтому эти кнопки стали «выше». Однако, суть ясна — третья кнопка в два раза шире, чем остальные.

Для вертикальных лэйаутов это работает точно так же, только меняется, соответственно, высота.

FrameLayout

Пожалуй, это самый простой Layout. Все, что он умеет — располагать элементы друг над другом (по оси «z»). Давайте вспомним немного математики, а точнее, систему координат.

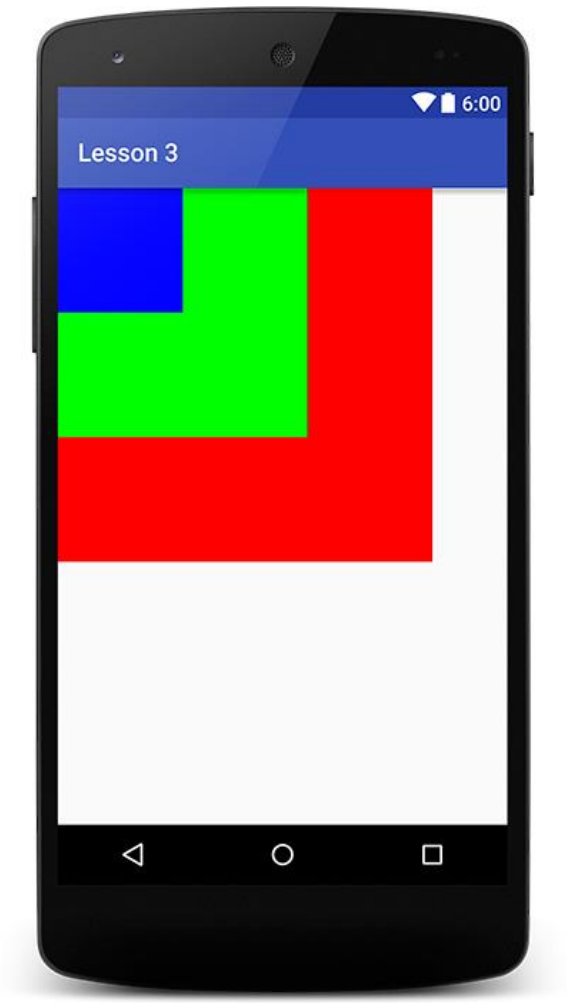
В двухмерном пространстве у нас есть две оси — X и Y. X идет слева направо, Y снизу вверх. В Android немного иначе, Y идет сверху вниз.

В трехмерном пространстве добавляется ось Z. Она идет «на нас». В интерфейсах Z обозначает глубину. Напрямую «глубина» задается редко, однако, например, во FrameLayout она есть. Давайте посмотрим на примере, как это работает. Измените код, чтобы он выглядел вот так:

activity_main.xml

```
 1  <?xml version="1.0" encoding="utf-8"?>
 2  <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 3     android:layout_width="match_parent"
 4     android:layout_height="match_parent">
 5
 6     <View
 7        android:layout_width="300dp"
 8        android:layout_height="300dp"
 9        android:background="#ff0000"
10        android:textSize="20sp" />
11
12     <View
13        android:layout_width="200dp"
14        android:layout_height="200dp"
15        android:background="#00ff00"
16        android:textSize="20sp" />
17
18     <View
19        android:layout_width="100dp"
20        android:layout_height="100dp"
21        android:background="#0000ff"
22        android:textSize="20sp" />
23
24 </FrameLayout>
```

Запустим проект, и увидим следующее:



Первым мы создали красный квадрат. Он находится «дальше» всех от нас. Вторым создали зеленый, он находится «над» красным квадратом. Ну и больше всех координата Z у синего квадрата.

Если мы поменяем элементы местами, у них изменится и координата Z.

У **FrameLayout**, как и у многих других лэйаутов, включая LinearLayout, есть понятие **gravity**. «Гравитация» может быть задана двумя способами:

- Атрибутом **gravity** у лэйаута. В таком случае она будет применена для всех дочерних элементов
- Атрибутом **layout_gravity** у дочернего элемента. Тогда она будет применена только для этого элемента.

Gravity задает положение элемента внутри контейнера. Гравитация может быть следующей:

- **bottom** — элемент «прижимается» к нижней границе контейнера.
- **center** — элемент располагается в центре контейнера
- **center_horizontal** — элемент находится в центре по оси X

- **center_vertical** — элемент находится в центре по оси Y
- **end** — элемент находится «в конце» контейнера. Обычно это означает, что он будет находиться справа, но на локали с написанием справа-налево он будет находиться слева.
- **start** — элемент находится «в начале» контейнера. Обычно — слева, на RTL локалях — справа.
- **top** — элемент «прижимается» к верхней границе контейнера.

left и right использовать не рекомендуется, поскольку это вызовет проблемы с верстской на RTL локалях.

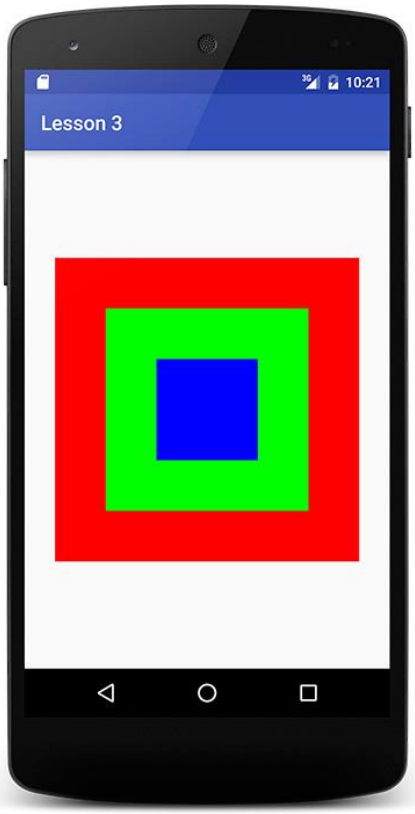
Существует еще несколько типов гравитации, но о них вам пока что знать не нужно, дабы не засорять память лишней информацией. Потом об этих типах вы с легкостью узнаете со страниц официальной документации.

Итак, давайте же попробуем на практике поработать с гравитацией. К сожалению, сам FrameLayout, в отличие от многих других контейнеров, не поддерживает атрибут gravity, однако гравитацию можно реализовать через дочерние элементы. Добавьте атрибут **layout_gravity** со значением ***center*** для каждого дочернего View:

activity_main.xml

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

    <View
        android:layout_width="300dp"
        android:layout_height="300dp"
        android:background="#ff0000"
        android:layout_gravity="center"
        android:textSize="20sp" />

    <View
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_gravity="center"
        android:background="#00ff00"
        android:textSize="20sp" />

    <View
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center"
        android:background="#0000ff"
        android:textSize="20sp" />

</FrameLayout>
```

Запустите проект:



Дочерние элементы выровнялись по центру.

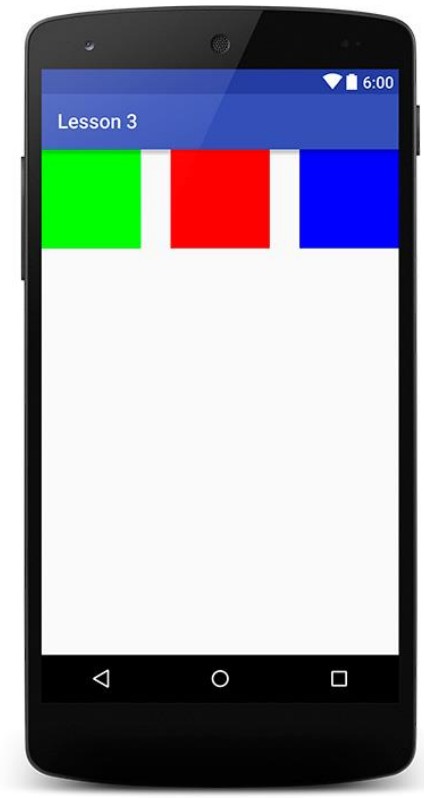
Теперь задайте всем элементам высоту в 100 dp, и давайте «раскидаем» их равномерно по углам и центру экрана. Пусть красный элемент будет расположен вверху по центру, зеленый — наверху слева, синий — наверху справа. Соответственно, у них должна быть задана гравитация start, center_horizontal и end:

activity_main.xml

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

    <View
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:background="#ff0000"
        android:layout_gravity="center_horizontal"
        android:textSize="20sp" />

    <View
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="start"
        android:background="#00ff00"
        android:textSize="20sp" />

    <View
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="end"
        android:background="#0000ff"
```

```
25        android:textSize="20sp" />
26
27 </FrameLayout>
```

Получится следующая картина:



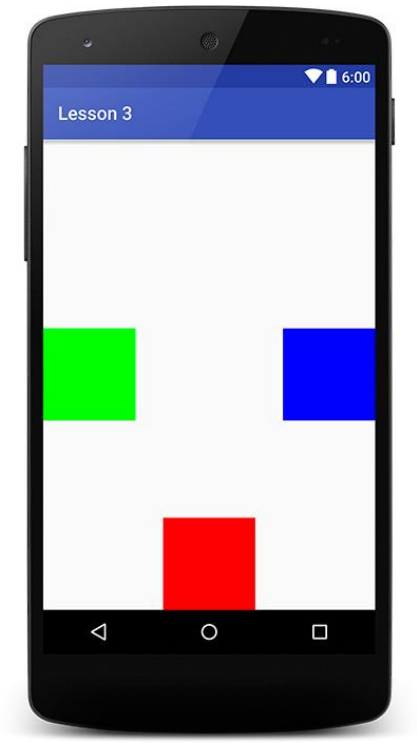
А еще атрибуты gravity можно комбинировать. К примеру, мы хотим, чтобы красный квадрат был по центру внизу, а остальные — слева и справа по центру. Тогда нам надо будет написать следующий код:

activity_main.xml

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
 3     android:layout_width="match_parent"
 4     android:layout_height="match_parent">
 5
 6     <View
 7         android:layout_width="100dp"
 8         android:layout_height="100dp"
 9         android:background="#ff0000"
10         android:layout_gravity="center_horizontal|bottom"
11         android:textSize="20sp" />
12
13     <View
14         android:layout_width="100dp"
15         android:layout_height="100dp"
16         android:layout_gravity="start|center_vertical"
17         android:background="#00ff00"
18         android:textSize="20sp" />
19
20     <View
21         android:layout_width="100dp"
```

```
22        android:layout_height="100dp"
23        android:layout_gravity="end|center_vertical"
24        android:background="#0000ff"
25        android:textSize="20sp" />
26
27 </FrameLayout>
```

И мы получим желаемое:



Что такое RelativeLayout?

RelativeLayout — прямой наследник ViewGroup, в котором дочерние элементы располагаются относительно друг друга или же самого RelativeLayout.

Доступны следующие варианты расположения дочерних элементов:

1. Относительно самого RelativeLayout. За это отвечают атрибуты ***layout_alignParentStart***, ***layout_alignParentEnd***, ***layout_alignParentTop***, ***layout_alignParentBottom***, ***layout_centerVertical***, ***layout_centerHorizontal***, ***layout_centerInParent***.
2. Относительно других элементов внутри RelativeLayout. Для этого используются атрибуты ***layout_toStartOf***, ***layout_toEndOf***, ***layout_above***, ***layout_below***, ***layout_alignStart***, ***layout_alignEnd***, ***layout_alignTop***, ***layout_alignBottom***.

Прежде чем попробовать это на практике, давайте поговорим об еще одной вещи в XML-разметке интерфейсов.

Что такое id у View в Android?

При создании интерфейса для Android-приложений мало просто создать интерфейс, нам нужно еще и как-то взаимодействовать с его элементами. А чтобы взаимодействовать с элементами интерфейса, нам надо как-то отличать их друг от друга. Для этого существует механизм присваивания id. Для каждого View может быть присвоен атрибут android:id, в котором мы задаем уникальный в пределах Activity идентификатор. Id начинается с «@+id/» и может состоять из латинских символов, цифр и знака подчеркивания. Начинаться ID должен с буквы. Может быть использован любой регистр, но принято писать в нижнем регистре.

Пример:

ids_example.xml

```
1 <Button
2    android:id="@+id/start_btn"
3    android:layout_width="match_parent"
4    android:layout_height="wrap_content" />
```

Пока что это все, что вам нужно знать об id. А теперь давайте вернемся к **RelativeLayout**.

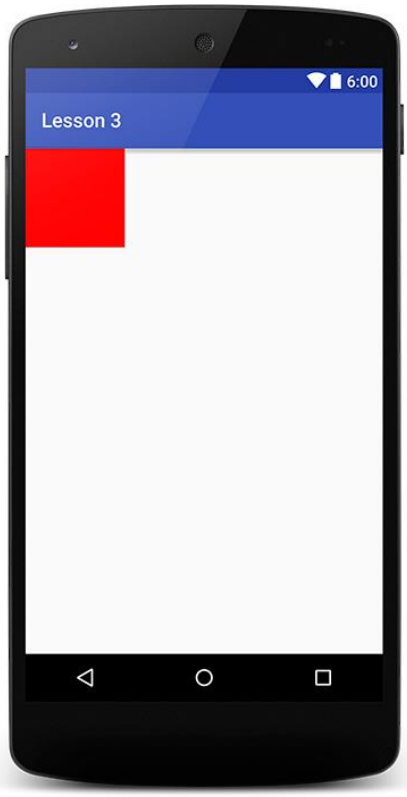
Расположение относительно RelativeLayout

Создайте новый проект так же, как мы делали это раньше.
Откройте ***activity_main.xml***, и замените код на этот:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6     <View
7        android:layout_width="100dp"
8        android:layout_height="100dp"
9        android:background="#ff0000"
10       android:textSize="20sp" />
11
12 </RelativeLayout>
```

В результате вы увидите следующее:



Как и в других ViewGroup, в **RelativeLayout** дочерние элементы по умолчанию выравниваются по левому верхнему краю.

Выравнивание по центру

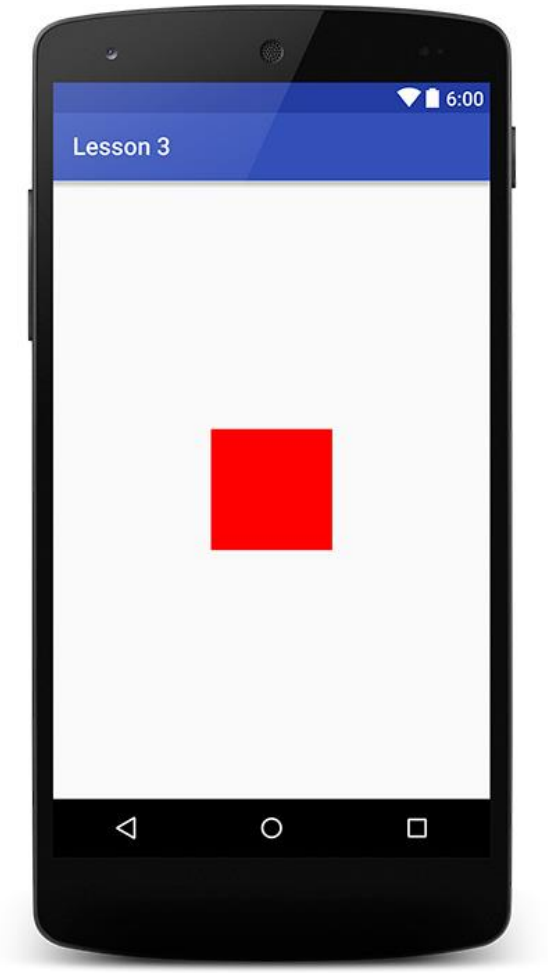
Чтобы выровнять элемент по центру относительно самого RelativeLayout, добавьте атрибут **layout_centerInParent** со значением *true:*

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <View
7          android:layout_width="100dp"
8          android:layout_height="100dp"
9          android:layout_centerInParent="true"
10         android:background="#ff0000"
11         android:textSize="20sp" />
```

```
12
13 </RelativeLayout>
```

В результате квадрат будет находиться в центре лэйаута:

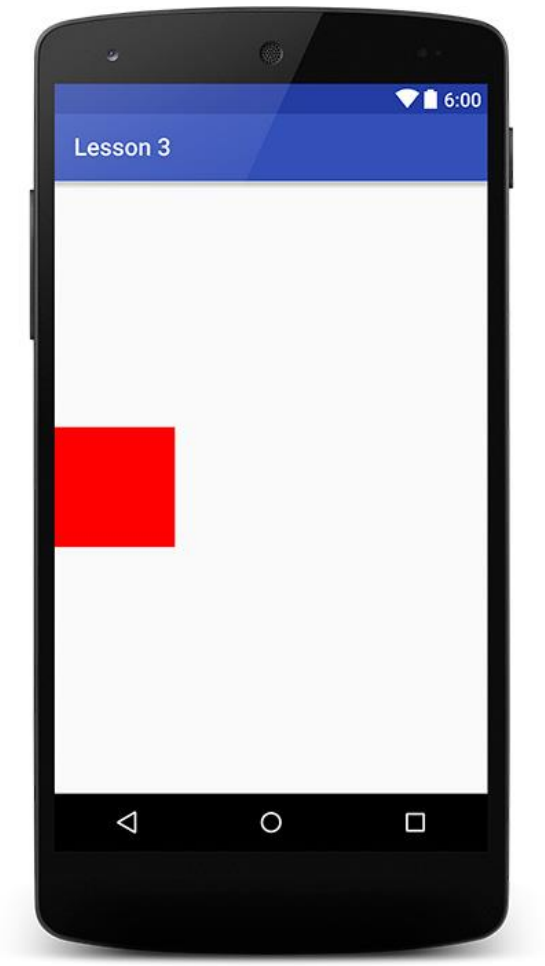


Для центрирования по вертикали нужно использовать атрибут **layout_centerVertical**:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <View
7          android:layout_width="100dp"
8          android:layout_height="100dp"
9          android:layout_centerVertical="true"
```

```
10        android:background="#ff0000"
11        android:textSize="20sp" />
12
13 </RelativeLayout>
```

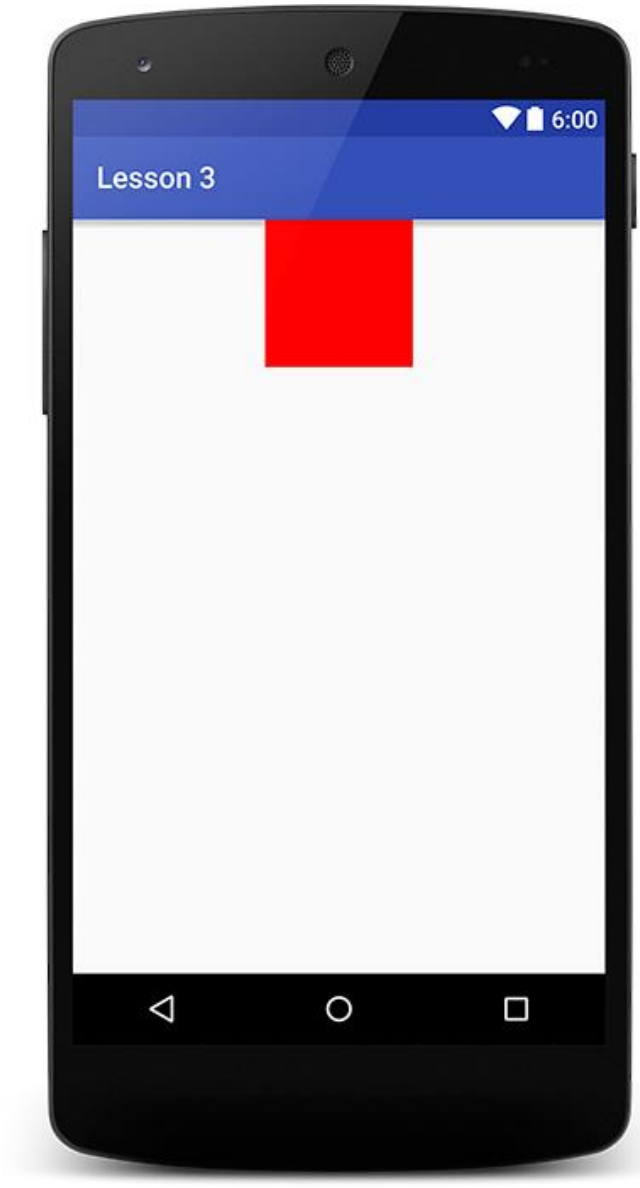


Для центрирования по вертикали, соответственно, **layout_centerHorizontal**:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6     <View
7        android:layout_width="100dp"
```

```
 8        android:layout_height="100dp"
 9        android:layout_centerHorizontal="true"
10        android:background="#ff0000"
11        android:textSize="20sp" />
12
13 </RelativeLayout>
```



Выравнивание по краям

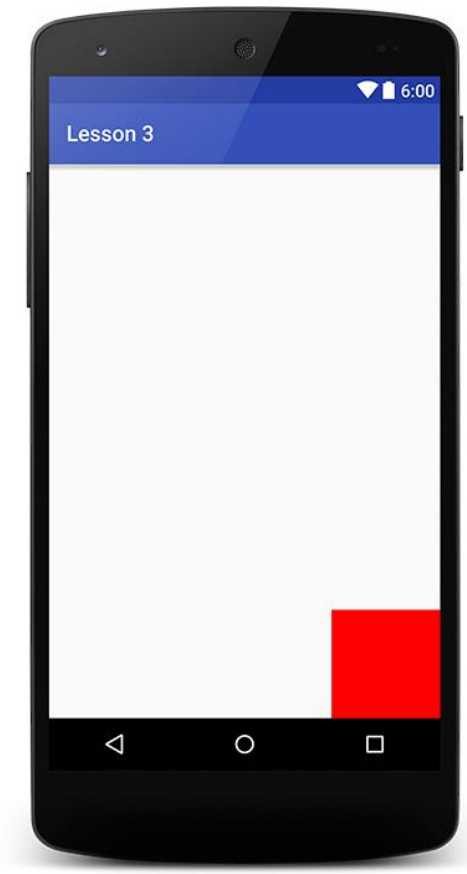
Остались
атрибуты **layout_alignParentStart**, **layout_alignParentEnd**, **layout_align ParentTop**, **layout_alignParentBottom.** Их можно комбинировать между собой (если они не противоречат друг другу).

Не будем подробно разбирать каждый из атрибутов, давайте рассмотрим следующий пример: нужно поместить квадрат в правый нижний угол экрана. За правую сторону отвечает атрибут **layout_alignParentEnd**, за нижнюю — **layout_alignParentBottom**. Соответственно, получится следующая разметка:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <View
7          android:layout_width="100dp"
8          android:layout_height="100dp"
9          android:layout_alignParentEnd="true"
10         android:layout_alignParentRight="true"
11         android:layout_alignParentBottom="true"
12         android:background="#ff0000"
13         android:textSize="20sp" />
14
15 </RelativeLayout>
```

В результате получится следующее:



Обратите внимание, что если минимальная версия SDK в проекте меньше, чем 17, то помимо **layout_alignParentEnd** нужно использовать атрибут **layout_alignParentRight**, а помимо **layout_alignParentStart** — **layout_alignParentLeft**.

Расположение относительно других элементов

Как я уже говорил, RelativeLayout — самый гибкий лэйаут из имеющихся в Android SDK. В нем мы можем выравнивать дочерние элементы не только относительно самого RelativeLayout, но и относительно других дочерних элементов. Существует два способа расположения одного дочернего элемента относительно другого:

1. После/до/под/над элементом
2. По краю элемента — левому, правому, верхнему или нижнему.

В первом случае нам потребуются атрибуты ***layout_toStartOf***, ***layout_toEndOf***, ***layout_above***, ***layout_below***, во втором — ***layout_alignStart***, ***layout_alignEnd***, ***layout_alignTop***, ***layout_alignBottom***.

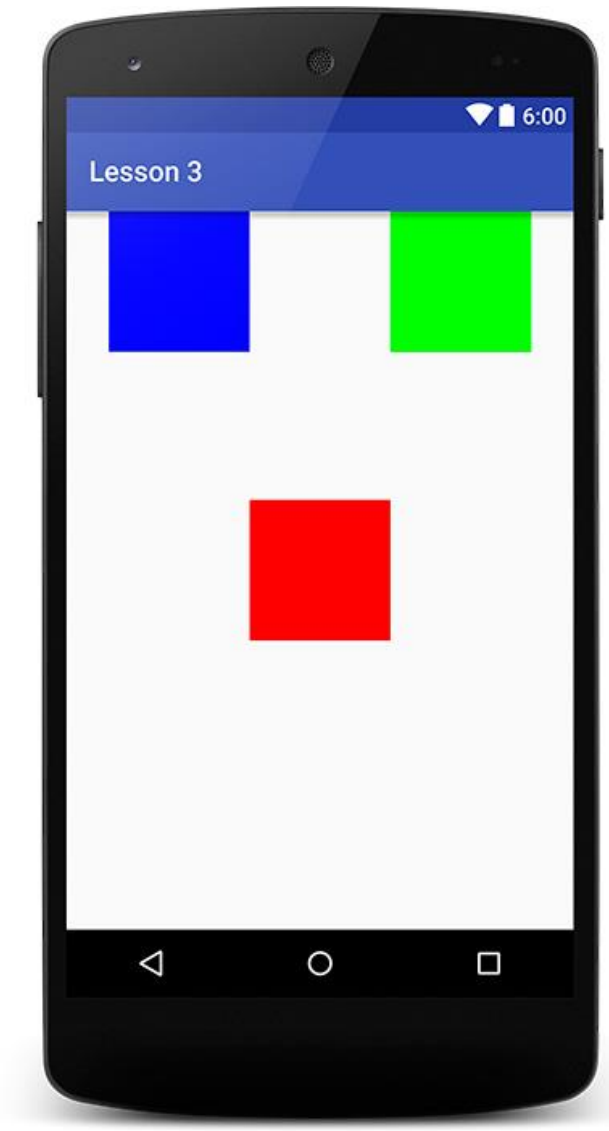
Давайте рассмотрим оба случая на практике.

Измените код на следующий:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     android:layout_width="match_parent"
4     android:layout_height="match_parent">
5
6     <View
7        android:id="@+id/red_view"
8        android:layout_width="100dp"
9        android:layout_height="100dp"
10       android:layout_centerInParent="true"
11       android:background="#ff0000"
12       android:textSize="20sp" />
13
14    <View
15       android:layout_width="100dp"
16       android:layout_height="100dp"
17       android:layout_toEndOf="@id/red_view"
18       android:layout_toRightOf="@id/red_view"
19       android:background="#00ff00"
20       android:textSize="20sp" />
21
22    <View
23       android:layout_width="100dp"
24       android:layout_height="100dp"
25       android:layout_toLeftOf="@id/red_view"
26       android:layout_toStartOf="@id/red_view"
27       android:background="#0000ff"
28       android:textSize="20sp" />
29
30 </RelativeLayout>
```

Как видите, у первого View мы добавили атрибут id, чтобы можно было ссылаться на этот View. Далее, второй View располагается справа от первого. Третий View располагается слева от первого. Посмотрим, что получилось:



Да, они располагаются, в принципе, верно, но не так, как нам хотелось бы. Как вы помните, по умолчанию элементы любого лэйаута выравниваются по верхнему левому краю. Горизонтальное расположение мы задали, а вертикальное — нет, и поэтому элементы «прижались» к верхней границе экрана. Давайте же выстроим их в один ряд, а заодно и рассмотрим второй вариант расположения, когда элементы выравниваются по краю другого элемента:

activity_main.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent">
5
6      <View
7          android:id="@+id/red_view"
```

```
 8        android:layout_width="100dp"
 9        android:layout_height="100dp"
10        android:layout_centerInParent="true"
11        android:background="#ff0000"
12        android:textSize="20sp" />
13
14    <View
15        android:layout_width="100dp"
16        android:layout_height="100dp"
17        android:layout_toEndOf="@id/red_view"
18        android:layout_toRightOf="@id/red_view"
19        android:layout_alignTop="@id/red_view"
20        android:background="#00ff00"
21        android:textSize="20sp" />
22
23    <View
24        android:layout_width="100dp"
25        android:layout_height="100dp"
26        android:layout_toLeftOf="@id/red_view"
27        android:layout_alignTop="@id/red_view"
28        android:layout_toStartOf="@id/red_view"
29        android:background="#0000ff"
30        android:textSize="20sp" />
31
32 </RelativeLayout>
```

Мы просто добавили атрибут layout_alignTop ко второму и третьему View, и они автоматически выровнялись по верхнему краю первого View.

Задание для RelativeLayout: выстройте 4 квадрата разных цветов лесенкой из левого верхнего угла в направлении правого нижнего.

Задание для LinearLayout: создайте приложение “Анкета”, содержащее вложенные линейные разметки (LinearLayout). Разместите внутри разметки следующие виджеты:

Слева в столбец расположите TextView с пунктами анкеты:

Фамилия

Имя

Отчество

Год

Школа

Город

Опыт программирования

Знание языков программирования

В центре экрана для каждого поля установите виджеты для ввода данных:

EditText для полей “Фамилия”, “Имя”, “Отчество”, “Год рождения”, "Школа".

RadioButton для полей “Город” и “Опыт программирования”.

CheckBox для поля “Знание языков программирования”.

В правой части экрана расположите календарь.