Урок 3 Верстка экранов Виды Layout

Визуальные части приложения

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](https://developer.android.com/intl/ru/reference/android/app/Activity.html" \t "_blank) и [View](https://developer.android.com/intl/ru/reference/android/view/View.html" \t "_blank).

**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](https://developer.android.com/intl/ru/reference/android/view/ViewGroup.html" \t "_blank). 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.

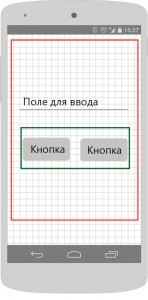
«Самых нужных» лэйаута всего 3:

* 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. Но нам для начала хватит и этих трех, остальные мы обязательно рассмотрим позже.

Посмотрите на следующий рисунок:

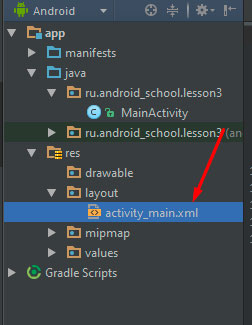


Кнопки, поле для ввода — это **View**. Зеленая рамочка вокруг кнопок — это границы **ViewGroup**, внутри которой находятся кнопки. В свою очередь, ViewGroup с кнопками и поле для ввода находятся внутри другого ViewGroup, границы которого обозначены красным.

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](https://ru.wikipedia.org/wiki/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

XHTML



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"      xmlns:tools="http://schemas.android.com/tools"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:paddingBottom="@dimen/activity\_vertical\_margin"      android:paddingLeft="@dimen/activity\_horizontal\_margin"      android:paddingRight="@dimen/activity\_horizontal\_margin"      android:paddingTop="@dimen/activity\_vertical\_margin"      tools:context="ru.android\_school.lesson2.MainActivity">        <TextView          <strong>android:layout\_width</strong>="wrap\_content"          <strong>android:layout\_height</strong>="wrap\_content"          android:text="Hello World!" />  </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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?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="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> |

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:



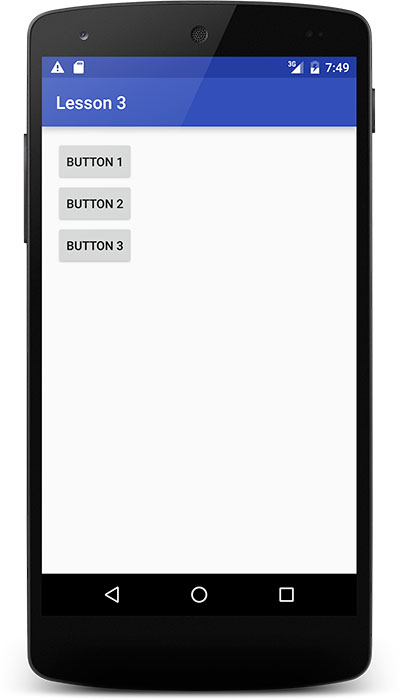
Теперь давайте изменим ориентацию с горизонтальной на вертикальную:

activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | <?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:

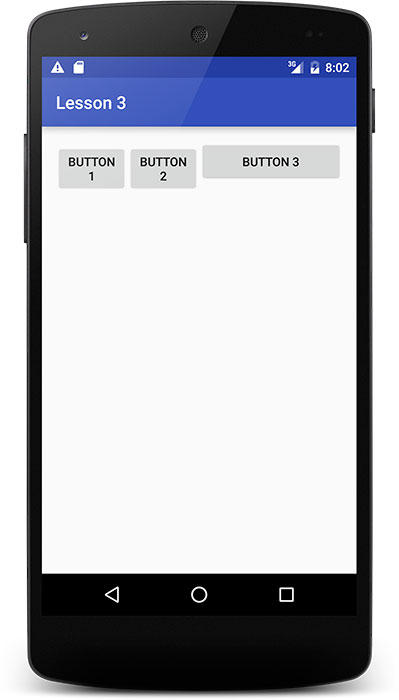
activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | <?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" />        <Button          android:layout\_width="0dp"          android:layout\_height="wrap\_content"          android:layout\_weight="2"          android:text="Button 3" />    </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.

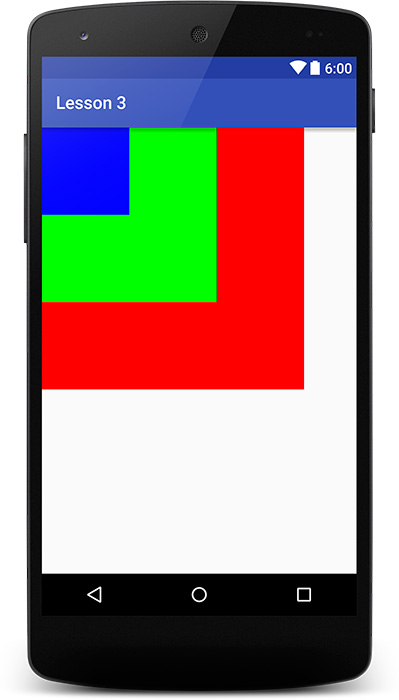
В трехмерном пространстве добавляется ось Z. Она идет «на нас». В интерфейсах Z обозначает глубину. Напрямую «глубина» задается редко, однако, например, во FrameLayout она есть. Давайте посмотрим на примере, как это работает. Измените код, чтобы он выглядел вот так:

activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | <?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:textSize="20sp" />        <View          android:layout\_width="200dp"          android:layout\_height="200dp"          android:background="#00ff00"          android:textSize="20sp" />        <View          android:layout\_width="100dp"          android:layout\_height="100dp"          android:background="#0000ff"          android:textSize="20sp" />    </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](https://developer.android.com/intl/ru/reference/android/view/Gravity.html) 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 версткой 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.

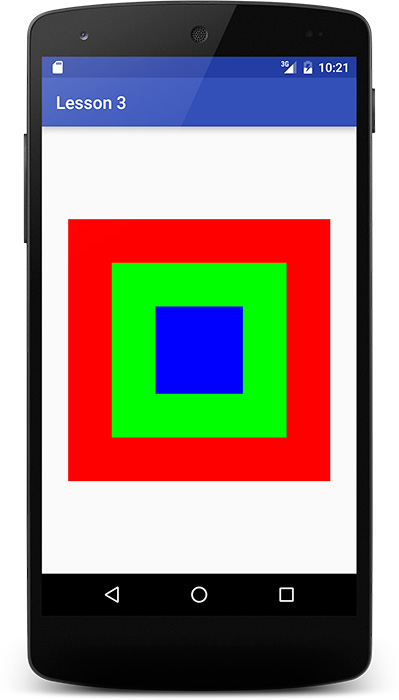
Итак, давайте же попробуем на практике поработать с гравитацией. К сожалению, сам FrameLayout, в отличие от многих других контейнеров, не поддерживает атрибут gravity, однако гравитацию можно реализовать через дочерние элементы. Добавьте атрибут **layout\_gravity** со значением ***center*** для каждого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

activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?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.

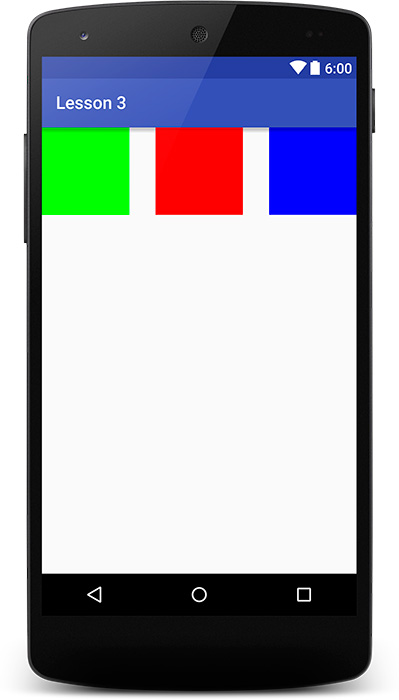
Теперь задайте всем элементам высоту в 100 dp, и давайте «раскидаем» их равномерно по углам и центру экрана. Пусть красный элемент будет расположен вверху по центру, зеленый — наверху слева, синий — наверху справа. Соответственно, у них должна быть задана гравитация start, center\_horizontal и end:

activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?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"          android:textSize="20sp" />    </FrameLayout> |

Получится следующая картина:



Отлично!

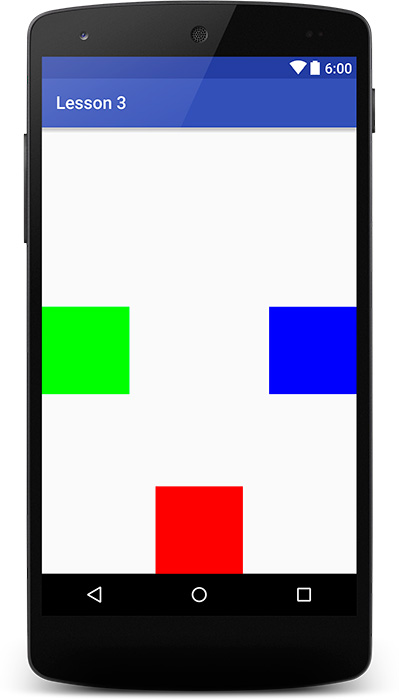
А еще атрибуты gravity можно комбинировать. К примеру, мы хотим, чтобы красный квадрат был по центру внизу, а остальные — слева и справа по центру. Тогда нам надо будет написать следующий код:

activity\_main.xml



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <?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|bottom"          android:textSize="20sp" />        <View          android:layout\_width="100dp"          android:layout\_height="100dp"          android:layout\_gravity="start|center\_vertical"          android:background="#00ff00"          android:textSize="20sp" />        <View          android:layout\_width="100dp"          android:layout\_height="100dp"          android:layout\_gravity="end|center\_vertical"          android:background="#0000ff"          android:textSize="20sp" />    </FrameLayout> |

И мы получим желаемое:



С гравитацией, пожалуй, все, но обязательно попробуйте поэкспериментировать самостоятельно с различными константами.

Давайте закончим на этом наш урок, а RelativeLayout разберем в [следующем занятии](https://android-school.ru/%d1%83%d1%80%d0%be%d0%ba-4-relativelayout/), поскольку это достаточно обширная тема, и лучше выделить ее в отдельный урок.