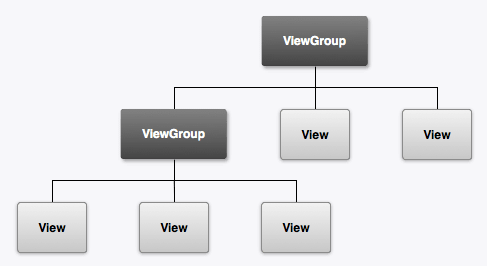
**Введение в создание интерфейса**

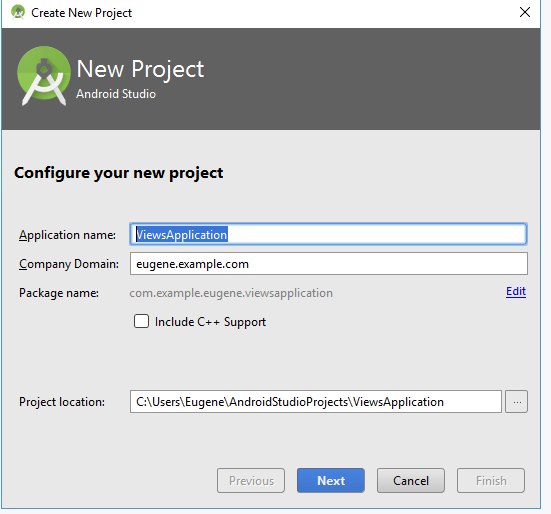
Графический интерфейс пользователя представляет собой иерархию объектов **android.view.View** и **android.view.ViewGroup**. Каждый объект ViewGroup представляет контейнер, который содержит и упорядочивает дочерние объекты **View**. В частности, к контейнерам относят такие элементы, как RelativeLayout, LinearLayout, GridLayout, ConstraintLayout и ряд других.

Простые объекты **View** представляют собой элементы управления и прочие виджеты, например, кнопки, текстовые поля и т.д., через которые пользователь взаимодействует с программой:

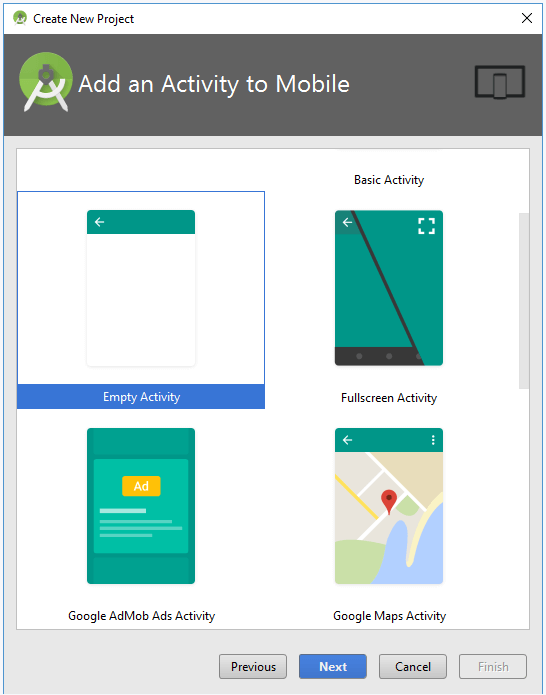


Большинство визуальных элементов, наследующихся от класса View, такие как кнопки, текстовые поля и другие, располагаются в пакете android.widget

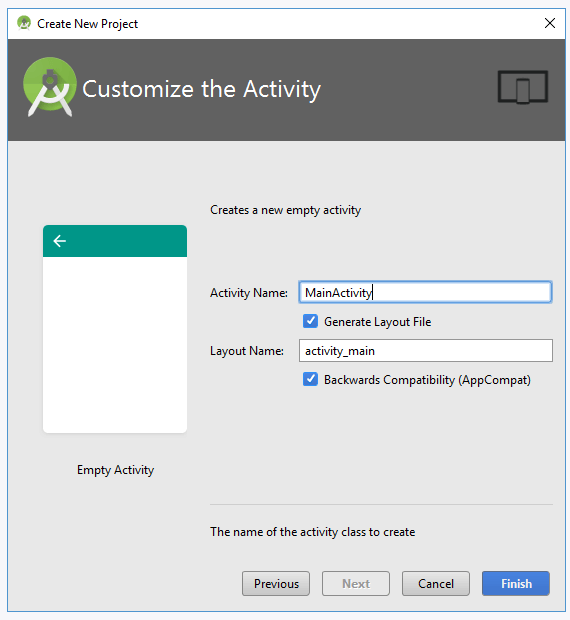
Для работы с визуальными элементами создадим новый проект. Пусть он будет называться ViewsApplication:



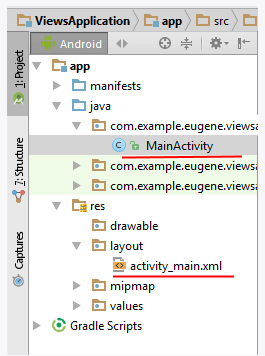
В качестве шаблона проекта выберем **Empty Activity**:

**

Имя activity при создании проекта оставим по умолчанию - MainActivity:

**

И после создания проекта два основных файла, которые будут нас интересовать при создании визуального интерфейса - это класс MainActivity и определение интерфейса для этой activity в файле activity\_main.xml.



### Стратегии определения интерфейса

Разметка определяет визуальную структуру пользовательского интерфейса. Установить разметку можно двумя способами:

* Создать элементы управления программно в коде java
* Объявить элементы интерфейса в XML
* Сочетание обоих способов - базовые элементы разметки определить в XML, а остальные добавлять во время выполнения

### Создание интерфейса в коде java

Определим в классе **MainActivity** простейший интерфейс:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package com.example.eugene.viewsapplication;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          // создание TextView          TextView textView = new TextView(this);          // установка текста в TextView          textView.setText("Hello Android!");          // установка высоты текста          textView.setTextSize(22);          // установка визуального интерфейса для activity          setContentView(textView);      }  } |

Здесь весь интерфейс представлен элементом TextView, которое предназначено для выводa текста. С помощью методов, которые, как правило, начинаются на **set**, можно установить различные свойства TextView. Например, в данном случае метод setText() устанавливает текст в поле, а setTextSize() задает высоту шрифта.

Для установки элемента в качестве интерфейса приложения в коде Activity вызывается метод **setContentView()**, в который передается визуальный элемент.

Если мы запустим приложение, то получим следующий визуальный интерфейс:

**

Хотя мы можем использовать подобный подход, в то же время более оптимально определять визуальный интерейс в файлах xml, а всю связанную логику определять в классе activity. Тем самым мы достигнем разграничения интерфейса и логики приложения, их легче будет разрабатывать и впоследствии модифицировать. И в следующей теме мы это рассмотрим.

## Определение интерфейса в файле XML. Файлы layout

В приложениях Android визуальный интерфейс нередко загружается из специальных файлов xml, которые хранят разметку. Эти файлы являются ресурсами разметки. Подобный подход напоминает создание веб-сайтов, когда интерфейс определяется в файлах html, а логика приложения - в коде javascript.

Объявление пользовательского интерфейса в файлах XML позволяет отделить интерфейс приложения от кода. Что означает, что мы можем изменять определение интерфейса без изменения кода java. Например, в приложении могут быть определены разметки в файлах XML для различных ориентаций монитора, различных размеров устройств, различных языков и т.д. Кроме того, объявление разметки в XML позволяет легче визуализировать структуру интерфейса и облегчает отладку.

Файлы разметки графического интерфейса располагаются в проекте в каталоге **res/layout**. По умолчанию при создании проекта уже есть один файл ресурсов разметки **activity\_main.xml**, который может выглядеть примерно так:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  1 | <?xml version="1.0" encoding="utf-8"?>  <android.support.constraint.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="com.example.eugene.viewsapplication.MainActivity">        <TextView          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Hello World!"          app:layout\_constraintBottom\_toBottomOf="parent"          app:layout\_constraintLeft\_toLeftOf="parent"          app:layout\_constraintRight\_toRightOf="parent"          app:layout\_constraintTop\_toTopOf="parent" />    </android.support.constraint.ConstraintLayout> |

В файле определяются все графические элементы и их атрибуты, которые составляют интерфейс. При создании разметки в XML следует соблюдать некоторые правила: каждый файл разметки должен содержать один корневой элемент, который должен представлять объект **View** или **ViewGroup**.

В данном случае корневым элементом является элемент ConstraintLayout, который содержит элемент TextView.

При компиляции каждый XML-файл разметки компилируется в ресурс View. Загрузка ресурса разметки осуществляется в методе Activity.onCreate. Чтобы установить разметку для текущего объекта activity, надо в метод setContentView в качестве параметра передать ссылку на ресурс разметки.

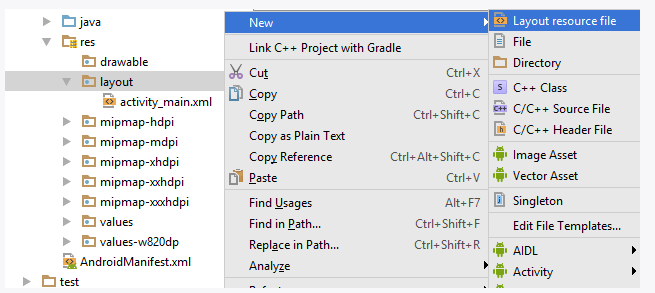
Для получения ссылки на ресурс в коде java необходимо использовать выражение R.layout.[название\_ресурса]. Название ресурса layout будет совпадать с именем файла, поэтому чтобы использовать файл **activity\_main.xml** в качестве источника визуального интерфейса, нам надо изменить код **MainActivity** следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  1  1 | package com.example.eugene.viewsapplication;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          setContentView(R.layout.activity\_main);      }  } |

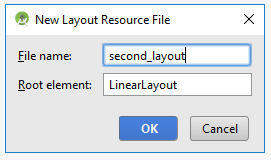
### Добавление файла layout

Но у нас может быть и несколько различных ресурсов layout. Как правило, каждый отдельный класс Activity использует свой файл layout. Либо для одного класса Activity может использоваться сразу несколько различных файлов layout.

К примеру, добавим в проект новый файл разметки интерфейса. Для этого нажмем на папку **res/layout** правой кнопкой мыши и в появившемся меню выберем пункт **New -> Layout resource file**:



После этого в специальном окошке будет предложено указать имя и корневой элемент для файла layout:

**

В качестве названия укажем **second\_layout**, а в качестве типа корневого элемента оставим **LinearLayout**.

После этого в папку *res/layout* будет добавлен новый файл **second\_layout.xml**, с которым мы можем работать точно также, как и с activity\_main.xml. В частности, откроем файл second\_layout.xml и изменим его содержимое следующим образом:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  1 | <?xml version="1.0" encoding="utf-8"?>  <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:orientation="vertical" android:layout\_width="match\_parent"      android:layout\_height="match\_parent">        <TextView android:id="@+id/header"          android:text="Second Activity"          android:textSize="26dp"          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content" />  </LinearLayout> |

Здесь определено текстовое поле TextView, которое выводит на экран простейший текст "Second Activity".

### Получение элементов в коде и их идентификаторы

Кроме текста, ширины и высоты текстовое поле устанавливает еще один важный атрибут - id. Знак + в записи android:id="@+id/header" означает, что если для элемента не определен id со значением header, то его следует определить.

Чтобы использовать этот файл в качестве основного интерфейса, перейдем к MainActivity и изменим ее код:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package com.example.eugene.viewsapplication;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            // устанавливаем в качестве интерфейса файл second\_layout.xml          setContentView(R.layout.second\_layout);            // получаем элемент textView          TextView textView = (TextView) findViewById(R.id.header);          // переустанавливаем у него текст          textView.setText("Hello Android 7!");      }  } |

С помощью метода setContentView() устанавливается разметка из файла *second\_layout.xml*.

Другой важный момент, который стоит отметить - получение визуального элемента TextView. Так как в его коде мы определили атрибут android:id, то через этот id мы можем его получить.

Для получения элементов по id класс Activity имеет метод **findViewById()**. В этот метод передается идентификатор ресурса в виде **R.id.[идентификатор\_элемента]**. Этот метод возвращает объект View - объект базового класса для всех элементов, поэтому результат метода еще необходимо привести к типу TextView.

Далее мы можем что-то сделать с этим элементом, в данном случае изменяем его текст.

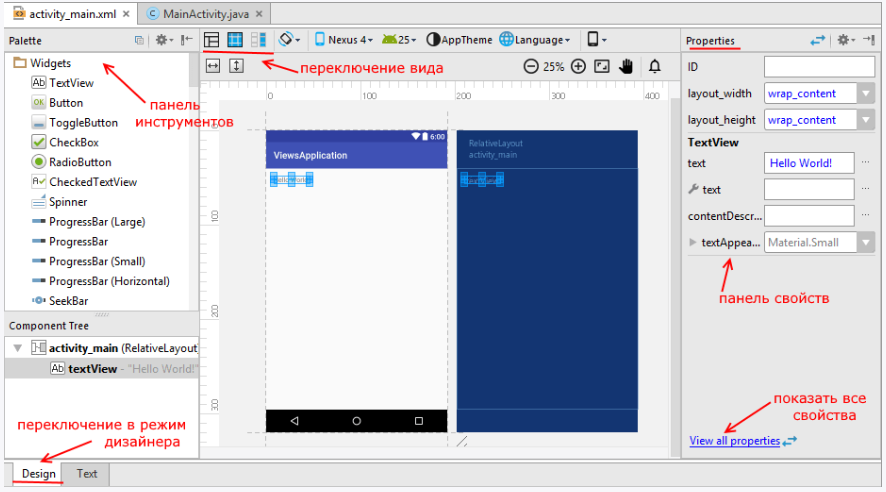
Причем что важно, получение элемента происходит после того, как в методе setContentView была установлена разметка, в которой этот визуальный элемент был определен.

И если мы запустим проект, то увидим, что TextView выводит новый текст:

**

## Графические возможности Android Studio

В прошлой теме мы рассмотрели, как в коде xml и java определять интерфейс. Но надо отметить, что Android Studio имеет довольно продвинутый инструментарий, который облегчает разработку графического интерфейса.



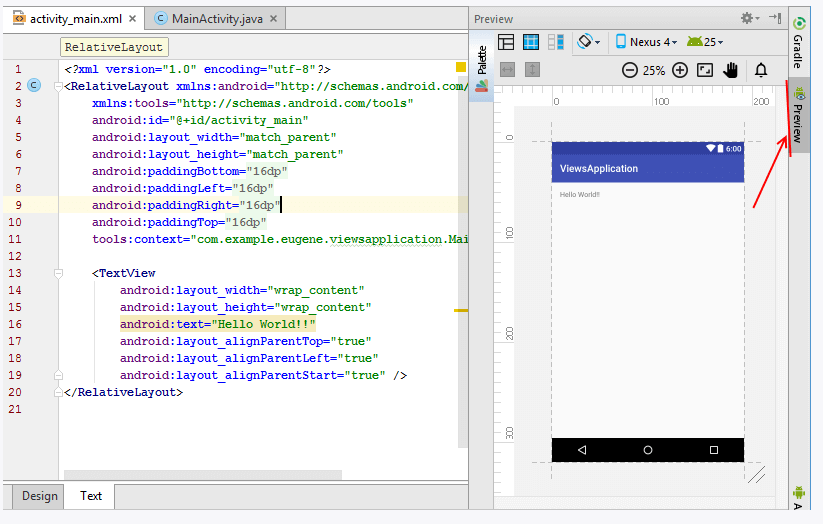
Мы можем открыть файл **activity\_main.xml** и внизу с помощью кнопки **Design** переключиться в режим дизайнера к графическому представлению интерфейсу в виде эскиза смартфона.

Слева будет находиться панель инструментов, с которой мы можем переносить нужный элемент мышкой на эскиз смартфона. И все перенесенные элементы будут автоматически добавляться в файл activity\_main.xml. С помощью мыши мы можем изменять позиционирование уже добавленных элементов, перенося их в другое место на смартфоне.

Справа будет окно Properties - панель свойств выделенного элемента. Здесь мы можем изменить значения свойств элемента. И опять же после изменения свойств изменится и содержимое файла activity\_main.xml.

То есть при любых изменениях в режиме дизайнера будет происходить синхронизация с файлом activity\_main.xml. Все равно, что мы вручную изменяли бы код непосредственно в файле activity\_main.xml.

Но даже если мы предпочитаем работать с разметкой интерфейса в текстовом виде, то даже здесь мы можем включить предварительный просмотр для файла activity\_main.xml. Для этого после переключения в текстовый режим необходимо нажать на вкладку **Preview** справа в Android Studio:



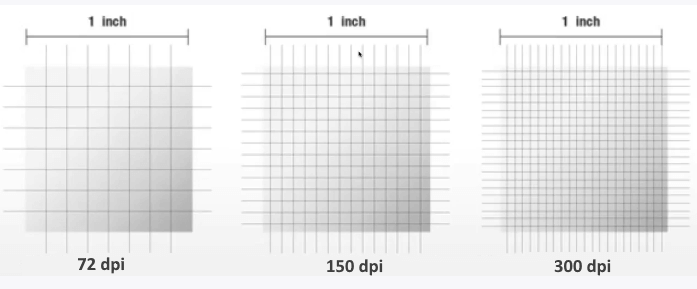
Это очень удобно, так как сразу позволяет просмотреть, как будет выглядеть приложение. А при любых изменениях область предварительного просмотра будет автоматически синхронизироваться с содержанием файла activity\_main.xml.

## Определение размеров

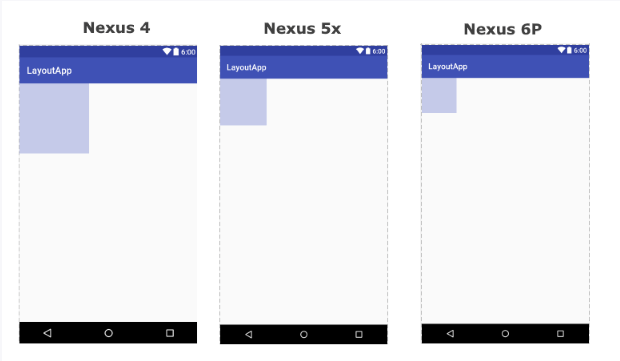
В ОС Android мы можем использовать различные типы измерений:

* px: пиксели текущего экрана. Однако эта единица измерения не рекомендуется, так как реальное представление внешнего вида может изменяться в зависимости от устройства; каждое устройство имеет определенный набор пикселей на дюйм, поэтому количество пикселей на экране может также меняться
* dp: (device-independent pixels) независимые от плотности экрана пиксели. Абстрактная единица измерения, основанная на физической плотности экрана с разрешением 160 dpi (точек на дюйм). В этом случае 1dp = 1px. Если размер экрана больше или меньше, чем 160dpi, количество пикселей, которые применяются для отрисовки 1dp соответственно увеличивается или уменьшается. Например, на экране с 240 dpi 1dp=1,5px, а на экране с 320dpi 1dp=2px. Общая формула для получения количества физических пикселей из dp: **px = dp \* (dpi / 160)**
* sp: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами.
* pt: 1/72 дюйма, базируются на физических размерах экрана
* mm: миллиметры
* in: дюймы

Предпочтительными единицами для использования являются dp. Это связано с тем, что мир мобильных устройств на Android сильно фрагментирован в плане разрешения и размеров экрана. И чем больше плотность пикселей на дюйм, тем соответственно больше пикселей нам будет доступно:

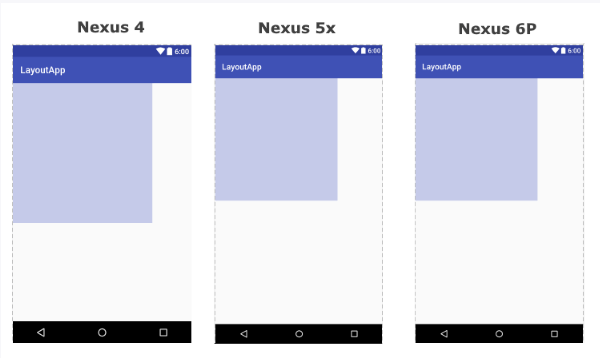


Используя же стандартные физические пиксели мы можем столкнуться с проблемой, что размеры элементов также будут сильно варьироваться в зависимости от плотности пикселей устройства. Например, возьмем 3 устройства с различными характеристиками экрана Nexus 4, Nexus 5X и Nexus 6P и выведем на экран квадрат размером 300px на 300px:



В одном случае квадрат по ширине будет занимать 40%, в другом - треть ширины, в третьем - 20%.

Теперь также возьмем квадрат со сторонами 300х300, но теперь вместо физических пикселей используем единицы dp:



Теперь же размеры квадрата на разных устройствах выглядят более консистентно.

Для упрощения работы с размерами все размеры разбиты на несколько групп:

* **ldpi (low)**: ~120dpi
* **mdpi (medium)**: ~160dpi
* **hdpi (high)**: ~240dpi (к данной группе можно отнести такое устройство как Nexus One)
* **xhdpi (extra-high)**: ~320dpi (Nexus 4)
* **xxhdpi (extra-extra-high)**: ~480dpi (Nexus 5/5X, Samsung Galaxy S5)
* **xxxhdpi (extra-extra-extra-high)**: ~640dpi (Nexus 6/6P, Samsung Galaxy S6)

## Ширина и высота элементов

Все визуальные элеметы, которые мы используем в приложении, как правило, упорядочиваются на экране с помощью контейнеров. В Android подобными контейнерами служат такие классы как RelativeLayout, LinearLayout, GridLayout, TableLayout, ConstraintLayout, FrameLayout. Все они по разному располагают элементы и управляют ими, но есть некоторые общие моменты при компоновке визуальных компонентов, которые мы сейчас рассмотрим.

Для организации элементов внутри контейнера используются параметры разметки. Для их задания в файле xml используются атрибуты, которые начинаются с префикса **layout\_**. В частности, к таким параметрам относятся атрибуты **layout\_height** и **layout\_width**, которые используются для установки размеров и могут принимать одно из следующих значений:

* точные размеры элемента, например 96 dp
* значение **wrap\_content**: элемент растягивается до тех границ, которые достаточны, чтобы вместить все его содержимое
* значение **match\_parent**: элемент заполняет всю область родительского контейнера

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  1 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/activity\_main"        android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      >        <TextView          android:text="Hello Android 7"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:background="#e0e0e0" />  </RelativeLayout> |

Контейнер самого верхнего уровня, в качестве которого в данном случае выступает RelativeLayout, для высоты и ширины имеет значение **match\_parent**, то есть он будет заполнять всю область для activity - как правило, весь экран.

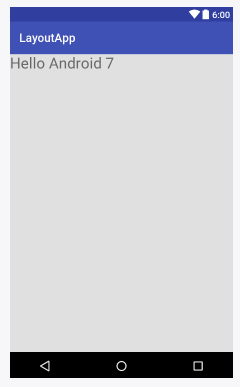
А элемент TextView растягивается до тех значений, которые достаточны для рамещения его текста.



Теперь изменим высоту и ширину на match\_parent:

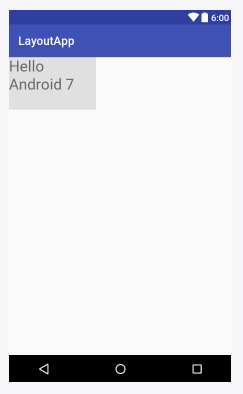
|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <TextView      android:text="Hello Android 7"      android:layout\_height="match\_parent"      android:layout\_width="match\_parent"      android:textSize="26sp"      android:background="#e0e0e0" /> |

Теперь TextView будет заполнять все пространство контейнера:



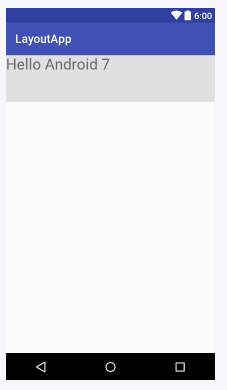
Либо мы можем установить точные значения:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <TextView      android:text="Hello Android 7"      android:layout\_height="90dp"      android:layout\_width="150dp"      android:textSize="26sp"      android:background="#e0e0e0" /> |



Также мы можем комбинировать несколько значений, например, растянуть по ширине и установить точные значения для высоты:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <TextView      android:text="Hello Android 7"      android:layout\_height="80dp"      android:layout\_width="match\_parent"      android:textSize="26sp"      android:background="#e0e0e0" /> |



Если для установки ширины и длины используется значение **wrap\_content**, то мы можем дополнительно ограничить минимальные и максимальные значения с помощью атрибутов **minWidth/maxWidth** и **minHeight/maxHeight**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <TextView android:text="Hello Android 7"            android:minWidth="200dp"          android:maxWidth="250dp"          android:minHeight="100dp"          android:maxHeight="200dp"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:background="#e0e0e0" /> |

В этом случае ширина TextView будет такой, которая достаточна для вмещения текста, но не больше значения maxWidth и не меньше значения minWidth. То же самое для установки высоты.

### Программная установка ширины и высоты

Если элемент, к примеру, тот же TextView создается в коде java, то для установки высоты и ширины можно использовать метод **setLayoutParams()**. Так, изменим код MainActivity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  27 | package com.example.eugene.layoutapp;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.ViewGroup;  import android.widget.RelativeLayout;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            RelativeLayout relativeLayout = new RelativeLayout(this);          TextView textView1 = new TextView(this);          textView1.setText("Hello Android 7");          textView1.setTextSize(26);            // устанавливаем размеры          textView1.setLayoutParams(new ViewGroup.LayoutParams                  (ViewGroup.LayoutParams.WRAP\_CONTENT, ViewGroup.LayoutParams.WRAP\_CONTENT));          // добавляем TextView в RelativeLayout          relativeLayout.addView(textView1);          setContentView(relativeLayout);      }  } |

В метод setLayoutParams() передается объект **ViewGroup.LayoutParams**. Этот объект инициализируется двумя параметрами: шириной и высотой. Для указания ширины и высоты можно использовать одну из констант **ViewGroup.LayoutParams.WRAP\_CONTENT** или **ViewGroup.LayoutParams.MATCH\_PARENT**.

Также мы можем передать точные значения или комбинировать типы значений:

|  |  |
| --- | --- |
| 1 | textView1.setLayoutParams(new ViewGroup.LayoutParams(ViewGroup.LayoutParams.MATCH\_PARENT, 200)); |

## Внутренние и внешние отступы

Параметры разметки позволяют задать отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

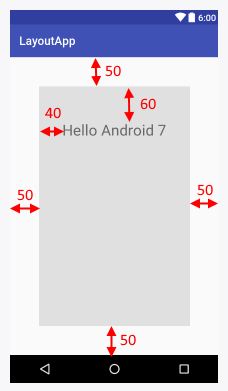
### Padding

Для установки внутренних отступов применяется атрибут android:padding. Он устанавливает отступы контента от всех четырех сторон контейнера. Можно устанавливать отступы только от одной стороны контейнера, применяя следующие атрибуты: android:paddingLeft, android:paddingRight, android:paddingTop и android:paddingBottom.

|  |  |
| --- | --- |
| 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>"      android:id="@+id/activity\_main"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:padding="50dp">        <TextView android:text="Hello Android 7"          android:layout\_height="match\_parent"          android:layout\_width="match\_parent"          android:textSize="26sp"          android:background="#e0e0e0"          android:paddingTop="60dp"          android:paddingLeft="40dp"/>    </RelativeLayout> |

У контейнера RelativeLayout установлен только один общий внутренний отступ в 50 единиц.

У TextView установлен отступ между верхней границей элемента и его внутренним содержимым (то есть текстом) в 60 единиц и отступ слева в 40 единиц:

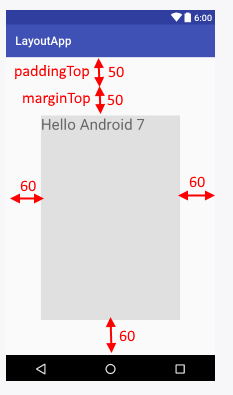


### Margin

Для установки внешних остступов используется атрибут layout\_margin. Данный атрибут имеет модификации, которые позволяют задать отступ только от одной стороны: android:layout\_marginBottom, android:layout\_marginTop, android:layout\_marginLeft и android:layout\_marginRight (отступы соответственно от нижней, верхней, левой и правой границ):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/activity\_main"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"        android:paddingTop="50dp">        <TextView android:text="Hello Android 7"          android:layout\_height="match\_parent"          android:layout\_width="match\_parent"            android:layout\_marginTop="50dp"          android:layout\_marginBottom="60dp"          android:layout\_marginLeft="60dp"          android:layout\_marginRight="60dp"            android:textSize="26sp"          android:background="#e0e0e0"/>    </RelativeLayout> |

Здесь у TextView задаются отступы от каждой стороны RelativeLayout. И стоит отметить, что у самого RelativeLayout установлен внутренний отступ сверху в 50 единиц, который также учитывается при формировании внешнего отступа сверху у TextView, поэтому общее расстояние от верхней границы TextView до верхнего края RelativeLayout будет составлять 100 единиц:



### Программная установка отступов

Для программной установки внутренних отступов у элементы вызыватся метод **setPadding(left, top, right, bottom)**, в который передаются четыре значения для каждой из сторон.

Для установки внешних отступов необходимо реализовать объект **LayoutParams** для того контейнера, который применяется. И затем вызвать у этого объекта LayoutParams метод **setMargins(left, top, right, bottom)**:

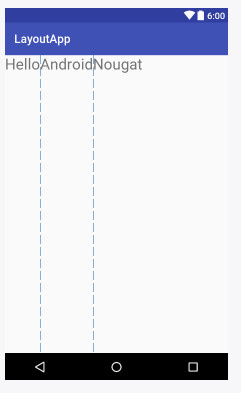
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  233 | package com.example.eugene.layoutapp;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.view.ViewGroup;  import android.widget.RelativeLayout;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            RelativeLayout relativeLayout = new RelativeLayout(this);          TextView textView1 = new TextView(this);          textView1.setBackgroundColor(0xFFBDBDBD);          textView1.setText("Hello Android 7");          textView1.setTextSize(26);            RelativeLayout.LayoutParams layoutParams = new RelativeLayout.LayoutParams                  (ViewGroup.LayoutParams.MATCH\_PARENT, 200);          // установка внешних отступов          layoutParams.setMargins(30,40,50,60);          // устанавливаем размеры          textView1.setLayoutParams(layoutParams);          // установка внутренних отступов          textView1.setPadding(30,30,30,30);          // добавляем TextView в RelativeLayout          relativeLayout.addView(textView1);          setContentView(relativeLayout);      }  } |

## LinearLayout

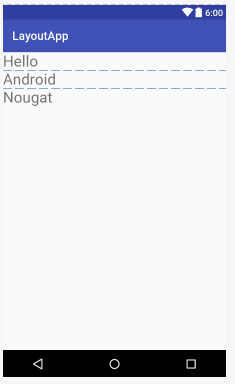
Контейнер **LinearLayout** представляет объект ViewGroup, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали. Все элемены расположены один за другим. Направление разметки указывается с помощью атрибута **android:orientation**.

Если, например, ориентация разметки вертикальная (android:orientation="vertical"), то все элементы располагаются в столбик - по одному элементу на каждой строке. Если ориентация горизонтальная (android:orientation="horizontal"), то элементы располагаются в одну строку. Например, расположим элементы в горизонтальный ряд:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | <?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" >        <TextView          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Hello"          android:textSize="26sp" />      <TextView          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Android"          android:textSize="26sp" />      <TextView          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Nougat"          android:textSize="26sp" />  </LinearLayout> |



Если бы мы указали для LinearLayout атрибут **android:orientation="vertical"**, то элементы размещались бы по вертикали:

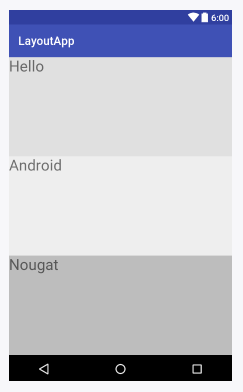


LinearLayout поддерживает такое свойство, как вес элемента, которое передается атрибутом **android:layout\_weight**. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент у нас будет иметь для свойства android:layout\_weight значение 2, а другой - значение 1, то в сумме они дадут 3, поэтому первый элемент будет занимать 2/3 оставшегося пространства, а второй - 1/3.

Если все элементы имеют значение android:layout\_weight="1", то все эти элементы будут равномерно распределены по всей площади контейнера:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <?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" >      <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:text="Hello"          android:background="#e0e0e0"          android:layout\_weight="1"          android:textSize="26sp" />      <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:background="#eeeeee"          android:text="Android"          android:layout\_weight="1"          android:textSize="26sp" />      <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:text="Nougat"          android:background="#bdbdbd"          android:layout\_weight="1"          android:textSize="26sp" />  </LinearLayout> |

В данном случае LinearLayout имеет вертикальную ориентацию, поэтому все элементы будут располагаться сверху вниз. Все три элемента имеют значение android:layout\_weight="1", поэтому сумма весов всех элементов будет равна 3, а каждый элемент получит по трети пространства в LinearLayout:



При этом так как у нас вертикальный стек, то нам надо также установить для свойства layout\_height значение **0dp**. Если бы LinearLayout имел горизонтальную ориентацию, то для свойства layout\_width надо было бы установить значение **0dp**.

Еще один атрибут **android:weightSum** позволяет указать сумму весов всех элементов. Например:

|  |  |
| --- | --- |
| 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"      android:weightSum="7">        <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:text="Hello"          android:background="#e0e0e0"          android:layout\_weight="1"          android:textSize="26sp" />      <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:background="#eeeeee"          android:text="Android"          android:layout\_weight="3"          android:textSize="26sp" />      <TextView          android:layout\_width="match\_parent"          android:layout\_height="0dp"          android:text="Nougat"          android:background="#bdbdbd"          android:layout\_weight="2"          android:textSize="26sp" />  </LinearLayout> |

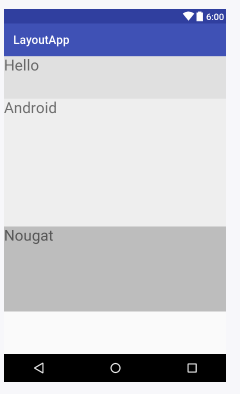
LinearLayout здесь задает сумму весов равную 7. То есть все пространство по вертикали (так как вертикальная ориентация) условно делится на семь равных частей.

Первый TextView имеет вес 1, то есть из этих семи частей занимает только одну. Второй TextView имеет вес 3, то есть занимает три части из семи. И третий имеет вес 2. Итоговая сумма составляет 6. Но так как LinearLayout задает вес 7, то одна часть будет свободна от всех элементов.

### Программное создание LinearLayout

Создание LinearLayout в коде MainActivity:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  27 | package com.example.eugene.layoutapp;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.LinearLayout;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);          LinearLayout linearLayout = new LinearLayout(this);          linearLayout.setOrientation(LinearLayout.VERTICAL);          TextView textView1 = new TextView(this);          textView1.setText("Hello");          linearLayout.addView(textView1, new LinearLayout.LayoutParams                  (LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.WRAP\_CONTENT));            TextView textView2 = new TextView(this);          textView2.setText("Android");          linearLayout.addView(textView2, new LinearLayout.LayoutParams                  (LinearLayout.LayoutParams.WRAP\_CONTENT, LinearLayout.LayoutParams.WRAP\_CONTENT));            setContentView(linearLayout);      }  } |

**

## RelativeLayout

**RelativeLayout** представляет объект ViewGroup, который располагает дочерние элементы относительно позиции других дочерних элементов разметки или относительно области самой разметки RelativeLayout. Используя относительное позиционирование, мы можем установить элемент по правому краю или в центре или иным способом, который предоставляет данный контейнер. Для установки элемента в файле xml мы можем применять следующие атрибуты:

* **android:layout\_above**: располагает элемент над элементом с указанным Id
* **android:layout\_below**: располагает элемент под элементом с указанным Id
* **android:layout\_toLeftOf**: располагается слева от элемента с указанным Id
* **android:layout\_toRightOf**: располагается справа от элемента с указанным Id
* **android:layout\_alignBottom**: выравнивает элемент по нижней границе другого элемента с указанным Id
* **android:layout\_alignLeft**: выравнивает элемент по левой границе другого элемента с указанным Id
* **android:layout\_alignRight**: выравнивает элемент по правой границе другого элемента с указанным Id
* **android:layout\_alignTop**: выравнивает элемент по верхней границе другого элемента с указанным Id
* **android:layout\_alignBaseline**: выравнивает базовую линию элемента по базовой линии другого элемента с указанным Id
* **android:layout\_alignParentBottom**: если атрибут имеет значение true, то элемент прижимается к нижней границе контейнера
* **android:layout\_alignParentRight**: если атрибут имеет значение true, то элемент прижимается к правому краю контейнера
* **android:layout\_alignParentLeft**: если атрибут имеет значение true, то элемент прижимается к левому краю контейнера
* **android:layout\_alignParentTop**: если атрибут имеет значение true, то элемент прижимается к верхней границе контейнера
* **android:layout\_centerInParent**: если атрибут имеет значение true, то элемент располагается по центру родительского контейнера
* **android:layout\_centerHorizontal**: при значении true выравнивает элемент по центру по горизонтали
* **android:layout\_centerVertical**: при значении true выравнивает элемент по центру по вертикали

Например, позиционирование относительно контейнера RelativeLayout:

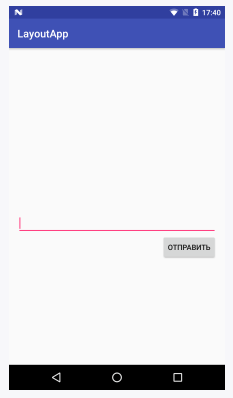
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/activity\_main"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">        <TextView android:text="Left Top"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:layout\_alignParentLeft="true"          android:layout\_alignParentTop="true" />        <TextView android:text="Right Top"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:layout\_alignParentRight="true"          android:layout\_alignParentTop="true" />        <TextView android:text="Left Bottom"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:layout\_alignParentLeft="true"          android:layout\_alignParentBottom="true" />        <TextView android:text="Right Bottom"          android:layout\_height="wrap\_content"          android:layout\_width="wrap\_content"          android:textSize="26sp"          android:layout\_alignParentRight="true"          android:layout\_alignParentBottom="true" />  </RelativeLayout> |



Для позиционирования относительно другого элемента, нам надо указать id этого элемента. Так, поместим на RelativeLayout текстовое поле и кнопку:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | <?xml version="1.0" encoding="utf-8"?>  <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:id="@+id/activity\_main"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">        <EditText          android:id="@+id/edit\_message"          android:layout\_width="match\_parent"          android:layout\_height="wrap\_content"          android:layout\_centerInParent="true"/>      <Button          android:layout\_width="wrap\_content"          android:layout\_height="wrap\_content"          android:text="Отправить"          android:layout\_alignRight="@id/edit\_message"          android:layout\_below="@id/edit\_message"          />  </RelativeLayout> |

В данном случае поле EditText располагается по центру в RelativeLayout, а кнопка помещается под EditText и выравнивается по его правой границе:



Создадим элемент RelativeLayout программно в коде MainActivity:

|  |  |
| --- | --- |
| 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  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package com.example.eugene.layoutapp;    import android.support.v7.app.AppCompatActivity;  import android.os.Bundle;  import android.widget.Button;  import android.widget.EditText;  import android.widget.RelativeLayout;    public class MainActivity extends AppCompatActivity {        @Override      protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            RelativeLayout relativeLayout = new RelativeLayout(this);            EditText editText = new EditText(this);          editText.setId(EditText.generateViewId());            Button button = new Button(this);          button.setText("Отправить");            // устанавливаем параметры положения для EditText          RelativeLayout.LayoutParams editTextParams = new RelativeLayout.LayoutParams(                  RelativeLayout.LayoutParams.MATCH\_PARENT,                  RelativeLayout.LayoutParams.WRAP\_CONTENT          );          editTextParams.addRule(RelativeLayout.CENTER\_IN\_PARENT);          relativeLayout.addView(editText, editTextParams);            // устанавливаем параметры положения для Button          RelativeLayout.LayoutParams buttonParams = new RelativeLayout.LayoutParams(                  RelativeLayout.LayoutParams.WRAP\_CONTENT,                  RelativeLayout.LayoutParams.WRAP\_CONTENT          );          buttonParams.addRule(RelativeLayout.BELOW, editText.getId());          buttonParams.addRule(RelativeLayout.ALIGN\_RIGHT, editText.getId());          relativeLayout.addView(button, buttonParams);            setContentView(relativeLayout);      }  } |

Чтобы задать положение элемента в контейнере, применяется класс **RelativeLayout.LayoutParams**. Через конструктор устанавливаются значения для для ширины и высоты. Например, у элемента EditText для ширины устанавливается значение MATCH\_PARENT, а для высоты - WRAP\_CONTENT.

С помощью метода **addRule()** мы можем добавлять дополнительные правила для позиционирования элемента. Этот метод в качестве параметра принимает числовую константу, которая представляет параметр позиционирования и которая аналогична атрибуту. Например, атрибуту android:layout\_centerInParent будет соответствовать константа CENTER\_IN\_PARENT, а атрибуту android:layout\_alignRight константа ALIGN\_RIGHT.

Стоит отметить, что в целях упрощения кода для установки id у EditText вызывается метод **generateViewId();**, который позволяет программно сгенерировать id для элемента управления.

Затем установленный id передается в качестве второго параметра в метод addRule при установке правил для кнопки:

|  |  |
| --- | --- |
| 1 | buttonParams.addRule(RelativeLayout.BELOW, editText.getId()); |

Тем самым мы указываем относительно какого элемента надо задать расположение.

## Gravity и layout\_gravity

Для управления позиционированием элемента при определении интерфейса мы можем использовать такие атрибуты как **gravity** и **layout\_gravity**.

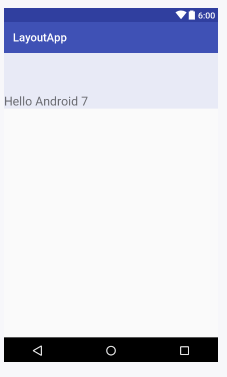
### Gravity

Атрибут **gravity** задает позиционирование содержимого внутри объекта. Он может принимать следующие значения:

* top: элементы размещаются вверху
* bottom: элементы размещаются внизу
* left: элементы размещаются в левой стороне
* right: элементы размещаются в правой стороне контейнера
* center\_vertical: выравнивает элементы по центру по вертикали
* center\_horizontal: выравнивает элементы по центру по горизонтали
* center: элементы размещаются по центру
* fill\_vertical: элемент растягивается по вертикали
* fill\_horizontal: элемент растягивается по горизонтали
* fill: элемент заполняет все пространство контейнера
* clip\_vertical: обрезает верхнюю и нижнюю границу элементов
* clip\_horizontal: обрезает правую и левую границу элементов
* start: элемент позиционируется в начале (в верхнем левом углу) контейнера
* end: элемент позиционируется в конце контейнера(в верхнем правом углу)

Например, поместим текст в самый низ в элементе TextView:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | <RelativeLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent">            <TextView              android:gravity="bottom"                android:layout\_width="match\_parent"              android:layout\_height="200px"              android:text="Hello Android 7"              android:background="#e8eaf6"/>  </RelativeLayout> |



При необходимости мы можем комбинировать значения, разделяя их вертикальной чертой:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <TextView      android:gravity="bottom|right"        android:layout\_width="match\_parent"      android:layout\_height="200px"      android:text="Hello Android 7"      android:background="#e8eaf6"/> |



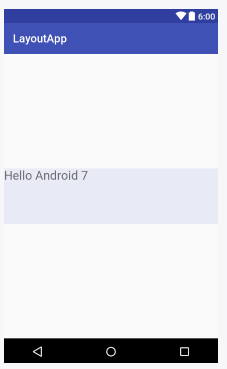
### Layout\_gravity

В отличие от gravity атрибут **layout\_gravity** устанавливает позиционирование в контейнере. Он принимает те же значения, только позиционирование идет относительно внешнего контейнера:

* top: выравнивает элемент по верхней границе контейнера
* bottom: выравнивает элемент по нижней границе контейнера
* left: выравнивает элемент по левой границе контейнера
* right: выравнивает элемент по правой границе контейнера
* center\_vertical: выравнивает элемент по центру по вертикали
* center\_horizontal: выравнивает элемент по центру по горизонтали
* center: элемент позиционируется в центре
* fill\_vertical: элемент растягивается по вертикали
* fill\_horizontal: элемент растягивается по горизонтали
* fill: элемент заполняет все пространство контейнера
* clip\_vertical: обрезает верхнюю и нижнюю границу элемента
* clip\_horizontal: обрезает правую и левую границу элемента
* start: элемент позиционируется в начале (в верхнем левом углу) контейнера
* end: элемент позиционируется в конце контейнера (в верхнем правом углу)

Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | <LinearLayout xmlns:android="<http://schemas.android.com/apk/res/android>"      android:layout\_width="match\_parent"      android:layout\_height="match\_parent"      android:orientation="horizontal">            <TextView              android:layout\_gravity="center"              android:layout\_width="match\_parent"              android:layout\_height="200px"              android:textSize="22sp"              android:text="Hello Android 7"              android:background="#e8eaf6"/>  </LinearLayout> |



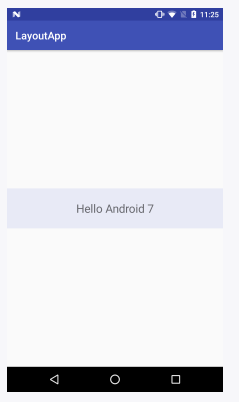
При этом надо учитывать, что layout\_gravity применяется только к классу LinearLayout или к его классам-наследникам, например, FrameLayout. В RelativeLayout этот атрибут не окажет никакого влияния.

Также внутри LinearLayout стоит учитывать ориентацию контейнера. Например, при вертикальной ориентации все элементы будут представлять вертикальный стек, идущий сверху вниз. Поэтому значения, которые относятся к позиционированию элемента по вертикали (например, top или bottom) никак не будут влиять на элемент. Также при горизонтальнйо ориентации LinearLayout не окажут никакого влияния значения, которые позиционируют элемент по горизонтали, например, left и right.

### Программная установка gravity и layout\_gravity

Чтобы установить параметр gravity у элемента надо вызвать метод **setGravity()**. Для установки программно параметра layout\_gravity надо задать поле **gravity** у объекта **LinearLayout.LayoutParams**:

|  |  |
| --- | --- |
| 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  233 | package com.example.eugene.layoutapp;    import android.os.Bundle;  import android.support.v7.app.AppCompatActivity;  import android.view.Gravity;  import android.widget.LinearLayout;  import android.widget.TextView;    public class MainActivity extends AppCompatActivity {        protected void onCreate(Bundle savedInstanceState) {          super.onCreate(savedInstanceState);            LinearLayout linearLayout = new LinearLayout(this);          TextView textView = new TextView(this);          textView.setText("Hello Android 7");          textView.setTextSize(22);          textView.setBackgroundColor(0xffe8eaf6);            // установка gravity          textView.setGravity(Gravity.CENTER);            // установка высоты и ширины          LinearLayout.LayoutParams layoutParams = new LinearLayout.LayoutParams                                              (LinearLayout.LayoutParams.MATCH\_PARENT, 200);          // установка layout\_gravity          layoutParams.gravity = Gravity.CENTER;          textView.setLayoutParams(layoutParams);            linearLayout.addView(textView);          setContentView(linearLayout);      }  } |

**