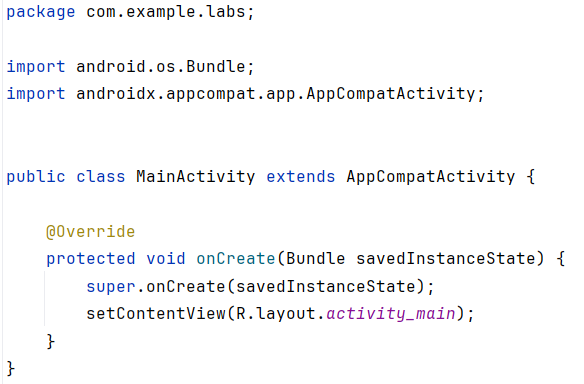
## **Создание графического интерфейса**

Выполнение приложения Android по умолчанию начинается с класса **MainActivity**, который по умолчанию открыт в Android Studio:



Каждый отдельный экран или страница в приложении описывается таким понятием как activity.

Класс MainActivity по сути представляет обычный класс java, в начале которого идет определение пакета данного класса:

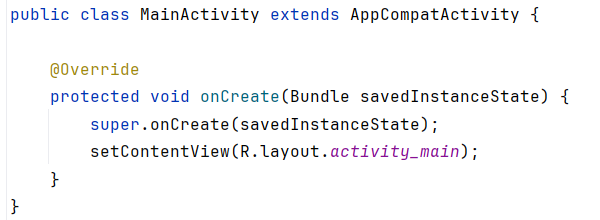
| **package** com.example.myapplication; |
| --- |

Далее идет импорт классов из других пакетов, функциональность которых используется в MainActivity:

| **import** androidx.appcompat.app.AppCompatActivity;  **import** android.os.Bundle; |
| --- |

**! Очень частой причиной ошибок является отсутствие импорта необходимых элементов. Для быстрого импорта наведитесь мышкой на объект класс которого нужно импортировать нажмите Alt + Enter и выберете Import Class !**

Затем идет определение класса:



По умолчанию MainActivity наследуется от класса AppCompatActivity, который выше подключен с помощью директивы импорта. Класс **AppCompatActivity** по сути представляет отдельный экран (страницу) приложения или его визуальный интерфейс. И MainActivity наследует весь этот функционал.

По умолчанию MainActivity содержит только один метод onCreate(), в котором фактически и создается весь интерфейс приложения:

В метод **setContentView()** передается ресурс разметки графического интерфейса:

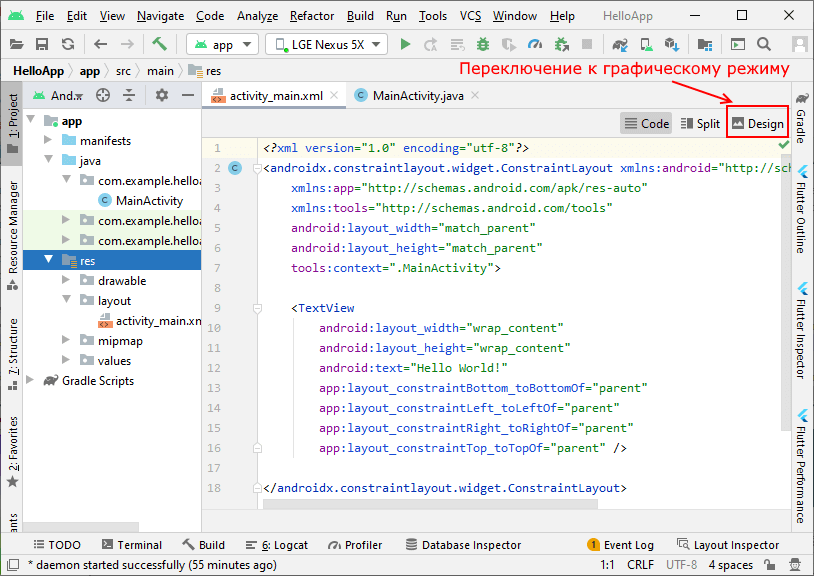
| setContentView(R.layout.***activity\_main***); |
| --- |

Именно здесь и решается, какой именно визуальный интерфейс будет иметь MainActivity. Но что в данном случае представляет ресурс **R.layout.activity\_main**? Это файл **activity\_main.xml** из папки res/layout (в принципе можно заметить, что название ресурса соответствует названию файла), который также по умолчанию открыт в Android Studio:

### Файл activity\_main.xml

Android Studio позволяет работать с визуальным интерфейсом как в режиме кода, так и в графическом режиме. Так, по умолчанию файл открыт в графическом режиме, и мы наглядно можем увидеть, как у нас примерно будет выглядеть экран приложения. И даже набросать с панели инструментов какие-нибудь элементы управления, например, кнопки или текстовые поля.

Но также мы можем работать с файлом в режиме кода, поскольку **activity\_main.xml** - это обычный текстовый файл с разметкой xml. Для переключения к коду нажмем на кнопку **Code** над графическим представлением. (Дополнительно с помощью кнопки **Split** можно переключиться на комбинированное представление код + графический дизайнер)



Здесь мы увидим, что на уровне кода файл **activity\_main.xml** содержит следующую разметку:

| *<?***xml version="1.0" encoding="utf-8"***?>* <**androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context=".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"** />  </**androidx.constraintlayout.widget.ConstraintLayout**> |
| --- |

Весь интерфейс представлен элементом-контейнером **androidx.constraintlayout.widget.ConstraintLayout**:

| <**androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:app="http://schemas.android.com/apk/res-auto"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context=".MainActivity"**>  //////////////////////////////////////////// </**androidx.constraintlayout.widget.ConstraintLayout**> |
| --- |

ConstraintLayout позволяет расположить вложенные элементы в определенных местах экрана. Вначале элемента ConstraintLayout идет определение пространств имен XML:

| **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"** |
| --- |

Каждое пространство имен задается следующим образом: xmlns:префикс="название\_ресурса". Например, в

| **xmlns:android="http://schemas.android.com/apk/res/android"** |
| --- |

Название ресурса (или URI - Uniform Resource Indicator) - "http://schemas.android.com/apk/res/android". И этот ресурс сопоставляется с префиксом android (xmlns:android).

Зачем эти пространства имен нужны? Каждый ресурс или URI определяет некоторую функциональность, которая используется в приложении, например, предоставляют теги и атрибуты, которые необходимые для построения приложения.

* xmlns:android="http://schemas.android.com/apk/res/android": содержит основные атрибуты, которые предоставляются платформой Android, применяются в элементах управления и определяют их визуальные свойства (например, размер, позиционирование)
* xmlns:app="http://schemas.android.com/apk/res-auto": содержит атрибуты, которые определены в рамках приложения
* xmlns:tools="http://schemas.android.com/tools": применяется для работы с режиме дизайнера в Android Studio

И чтобы упростить работу с этими ресурсами, применяются префиксы. Например, дальше мы видим:

| **android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context=".MainActivity"**> |
| --- |

**android:layout\_width** определяет ширину контейнера. Этот атрибут (layout\_width) расположен в ресурсе "http://schemas.android.com/apk/res/android". И поскольку этот ресурс сопоставляется с префиксом android, то для обращения к атрибуту перед ним через двоеточие указывается префикс данного ресурса.

Значением атрибута **android:layout\_weight**является **"match\_parent"**. Это значит, что элемент (ConstraintLayout) будет растягиваться по всей ширине контейнера (экрана устройства).

Атрибут **android:layout\_height="match\_parent"** определяет высоту контейнера и также определен в "http://schemas.android.com/apk/res/android". Значение "match\_parent" указывает, что ConstraintLayout будет растягивается по всей длине контейнера (экрана устройства).

Атрибут **tools:context** определяет, какой класс activity (экрана приложения) связан с текущим определением интерфейса. В данном случае это класс MainActivity. Это позволяет использовать в Android Studio различные возможности в режиме дизайнера, которые зависят от класса activity.

#### View

Базовый элемент на основе которого строятся большинство остальных элементов



Базовый элемент на основе которого строятся большинство остальных элементов

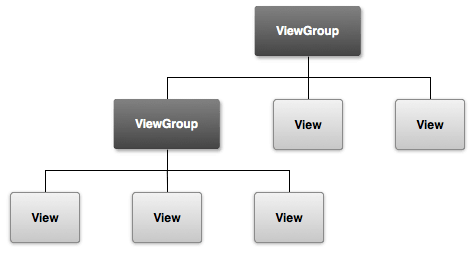
* **android:layout\_width** устанавливает ширину виджета. Значение wrap\_content задает для виджета величину, достаточную для отображения в контейнере.
* **android:layout\_height** устанавливает высоту виджета. Значение wrap\_content аналогично установке ширины задает для виджета высоту, достаточную для отображения в контейнере

Остальные свойства рассмотрим позднее

# **Основы создания 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

При определении визуального у нас есть три стратегии:

* 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

Сначала рассмотрим первую стратегию - определение интерейса в коде Java.

## **Создание интерфейса в коде java**

Определим в классе **MainActivity** простейший интерфейс:

Результат при запуске 



При создании виджетов в коде Java применяется их конструктор, в который передается контекст данного виджета, а точнее объект **android.content.Context**, в качестве которого выступает текущий класс MainActivity.



Здесь весь интерфейс представлен элементом View. С помощью методов, которые, как правило, начинаются на **set**, можно установить различные свойства TextView. Например, в данном случае метод setBackgroundColor() устанавливает цвет фона.

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

## **LinearLayout**

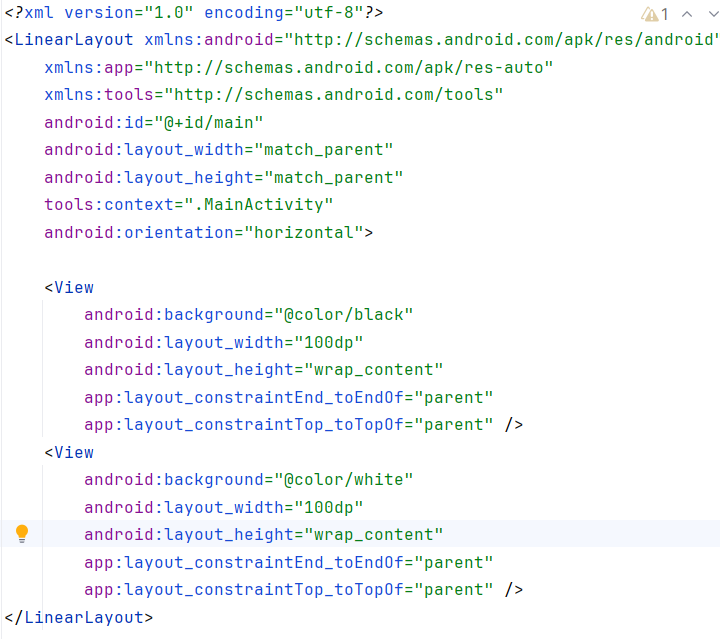
Для изучения работы контейнеров приведем пример контейнера **LinearLayout.**  **LinearLayout** представляет простейший контейнер - объект ViewGroup, который упорядочивает все дочерние элементы в одном направлении: по горизонтали или по вертикали. Все элементы расположены один за другим. Направление разметки указывается с помощью атрибута **android:orientation**.

Если, 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:





Изменим значения размеров элементов и атрибут LinearLayout **android:orientation="vertical"**, элементы размещаются по вертикали:

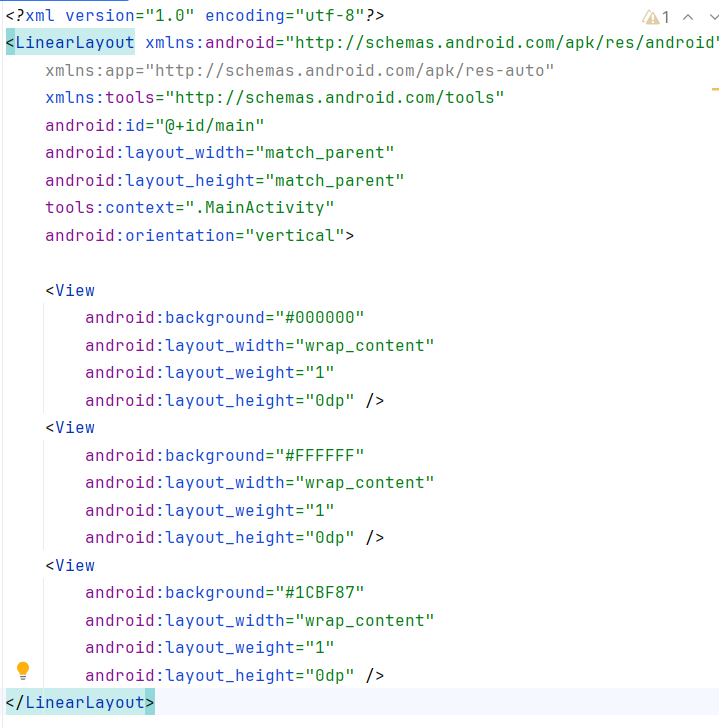




**Установка веса компонентов**

Контейнер **LinearLayout** позволяет назначить вложенным компонентам высоту в соответствии с их весом. Для указания веса применяется атрибут **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:

****

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:



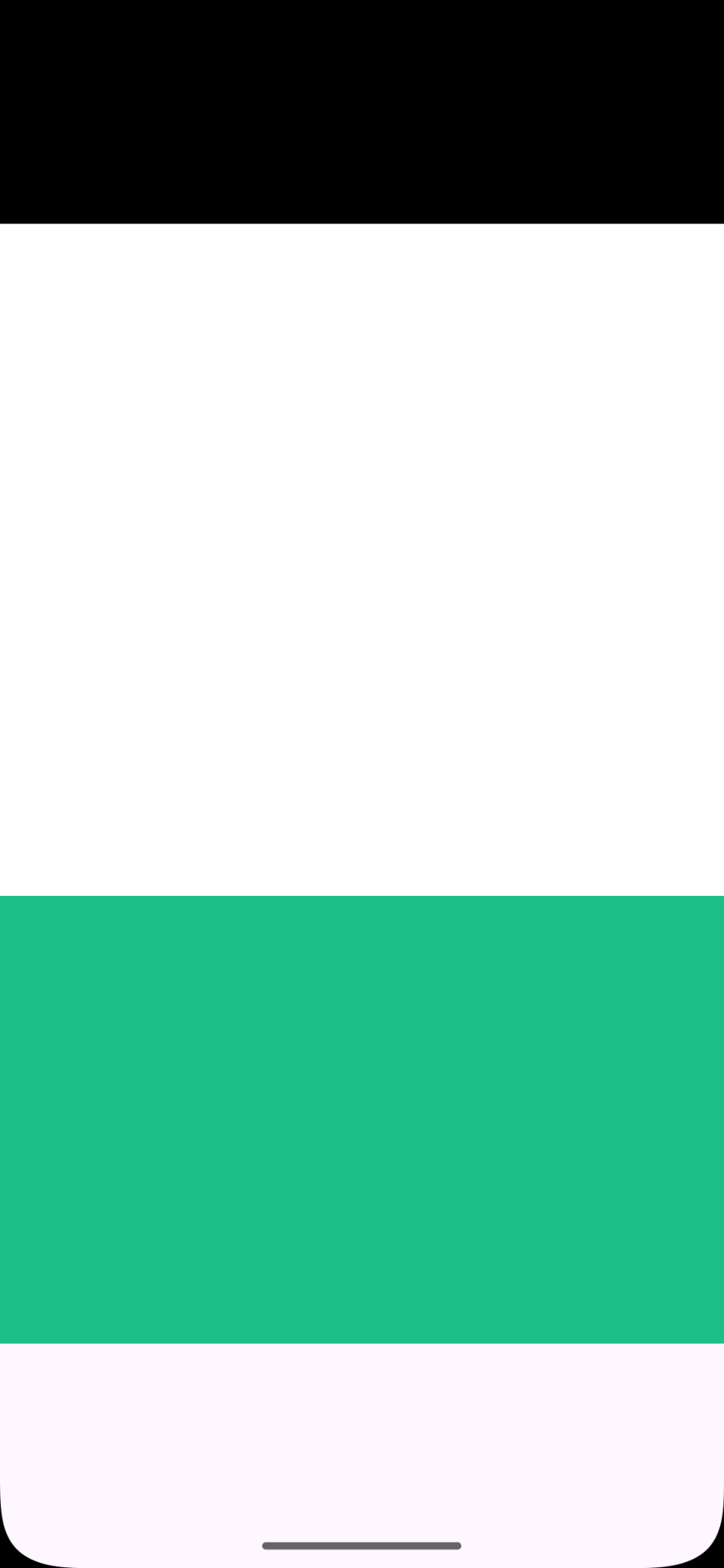
При этом так как у нас вертикальный стек, то нам надо также установить для свойства layout\_height значение **0dp**. Если бы LinearLayout имел горизонтальную ориентацию, то для свойства layout\_width надо было бы установить значение **0dp**.

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



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

Первый View имеет вес 1, то есть из этих семи частей занимает только одну. Второй View 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.

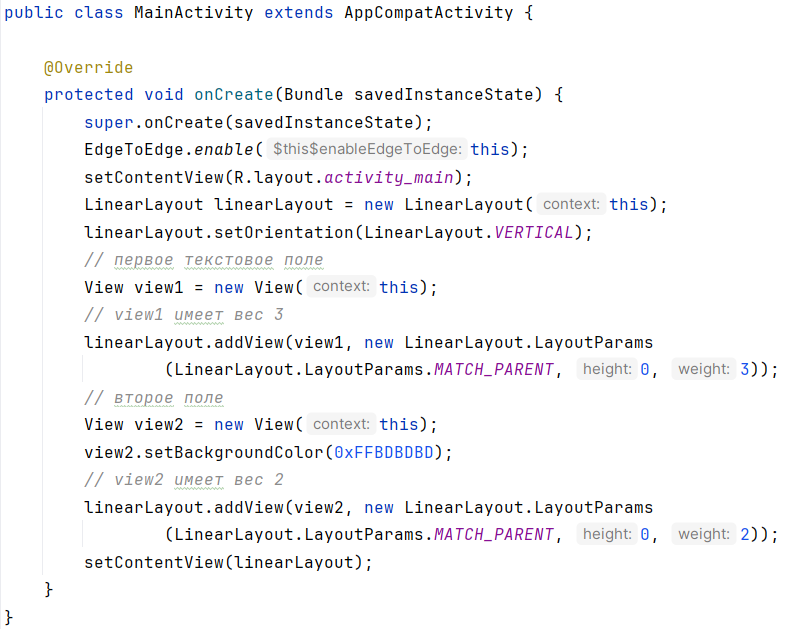


### Программное создание LinearLayout

Создание LinearLayout в коде java:



Дополнительная версия конструктора **LinearLayout.LayoutParams()** в качестве третьего параметра позволяет указать вес элемента:





### Layout\_gravity

Атрибут **layout\_gravity** позволяет устанавливать позиционирование относительно LinearLayout. Он принимает следуюшие значения:

* 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:



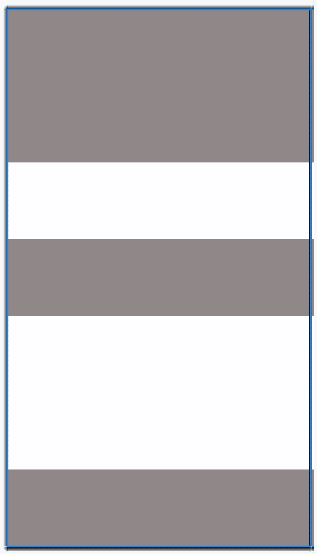
В данном случае первый элемент TextView будет позиционироваться по левой стороне контейнера (android:layout\_gravity="left"), второй TextView по центру (android:layout\_gravity="center"), третий - по правой стороне (android:layout\_gravity="right") и четвертый - по центру (android:layout\_gravity="center")



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 горизонтальной ориентации LinearLayout не окажут никакого влияния значения, которые позиционируют элемент по горизонтали, например, left и right.

**Задание для самостоятельной работы:**

1. Создайте макет интерфейса соответствующий следующему макету:



1. Создайте макет интерфейса соответствующий следующему макету:

