Практическая работа 4

Методические указания

Основы создания интерфейса(продолжение)

LinearLayout

Контейнер 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.

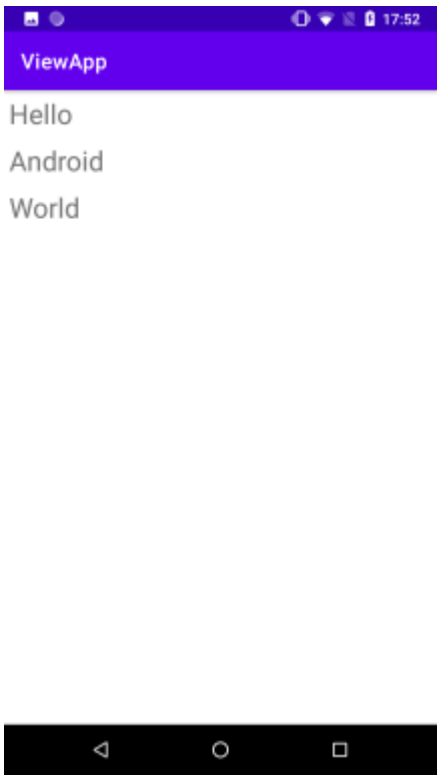
Если, например, ориентация разметки вертикальная (android:orientation="vertical"), то все элементы располагаются в столбик - по одному элементу на каждой строке. Если ориентация горизонтальная (android:orientation="horizontal"), то элементы располагаются в одну строку. Например, расположим элементы в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

```
<?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:layout_margin="5dp"
        android:text="Hello"
        android:textSize="26sp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
```

```
        android:text="Android"
        android:textSize="26sp" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="5dp"
        android:text="World"
        android:textSize="26sp" />
</LinearLayout>
```

LinearLayout в Android Studio

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



Вертикальный LinearLayout в Android Studio

Вес элемента

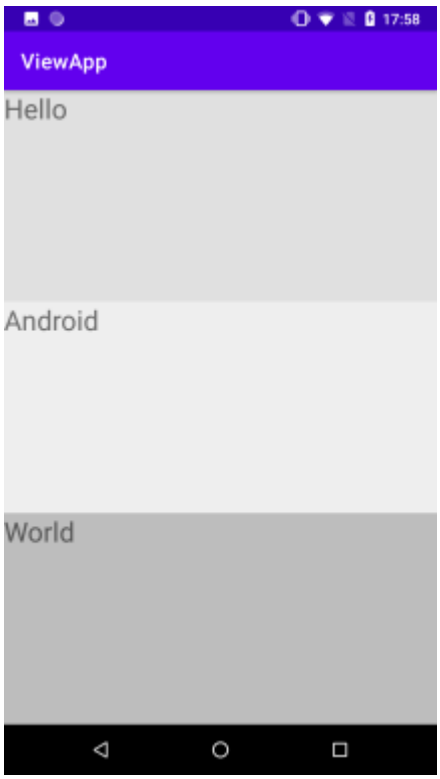
LinearLayout поддерживает такое свойство, как вес элемента, которое передается атрибутом android:layout_weight. Это свойство принимает значение, указывающее, какую часть оставшегося свободного места контейнера по отношению к другим объектам займет данный элемент. Например, если один элемент у нас будет 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:

```
<?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="World"
        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:



layout_weight в Android

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**.

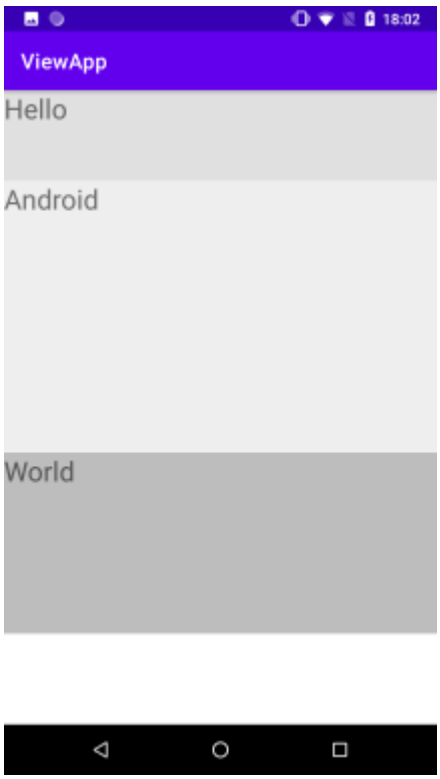
Еще один атрибут **android:weightSum** позволяет указать сумму весов всех элементов. Например:

```
<?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="World"
        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.



weightSum в LinearLayout

Программное создание LinearLayout

Создание LinearLayout в коде java:

```
package com.example.viewapp;
 import androidx.appcompat.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);
       //setContentView(R.layout.activity_main);
```

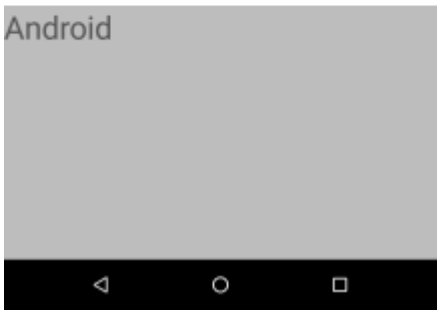
```
        LinearLayout linearLayout = new LinearLayout(this);
        // горизонтальная ориентация
        linearLayout.setOrientation(LinearLayout.HORIZONTAL);

        TextView textView = new TextView(this);
        textView.setText("Hello");
        textView.setTextSize(30);
        // создаем параметры позиционирования для элемента
        LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
        // устанавливаем отступы
        layoutParams.setMargins(100, 100, 0, 0);
        textView.setLayoutParams(layoutParams);
        // добавляем элемент в LinearLayout
        linearLayout.addView(textView);

        setContentView(linearLayout);
    }
}
```

Программное создание LinearLayout в Android и Java

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

```
package com.example.viewapp;

import androidx.appcompat.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);

        // первое текстовое поле
        TextView textView1 = new TextView(this);
        textView1.setText("Hello");
        textView1.setTextSize(30);
        // textView1 имеет вес 3
        linearLayout.addView(textView1, new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.MATCH_PARENT, 0, 3));

        // второе текстовое поле
        TextView textView2 = new TextView(this);
        textView2.setText("Android");
        textView2.setBackgroundColor(0xFFBDBDBD);
        textView2.setTextSize(30);
        // textView2 имеет вес 2
        linearLayout.addView(textView2, new LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.MATCH_PARENT, 0, 2));

        setContentView(linearLayout);
    }
}
```



Программное создание LinearLayout с layout_weight в Android и Java

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

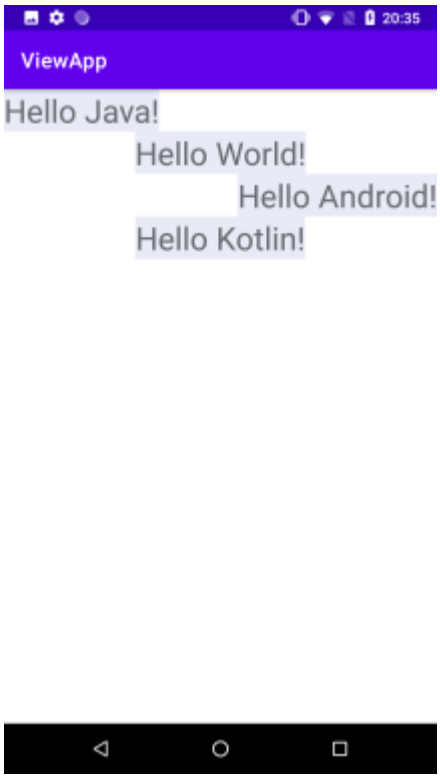
- center_horizontal: выравнивает элемент по центру по горизонтали
- center: элемент позиционируется в центре
- fill_vertical: элемент растягивается по вертикали
- fill_horizontal: элемент растягивается по горизонтали
- fill: элемент заполняет все пространство контейнера
- clip_vertical: обрезает верхнюю 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:

```
<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_gravity="left"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="Hello Java!"
        android:background="#e8eaf6"/>
    <TextView
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="Hello World!"
        android:background="#e8eaf6"/>
    <TextView
        android:layout_gravity="right"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="Hello Android!"
        android:background="#e8eaf6"/>
    <TextView
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="Hello Kotlin!"
        android:background="#e8eaf6"/>
</LinearLayout>
```

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")



layout_gravity в LinearLayout в Android

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.

Для установки программно параметра **layout_gravity** надо задать поле **gravity** у объекта **LinearLayout.LayoutParams**:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
```

```
import android.view.Gravity;
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);
        LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams
                (LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
// установка layout_gravity
        layoutParams.gravity = Gravity.CENTER;
        // первое текстовое поле
        TextView textView1 = new TextView(this);
        textView1.setText("Hello");
        textView1.setTextSize(30);
        linearLayout.addView(textView1, layoutParams);
        setContentView(linearLayout);
    }
}
```

В качестве значения передается одна из констант класса Gravity, которые аналогичны значениям атрибута.

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 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_toStartOf*: располагает начало текущего элемента, где начинается элемент с указанным Id
- **android:layout_toEndOf**: располагает начало текущего элемента, где завершается элемент 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_alignStart**: выравнивает элемент по линии, у которой начинается другой элемент с указанным Id

- **android:layout_alignEnd**: выравнивает элемент по линии, у которой завершается другой элемент 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_alignParentStart**: если атрибут имеет значение true, то элемент прижимается к начальному краю контейнера (при левосторонней ориентации текста - левый край)

- **android:layout_alignParentEnd**: если атрибут имеет значение true, то элемент прижимается к конечному краю контейнера (при левосторонней ориентации текста - правый край)

- **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:

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

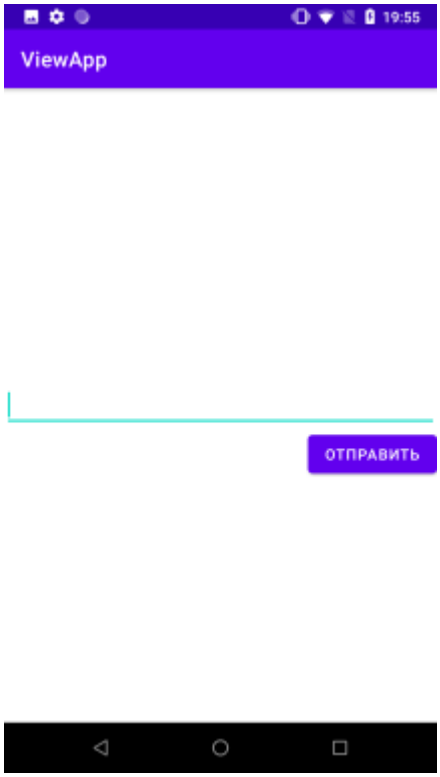
RelativeLayout в Android Studio

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

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



Позиционирование RelativeLayout в Android

Программное создание RelativeLayout

Создадим элемент RelativeLayout программно в коде Java:

```
package com.example.viewapp;
 import androidx.appcompat.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
       );
       // выравнивание по центру родительского контейнера
       editTextParams.addRule(RelativeLayout.CENTER_IN_PARENT);
       // добавляем в RelativeLayout
       relativeLayout.addView(editText, editTextParams);

       // устанавливаем параметры положения для Button
       RelativeLayout.LayoutParams buttonParams = new
RelativeLayout.LayoutParams(
```

```
            RelativeLayout.LayoutParams.WRAP_CONTENT,
            RelativeLayout.LayoutParams.WRAP_CONTENT
        );
        // выравнивание справа и снизу от поля EditText
        buttonParams.addRule(RelativeLayout.BELOW, editText.getId());
        buttonParams.addRule(RelativeLayout.ALIGN_RIGHT, editText.getId());
        // добавляем в RelativeLayout
        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:

```
buttonParams.addRule(RelativeLayout.BELOW, editText.getId());
```

Тем самым мы указываем относительно какого элемента надо задать расположение.

TableLayout

Контейнер TableLayout структурирует элементы управления в виде таблицы по столбцам и строкам. Определим в файле activity_main.xml элемент TableLayout, который будет включать две строки и два столбца:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
        <TextView
            android:layout_weight="0.5"
            android:text="Логин"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <EditText
            android:layout_weight="1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"  />
    </TableRow>

    <TableRow>
        <TextView
            android:layout_weight="0.5"
            android:text="Email"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
```

```
        <EditText
            android:layout_weight="1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>
</TableLayout>
```

Разметка TableLayout в Android

Используя элемент TableRow, мы создаем отдельную строку. Как разметка узнает сколько столбцов надо создать? Android находит строку с максимальным количеством виджетов одного уровня, и это количество будет означать количество столбцов. Например, в данном случае у нас определены две строки и в каждой по два элемента. Если бы в какой-нибудь из них было бы три виджета, то соответственно столбцов было бы также три, даже если в другой строке осталось бы два виджета.

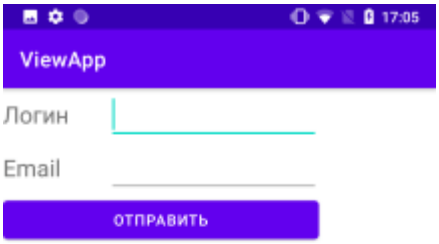
Причем элемент TableRow наследуется от класса LinearLayout, поэтому мы можем к нему применять тот же функционал, что и к LinearLayout. В

частности, для определения пространства для элементов в строке используется атрибут android:layout_weight.

Если какой-то элемент должен быть растянут на ряд столбцов, то мы можем растянуть его с помощью атрибута **layout_span**, который указывает на какое количество столбцов надо растянуть элемент:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TableRow>
        <TextView
            android:textSize="22sp"
            android:text="Логин"
            android:layout_width="100dp"
            android:layout_height="wrap_content" />

        <EditText
            android:textSize="22sp"
            android:layout_width="200dp"
            android:layout_height="wrap_content"  />
    </TableRow>

    <TableRow>
        <TextView
            android:textSize="22sp"
            android:text="Email"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />

        <EditText
```

```
        android:textSize="22sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    </TableRow>
    <TableRow>
        <Button
        android:text="Отправить"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_span="2"/>
    </TableRow>
</TableLayout>
```



Растягиваем элемент на несколько столбцов в TableLayout в Android

Также можно растянуть элемент на всю строку, установив у него атрибут android:layout_weight="1":

```
<TableRow>
```

```
    <Button
        android:text="Отправить"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" />
</TableRow>
```

Программное создание TableLayout

Создадим TableLayout программным образом, переложив на код java самый первый пример из данного материала:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.EditText;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TableLayout tableLayout = new TableLayout( this);

        // первая строка
```

```
        TableRow tableRow1 = new TableRow(this);

        TextView textView1 = new TextView(this);
        textView1.setText("Логин");
        tableRow1.addView(textView1, new TableRow.LayoutParams(
                TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 0.5f));

        EditText editText1 = new EditText(this);
        tableRow1.addView(editText1, new TableRow.LayoutParams(
                TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 1.0f));

        // вторая строка
        TableRow tableRow2 = new TableRow(this);

        TextView textView2 = new TextView(this);
        textView2.setText("Email");
        tableRow2.addView(textView2, new TableRow.LayoutParams(
                TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 0.5f));

        EditText editText2 = new EditText(this);
        tableRow2.addView(editText2, new TableRow.LayoutParams(
                TableRow.LayoutParams.WRAP_CONTENT,
TableRow.LayoutParams.WRAP_CONTENT, 1.f));

        tableLayout.addView(tableRow1);
        tableLayout.addView(tableRow2);
        setContentView(tableLayout);
```

```
    }
}
```

FrameLayout

Контейнер FrameLayout предназначен для вывода на экран одного помещенного в него визуального элемента. Если же мы поместим несколько элементов, то они будут накладываться друг на друга. Тем не менее также можно располагать в FrameLayout несколько элементов.

Допустим, вложим в FrameLayout два элемента TextView:

```
<?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">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="26sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android!"
        android:textSize="26sp"
        android:layout_marginTop="50dp"/>

</FrameLayout>
```

Здесь оба элемента позиционируются в одно и то же место - в левый верхний угол контейнера FrameLayout, и чтобы избежать наложения, в данном случае у второго TextView устанавливается отступ сверху в 50 единиц.

FrameLayout в Android

Нередко FrameLayout применяется для создания производных контейнеров, например, ScrollView, который обеспечивает прокрутку.

Элементы управления, которые помещаются в FrameLayout, могут установить свое позиционирование с помощью атрибута **android:layout_gravity:**

```
<?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">

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="26sp"
        android:layout_gravity="center_horizontal" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Java World"
        android:textSize="26sp"
        android:layout_gravity="center"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android!"
        android:textSize="26sp"
        android:layout_gravity="bottom|center_horizontal"/>

</FrameLayout>
```

При указании значения мы можем комбинировать ряд значений, разделяя их вертикальной чертой: bottom | center_horizontal

Gravity в FrameLayout в Android

Программное создание FrameLayout в коде MainActivity:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.Gravity;
import android.widget.FrameLayout;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        FrameLayout frameLayout = new FrameLayout(this);
        TextView textView = new TextView(this);
        textView.setText("Hello World!");

        FrameLayout.LayoutParams layoutParams = new
FrameLayout.LayoutParams
                (FrameLayout.LayoutParams.WRAP_CONTENT,
FrameLayout.LayoutParams.WRAP_CONTENT);
        layoutParams.gravity = Gravity.CENTER_HORIZONTAL | Gravity.TOP;

        textView.setLayoutParams(layoutParams);
        textView.setTextSize(26);
        frameLayout.addView(textView);
        setContentView(frameLayout);
    }
}
```

GridLayout

GridLayout представляет еще один контейнер, который позволяет создавать табличные представления. GridLayout состоит из коллекции строк, каждая из которых состоит из отдельных ячеек:

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="3"
    android:columnCount="3">

    <Button android:text="1" />
```

```
    <Button android:text="2" />
    <Button android:text="3" />
    <Button android:text="4" />
    <Button android:text="5" />
    <Button android:text="6" />
    <Button android:text="7" />

    <Button android:text="8" />

    <Button android:text="9" />
</GridLayout>
```

С помощью атрибутов **android:rowCount** и **android:columnCount** устанавливается число строк и столбцов соответственно. Так, в данном случае устанавливаем 3 строки и 3 столбца. GridLayout автоматически может позиционировать вложенные элементы управления по строкам. Так, в нашем случае первая кнопка попадает в первую ячейку (первая строка первый столбец), вторая кнопка - во вторую ячейку и так далее.

При этом ширина столбцов устанавливается автоматически по ширине самого широкого элемента.

GridLayout в Android

Однако мы можем явно задать номер столбца и строки для определенного элемента, а при необходимости растянуть на несколько столбцов или строк. Для этого мы можем применять следующие атрибуты:

- **android:layout_column**: номер столбца (отсчет идет от нуля)
- **android:layout_row**: номер строки
- **android:layout_columnSpan**: количество столбцов, на которые растягивается элемент
- *android:layout_rowSpan*: количество строк, на которые растягивается элемент

Например:

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:rowCount="3"
    android:columnCount="3">

    <Button
        android:text="1"
        android:layout_column="0"
        android:layout_row="0" />
    <Button android:text="2"
        android:layout_column="1"
        android:layout_row="0"/>
    <Button android:text="3"
        android:layout_column="2"
        android:layout_row="0"  />
    <Button android:text="4"
        android:layout_width="180dp"
        android:layout_columnSpan="2"/>
    <Button android:text="5"
        android:layout_height="100dp"
        android:layout_rowSpan="2"/>
    <Button android:text="6" />
    <Button android:text="7"/>
</GridLayout>
```

Растяжение строк и столбцов в GridLayout в Android

Программное создание GridLayout

Среди методов GridLayout следует отметить методы setRowCount() и setColumnCount(), которые позволяют задать соответственно количество строк и столбцов. Например, определим в коде GridLayout, аналогичнй первому примеру в данном материале:

```
package com.example.viewapp;

 import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import android.view.Gravity;

import android.widget.Button;

import android.widget.EditText;

import android.widget.GridLayout;

import android.widget.LinearLayout;

import android.widget.TableLayout;
```

```
import android.widget.TableRow;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        GridLayout gridLayout = new GridLayout( this);
        // количество строк
        gridLayout.setRowCount(3);
        // количество столбцов
        gridLayout.setColumnCount(3);

        for(int i = 1; i <=9; i++){
            Button btn = new Button(this);
            btn.setText(String.valueOf(i));
            gridLayout.addView(btn);
        }
        setContentView(gridLayout);
    }
}
```

В данном случае GridLayout имеет три строки и три столбца. При добавлении виджетов (в данном случае кнопок) они последовательно помещаются в ячейки грида по одному виджету в ячейке.

Для более детальной настройки расположения виджета в гриде можно использовать класс **GridLayout.LayoutParams**. Этот класс имеет ряд свойств, которые позволяют настроить расположение:

- **columnSpec**: задает столбец для расположения в виде объекта GridLayout.Spec
- **rowSpec:** задает строку для расположения в виде объекта GridLayout.Spec
- **leftMargin**: задает отступ слева
- **rightMargin**: задает отступ справа
- **topMargin**: задает отступ сверху
- **bottomMargin**: задает отступ снизу
- **width**: задает ширину виджета
- **height**: задает высоту виджета

Объект GridLayout.Spec позволяет задать размещение в ячейках столбца или строки. Для создание этого объекта применяется статический метод GridLayout.spec(), который имеет ряд версий. Отметим среди них следующие:

- **GridLayout.spec(int)**: задает столбец или строку, где располагается виджет. Отсчет ячеек начинается с нуля. Виджет занимает только одну ячейку

- **GridLayout.spec(int, int)**: первый параметр задает столбец или строку, где располагается виджет. Второй параметр указывает, насколько ячеек растягивается виджет

- **GridLayout.spec(int, android.widget.GridLayout.Alignment)**: первый параметр задает столбец или строку, где располагается виджет. Второй параметр устанавливает выравнивание виджета

- **GridLayout.spec(int, int, android.widget.GridLayout.Alignment)**: первый параметр задает столбец или строку, где располагается виджет. Второй параметр указывает, насколько ячеек растягивается виджет. Третий параметр устанавливает выравнивание виджета

Пример применения GridLayout.LayoutParams:

```
Button btn = new Button(this);
btn.setText("нажми");
GridLayout.LayoutParams layoutParams = new GridLayout.LayoutParams();
 // кнопка помещается в нулевой столбец и растягивается на 2 столбца
layoutParams.columnSpec = GridLayout.spec(0,2);
 // кнопка помещается во вторую строку и растягивается на 1 строку
layoutParams.rowSpec = GridLayout.spec(1,1);
layoutParams.leftMargin=5;
layoutParams.rightMargin=5;
layoutParams.topMargin=4;
layoutParams.bottomMargin=4;
layoutParams.width = GridLayout.LayoutParams.MATCH_PARENT;
layoutParams.height = GridLayout.LayoutParams.WRAP_CONTENT;
gridLayout.addView(btn, layoutParams);
```

Например, реализуем в коде второй пример из данного материала:

```
package com.example.viewapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.util.TypedValue;
import android.view.Gravity;
import android.widget.Button;
import android.widget.EditText;
import android.widget.GridLayout;
import android.widget.LinearLayout;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        GridLayout gridLayout = new GridLayout( this);

        // количество строк
        gridLayout.setRowCount(3);
        // количество столбцов
        gridLayout.setColumnCount(3);

        for(int i = 1; i <=3; i++){
            Button btn = new Button(this);
            btn.setText(String.valueOf(i));
```

```
            gridLayout.addView(btn);
        }

        Button btn4 = new Button(this);
        btn4.setText("4");
        GridLayout.LayoutParams layoutParams4 = new
GridLayout.LayoutParams();
        layoutParams4.columnSpec = GridLayout.spec(0,2);
        layoutParams4.width = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 180,
getResources().getDisplayMetrics());
        gridLayout.addView(btn4, layoutParams4);



        Button btn5 = new Button(this);
        btn5.setText("5");
        GridLayout.LayoutParams layoutParams5 = new
GridLayout.LayoutParams();
        layoutParams5.rowSpec = GridLayout.spec(1,2);
        layoutParams5.height = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 100,
getResources().getDisplayMetrics());
        gridLayout.addView(btn5, layoutParams5);


        Button btn6 = new Button(this);
        btn6.setText("6");
        Button btn7 = new Button(this);
        btn7.setText("7");
        gridLayout.addView(btn6);
        gridLayout.addView(btn7);
```

```
        setContentView(gridLayout);
    }
}
```

ScrollView

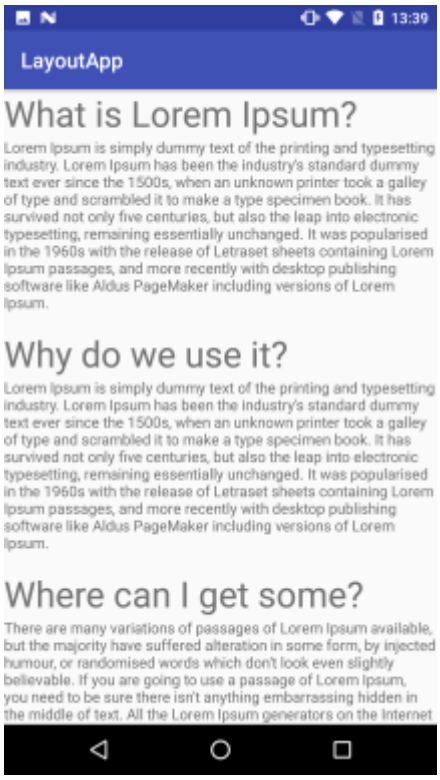
Контейнер ScrollView предназначен для создания прокрутки для такого интерфейса, все элементы которого одномоментно не могут поместиться на экране устройства. ScrollView может вмещать только один элемент, поэтому если мы хотим разместить несколько элементов, то их надо поместить в какой-нибудь контейнер.

Например, определим ряд TextView с большими текстами:

```
<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical"
        >
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="What is Lorem Ipsum?"
            android:textSize="34sp" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
        android:text="Lorem Ipsum is simply dummy text of the printing and typesetting industry...like Aldus PageMaker including versions of Lorem Ipsum."
        android:textSize="14sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Why do we use it?"
        android:layout_marginTop="16dp"
        android:textSize="34sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Lorem Ipsum is simply dummy text of the printing and typesetting industry...like Aldus PageMaker including versions of Lorem Ipsum."
        android:textSize="14sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Where can I get some?"
        android:layout_marginTop="16dp"
        android:textSize="34sp"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="There are many variations of passages of Lorem Ipsum available ... or non-characteristic words etc."
        android:textSize="14sp"/>
  </LinearLayout>
</ScrollView>
```

Так как в ScrollView можно поместить только один элемент, то все TextView заключены в LinearLayout. И если площадь экрана будет недостаточной, чтобы поместить все содержимое LinearLayout, то станет доступной прокрутка:



ScrollView в Android Studio и Java

Создание ScrollView программно в коде MainActivity:

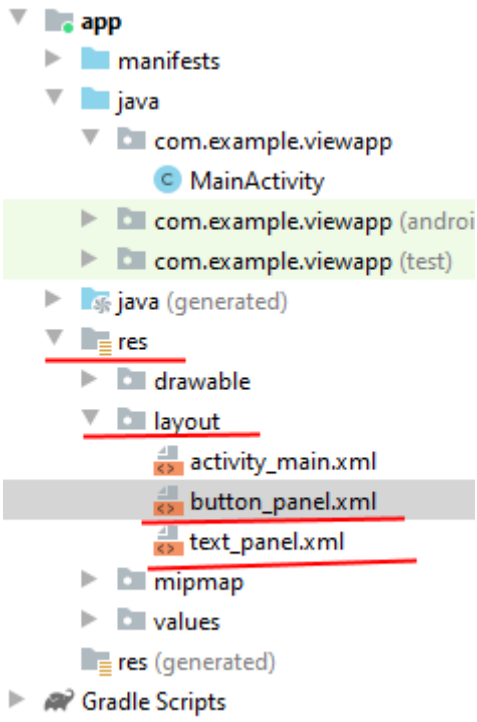
```
package com.example.viewapp;
 import androidx.appcompat.app.AppCompatActivity;
 import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.ScrollView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {



@Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);

        ScrollView scrollView = new ScrollView(this);

        TextView textView = new TextView(this);
        textView.setText("Lorem Ipsum is simply dummy text of the printing and typesetting industry...like Aldus PageMaker including versions of Lorem Ipsum.");
        textView.setLayoutParams(new ViewGroup.LayoutParams
                (ViewGroup.LayoutParams.WRAP_CONTENT, ViewGroup.LayoutParams.WRAP_CONTENT));
        textView.setTextSize(26);
        scrollView.addView(textView);
        setContentView(scrollView);
    }
}
```

Вложенные layout

Одна layout может содержать другую layout. Для этого применяется элемент **include.**

Например, добавим в папку **res/layout** два файла layout, которые пусть будут называться **text_panel.xml и button_panel.xml**:



Include layouts in Android

В файле **text_panel.xml** определим следующий код:

```
<?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"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/clicksText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="0 Clicks"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

```
    />
</androidx.constraintlayout.widget.ConstraintLayout>
```

По сути, здесь просто определено поле TextView для вывода текста.

В файле **button_panel.xml** определим 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"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click"
        android:onClick="onClick"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определена кнопка, нажатия которой мы будем обрабатывать.

Основным файлом разметки, который определяет интерфейс приложения, по-прежнему является **activity_main.xml.** Изменим его:

```
<?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"
    android:padding="16dp"
    tools:context=".MainActivity">

    <include
        android:id="@+id/textView"
        layout="@layout/text_panel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/button"
        />
    <include
        android:id="@+id/button"
        layout="@layout/button_panel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        />

</androidx.constraintlayout.widget.ConstraintLayout>
```

С помощью ConstraintLayout весь интерфейс здесь организуется в виде вертикального стека. С помощью элементов **include** внутрь ConstraintLayout добавляется содержимое файлов text_panel.xml и button_panel.xml. Для указания названия файла применяется атрибут **layout**.

Это все равно, что, если бы мы напрямую вместо элемента include добавили содержимое файлов. Однако такой способ имеет свои преимущества. Например, какая-то часть разметки, группа элементов управления может повторяться в различных activity. И чтобы не определять по сто раз эти элементы, можно вынести их в отдельный файл layout и с помощью include подключать их.

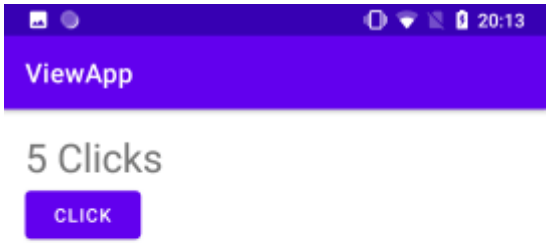
После добавления в ConstraintLayout к элементам include можно применять все те стандартные атрибуты, которые применяются в этом контейнере к вложенным элементам, например, настроить размеры, расположение. Также стоит отметить, что добавлять внешние layout можно не только в ConstraintLayout, но и в другие контейнеры (LinearLayout, RelativeLayout и т.д.)

Также изменим код **MainActivity**:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.TextView;


public class MainActivity extends AppCompatActivity {


    int clicks = 0;

    @Override

    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);
```

```
    }

    public void onClick(View view){
        TextView clicksText = findViewById(R.id.clicksText);
        clicks++;
        clicksText.setText(clicks + " Clicks");
    }
}
```

В MainActivity мы можем обращаться к элементам во вложенных файлах layout. Например, мы можем установить обработчик нажатия кнопки, в котором при нажатии изменять текст в TextView.



Вложенные файлы layout в Activity в Android

При этом мы несколько раз можем добавлять в один файл layout другой файл layout. Для этого вначале изменим файл button_panel.xml следующим образом:

```
<?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"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#3F51B5"
    android:paddingTop="10dp"
    android:paddingBottom="10dp">
    <Button
        android:id="@+id/clickBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

</androidx.constraintlayout.widget.ConstraintLayout>
```

И изменим файл activity_main.xml:

```
<?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"
    android:padding="16dp"
    tools:context=".MainActivity">

    <include
        layout="@layout/text_panel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />
    <include layout="@layout/button_panel"
        android:id="@+id/plus_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toLeftOf="@+id/minus_button"/>

    <include layout="@layout/button_panel"
        android:id="@+id/minus_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="36dp"
        app:layout_constraintLeft_toRightOf="@id/plus_button"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Теперь файл button_panel.xml добавляется два раза. Важно, что при добавлении этого файла каждому элементу include присвоен определенный id. По этому id мы сможем узнать, о каком именно элементе include идет речь.

Также изменим MainActivity:

```
package com.example.viewapp;
 import androidx.appcompat.app.AppCompatActivity;
 import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    int clicks = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        View plusButtonView = findViewById(R.id.plus_button);
        View minusButtonView = findViewById(R.id.minus_button);
        TextView clicksText = findViewById(R.id.clicksText);

        Button plusButton = plusButtonView.findViewById(R.id.clickBtn);
        Button minusButton = minusButtonView.findViewById(R.id.clickBtn);

        plusButton.setText("+");
        minusButton.setText("-");
```

```
        plusButton.setOnClickListener(v -> {
            clicks++;
            clicksText.setText(clicks + " Clicks");
        });
        minusButton.setOnClickListener(v -> {
            clicks--;
            clicksText.setText(clicks + " Clicks");
        });
    }
}
```

Здесь вначале мы получаем отдельные элементы include по id. Затем в рамках этих элементов получаем кнопку. После этого мы можем установить у кнопко любой текст и повесить обработчик события нажатия. И таким образом, поведение обеих кнопок будет различаться.

вложение layout с помощью include в android

Gravity

Атрибут gravity задает позиционирование содержимого внутри визуального элемента. Он может принимать следующие значения:

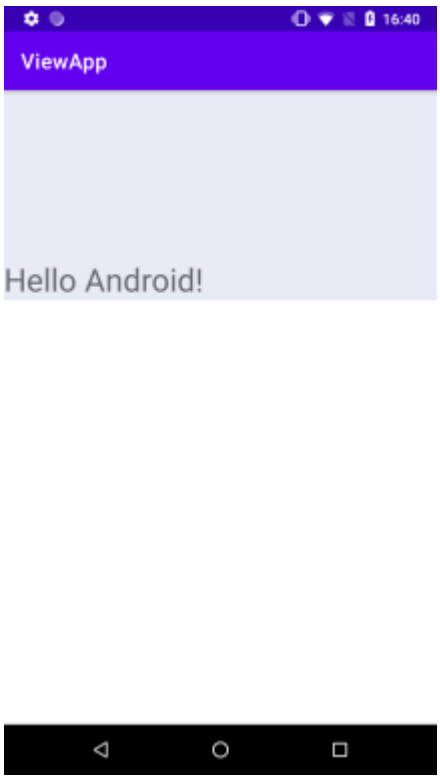
- top: элементы размещаются вверху
- bottom: элементы размещаются внизу
- left: элементы размещаются в левой стороне
- right: элементы размещаются в правой стороне контейнера
- center_vertical: выравнивает элементы по центру по вертикали
- center_horizontal: выравнивает элементы по центру по горизонтали
- 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:

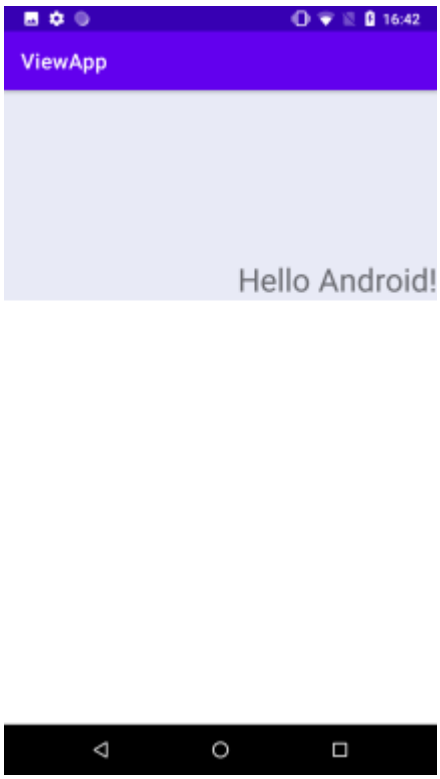
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:gravity="bottom"

        android:layout_width="0dp"
        android:layout_height="200dp"
        android:text="Hello Android!"
        android:textSize="30sp"
        android:background="#e8eaf6"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```



Gravity в Android Studio

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

```
<TextView
    android:gravity="bottom | right"

    android:layout_width="0dp"
    android:layout_height="200dp"
    android:text="Hello Android!"
    android:textSize="30sp"
    android:background="#e8eaf6"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



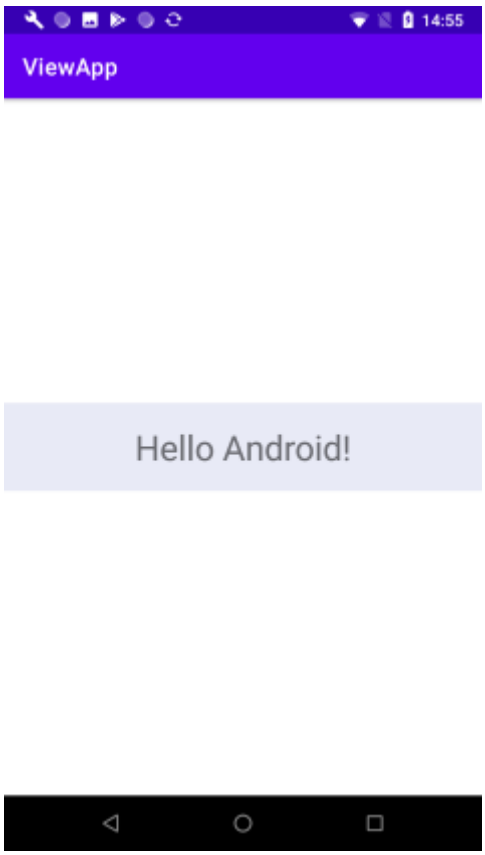
Gravity right bottom в Android

Программная установка gravity

Чтобы установить параметр gravity у элемента надо вызвать метод **setGravity().** В качестве параметра в метод передается одна из констант класса **Gravity**, которые аналогичны значениям атрибута (за тем исключенем, что названия в верхнем регистре):

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;

import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;

import android.view.Gravity;

import android.widget.TextView;

public class MainActivity extends AppCompatActivity {


    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        TextView textView = new TextView(this);
        textView.setText("Hello Android!");
        textView.setTextSize(30);
        textView.setBackgroundColor(0xffe8eaf6);

        // установка gravity
        textView.setGravity(Gravity.CENTER);

        // установка высоты и ширины
        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, 200);
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.rightToRight = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        layoutParams.bottomToBottom =
ConstraintLayout.LayoutParams.PARENT_ID;
        textView.setLayoutParams(layoutParams);

        constraintLayout.addView(textView);
        setContentView(constraintLayout);
    }
}
```



Программная установка layout_gravity в Android

Для сочетания нескольких значений также можно использовать вертикальную черту:

textView.setGravity(Gravity.BOTTOM | Gravity.CENTER);

Задание

Создать примеры реализующие различные способы разметки экрана:

- Контейнер LinearLayout. Вес элемента. Программное создание LinearLayout. Атрибут Layout_gravity.
- Контейнер RelativeLayout. Программное создание RelativeLayout.
- Контейнер TableLayout. Элемент TableRow. Атрибут layout_span. Программное создание TableLayout.
- Контейнер FrameLayout. Атрибут android:layout_gravity. Программное создание FrameLayout в коде MainActivity.
- Контейнер GridLayout. Атрибуты android:rowCount и android:columnCount. Атрибуты android:layout_column,

android:layout_row, android:layout_columnSpan. android:layout_rowSpan. Программное создание GridLayout. Класс GridLayout.LayoutParams. Свойство columnSpec, rowSpec, leftMargin, rightMargin, topMargin, bottomMargin, width, height. Объект GridLayout.Spec.

- Контейнер ScrollView. Создание ScrollView программно в коде MainActivity.
- Вложенные layout. Элемент include.
- Атрибут Gravity. Позиционирование внутри элемента. Программная установка gravity.