Практическая работа 3

Методические указания

Основы создания интерфейса

Создание графического интерфейса

Мы рассмотрели создание простого приложения, который предлагает Android Studio по умолчанию и которое просто выводит на экран строку Hello Android. Затем мы усложнили вариант вывода, используя дополнительные представления (кнопки, текстовые поля и т.д)

Но почему у нас выводится именно эта строка? Почему у нас вообще создается именно такой визуальный интерфейс?

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

```
package com.example.helloapp;

import androidx.appcompat.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);
    }
}
```

Каждый отдельный экран или страница в приложении описывается таким понятием как activity. В литературе могут использоваться различные термины: экран, страница, активность. В данном случае я буду использовать понятие "activity". Так вот, если мы запустим приложение на устройстве, то на экране мы, по сути, увидим определенную activity, которая предсталяет данный интерфейс.

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

```
package com.example.helloapp;
```

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

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
```

Затем идет собственно определение класса:

```
public class MainActivity extends AppCompatActivity
```

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.

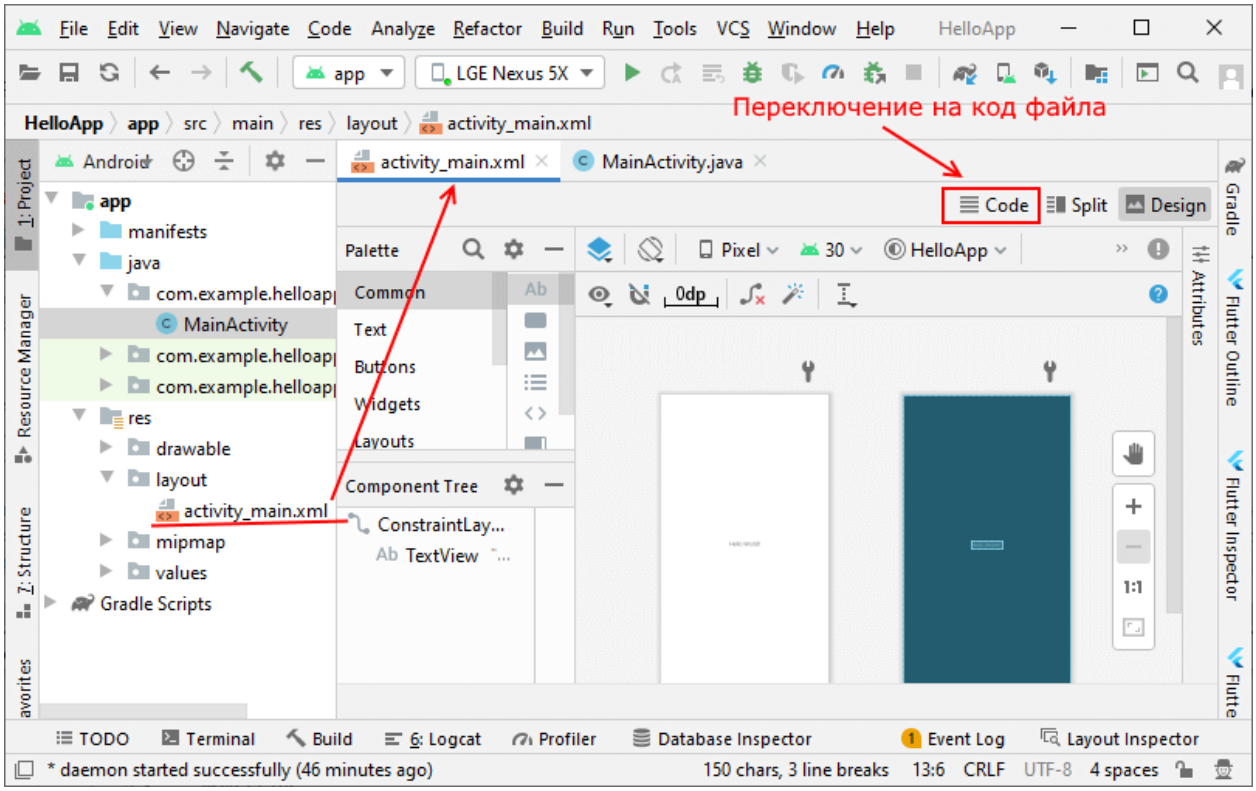
По умолчанию MainActivity содержит только один метод onCreate(), в котором фактически и создается весь интерфейс приложения:

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

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:

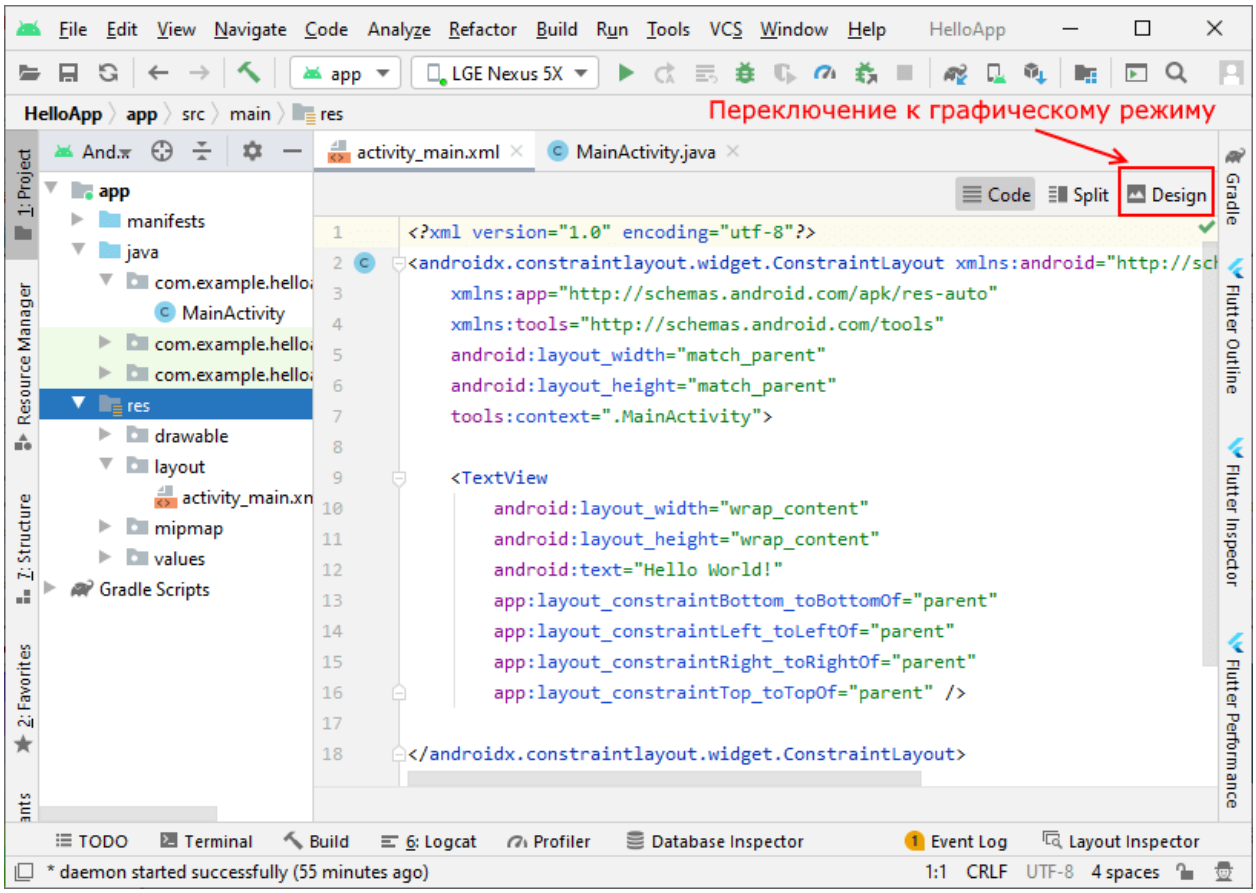


Файл activity_main.xml

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.

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 набрасать 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">
```

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.

TextView

Текстовое поле устанавливает текст с помощью атрибута android:text.

```
<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: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

android:text устанавливает текст, который будет выводиться в TextView (в данном случае это строка "Hello World!")

app:layout_constraintLeft_toLeftOf="parent": указывает, что левая граница элемента будет выравниваться по левой стороне контейнера ConstraintLayout

Обратите внимание, что этот атрибут определен в пространстве имен с префиксом app, то есть в "http://schemas.android.com/apk/res-auto".

app:layout_constraintTop_toTopOf="parent": указывает, что верхняя граница элемента будет выравниваться по верхней стороне контейнера ConstraintLayout

app:layout_constraintRight_toRightOf="parent": указывает, что правая граница элемента будет выравниваться по правой стороне контейнера ConstraintLayout

app:layout_constraintBottom_toBottomOf="parent": указывает, что нижняя граница элемента будет выравниваться по нижней стороне контейнера ConstraintLayout

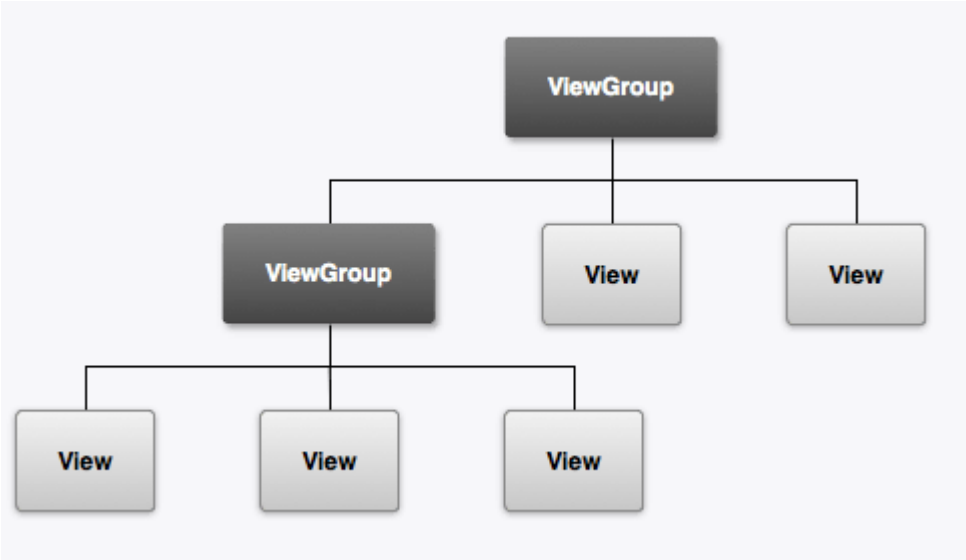
Стоит отметить, что последние четыре атрибута вместе будут приводить к расположению TextView по центру экрана.

Таким образом, при запуске приложения сначала запускается класс MainActivity, который в качестве графического интерфейса устанавливает разметку из файла activity_main.xml. И поскольку в этой разметке прописан элемент TextView, который представляет некоторый текст, то мы и увидим его текст на экране смартфона.

Основы создания интерфейса

Графический интерфейс пользователя представляет собой иерархию объектов android.view.View и android.view.ViewGroup. Каждый объектViewGroup 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

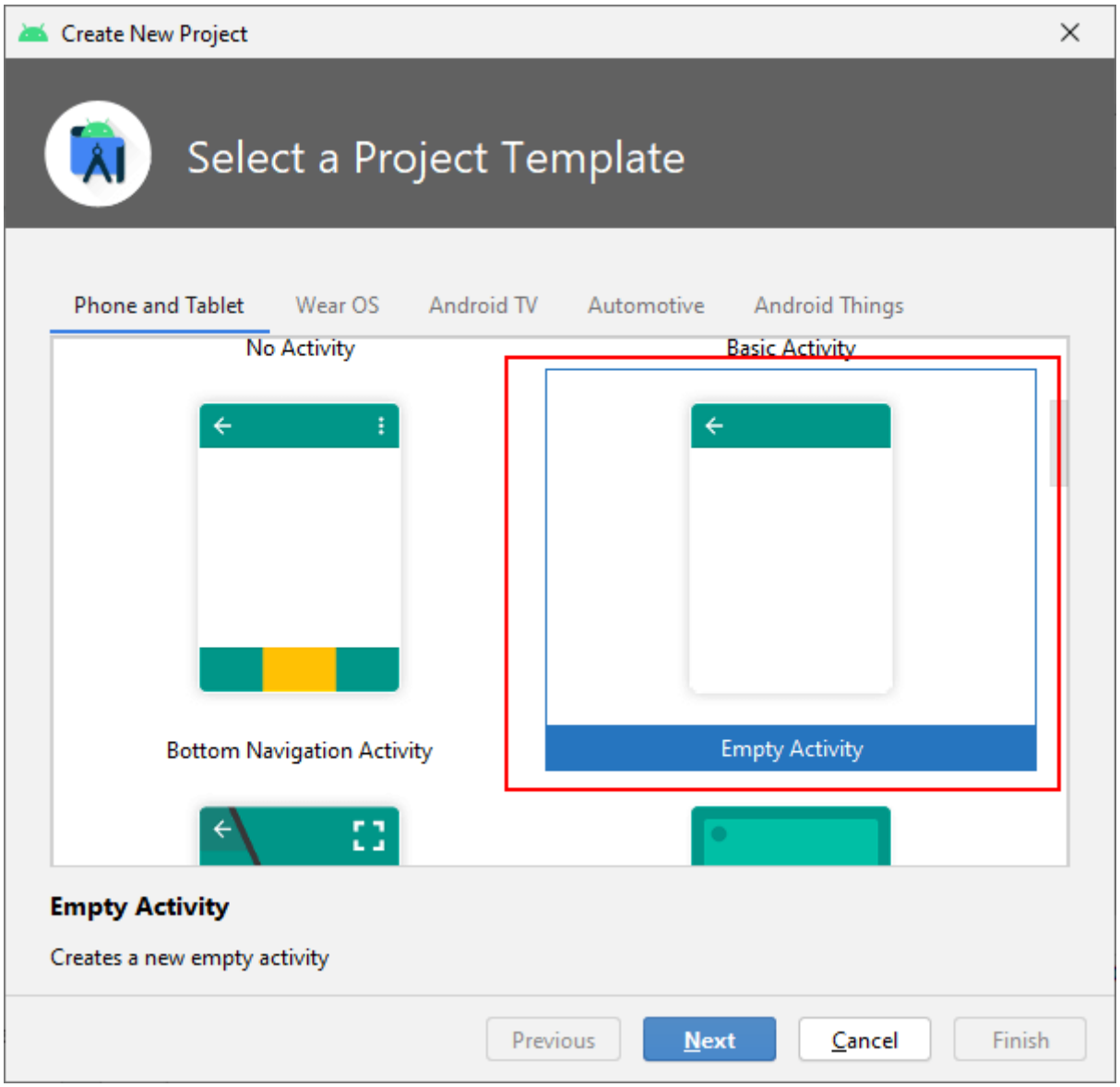
При определении визуального у нас есть три стратегии:

- 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

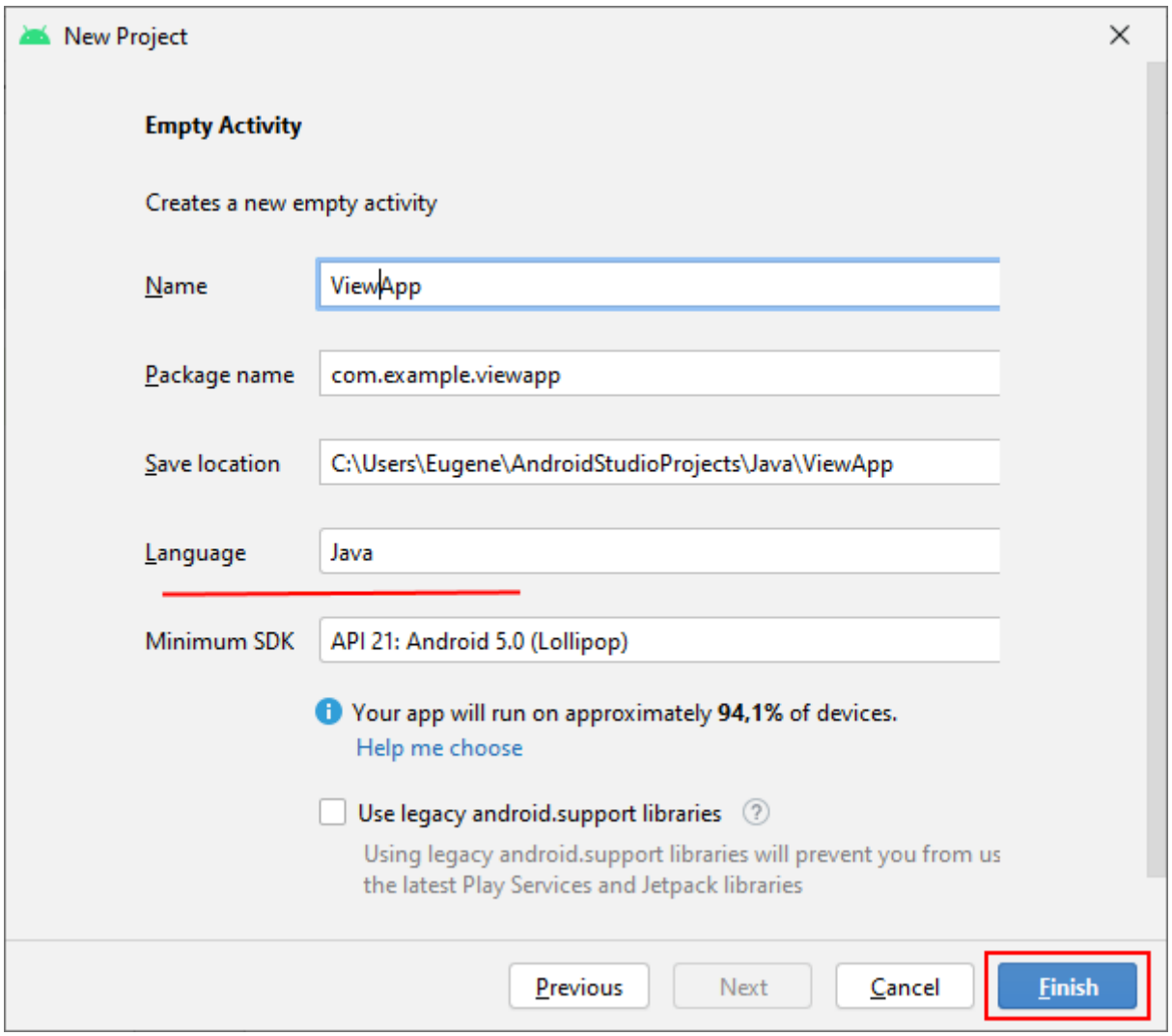
Сначала рассмотрим первую стратегию - определение интерейса в коде Java.

Создание интерфейса в коде java

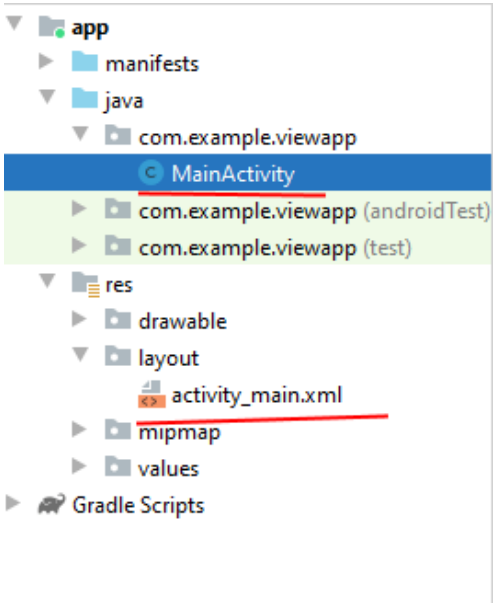
Для работы с визуальными элементами создадим новый проект. В качестве шаблона проекта выберем **Empty Activity**:



Пусть он будет называться ViewsApp:



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.**



Определим в классе MainActivity простейший интерфейс:

```java
package com.example.viewapp;

import androidx.appcompat.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);
    }
}
```

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.

```java
TextView textView = new TextView(this);
```

Здесь весь интерфейс представлен элементом TextView, которое предназначено для вывода 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:

Подобным образом мы можем создавать более сложные интерфейсы. Например, TextView, вложенный в ConstraintLayout:

```java
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
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(26);
        // устанавливаем параметры размеров и расположение элемента
        ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT, ConstraintLayout.LayoutParams.WRAP_CONTENT);
        // выравнивание по левому краю ConstraintLayout
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        // выравнивание по верхней границе ConstraintLayout
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        // устанавливаем параметры для textView
        textView.setLayoutParams(layoutParams);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);
        // в качестве корневого
        setContentView(constraintLayout);
    }
}
```

Для каждого контейнера конкретные действия по добавлению и позиционированию в нем элемента могут отличаться. В данном случае контейнеров выступает класс ConstraintLayout, поэтому для определения позиционирования и размеров элемента необходимо создать объект **ConstraintLayout.LayoutParams**. (Для LinearLayout это соответственно будет LinearLayout.LayoutParams, а для RelativeLayout - RelativeLayout.LayoutParams и т.д.). Этот объект инициализируется двумя параметрами: шириной и высотой. Для указания ширины и высоты можно использовать константу **ViewGroup.LayoutParams.WRAP_CONTENT**, которая устанавливает размеры элемента, необходимые для размещения а экране его содержимого.

Далее определяется позиционирование. В зависимости от типа контейнера набор устанавливаемых свойств может отличаться. Так, строка кода

```
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
```

указывает, что левая граница элемента будет выравниваться по левой ганице контейнера.

А строка кода

```
layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
```

указывает, что верхняя граница элемента будет выравниваться по верхней ганице контейнера. В итоге элемент будет размещен в левом верхнем углу ConstraintLayout.

Для установки всех этих значений для конкретного элемента (TextView) в его метод setLayoutParams() передается объект **ViewGroup.LayoutParams** (или один из его наследников, например, ConstraintLayout.LayoutParams).

```
textView.setLayoutParams(layoutParams);
```

Все классы контейнеров, которые наследуются от android.view.ViewGroup (RelativeLayout, LinearLayout, GridLayout, ConstraintLayout и т.д.), имеют метод void addView(android.view.View child), который позволяет добавить в контейнер другой элемент - обычный виджет типа TextView или другой контейнер. И в данном случае посредством данного метода TextView добавляется в ConstraintLayout:

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

Опять же отмечу, что для конкретного контейнера конкретные действия могут отличаться, но как правило для всех характерно три этапа:

- Создание объекта ViewGroup.LayoutParams и установка его свойств
- Передача объекта ViewGroup.LayoutParams в метод setLayoutParams() элемента
- Передача элемента для добавления в метод addView() объекта контейнера

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

Как правило, для определения визуального интерфейса в проектах под Android используются специальные файлы xml. Эти файлы являются ресурсами разметки и хранят определение визуального интерфейса в виде кода XML. 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 с пустой activity уже есть один файл ресурсов разметки activity_main.xml, который может выглядеть примерно так:

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:andro
    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>
```

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**.

Как правило, корневой элемент содержит определение используемых пространств имен XML. Например, в коде по умолчанию в ConstraintLayout мы можем увидеть такие атрибуты:

```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

```xml
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). То есть через префикс мы сможем ссылаться на функциональность этого пространства имен.

Каждое пространство имен 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). Например, в коде ConstraintLayout используется следующий атрибут из пространства имен "http://schemas.android.com/apk/res/android":

```
android:layout_width="match_parent"
```

- xmlns:app="http://schemas.android.com/apk/res-auto": содержит атрибуты, которые определены в рамках приложения. Например, в коде TextView:

```
app:layout_constraintBottom_toBottomOf="parent"
```

- xmlns:tools="http://schemas.android.com/tools": применяется для работы с режиме дизайнера в Android Studio

Это наиболее распространенные пространства имен. И обычно каждый корневой элемент (не обязательно только ConstraintLayout) их содержит. Однако, если вы не планируете пользоваться графическим дизайнером в Android Studio и хотите работать целиком в коде xml, то соответственно смысла в пространстве имен "http://schemas.android.com/tools" нет, и его можно убрать.

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.

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

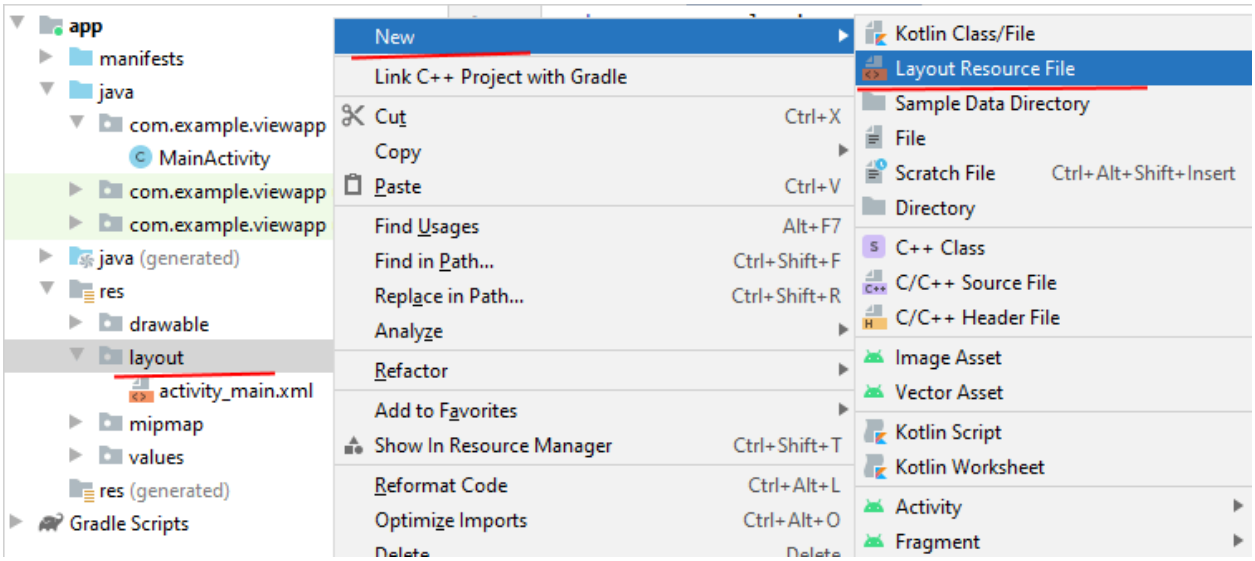
Для получения ссылки на ресурс в коде java необходимо использовать выражение R.layout.[название_ресурса]. Название ресурса layout будет совпадать с именем файла, поэтому чтобы использовать файл activity_main.xml в качестве источника визуального интерфейса, можно определить следующий код в классе MainActivity:

```java
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

        // загрузка интерфейса из файла activity_main.xml
        setContentView(R.layout.activity_main);
    }
}
```

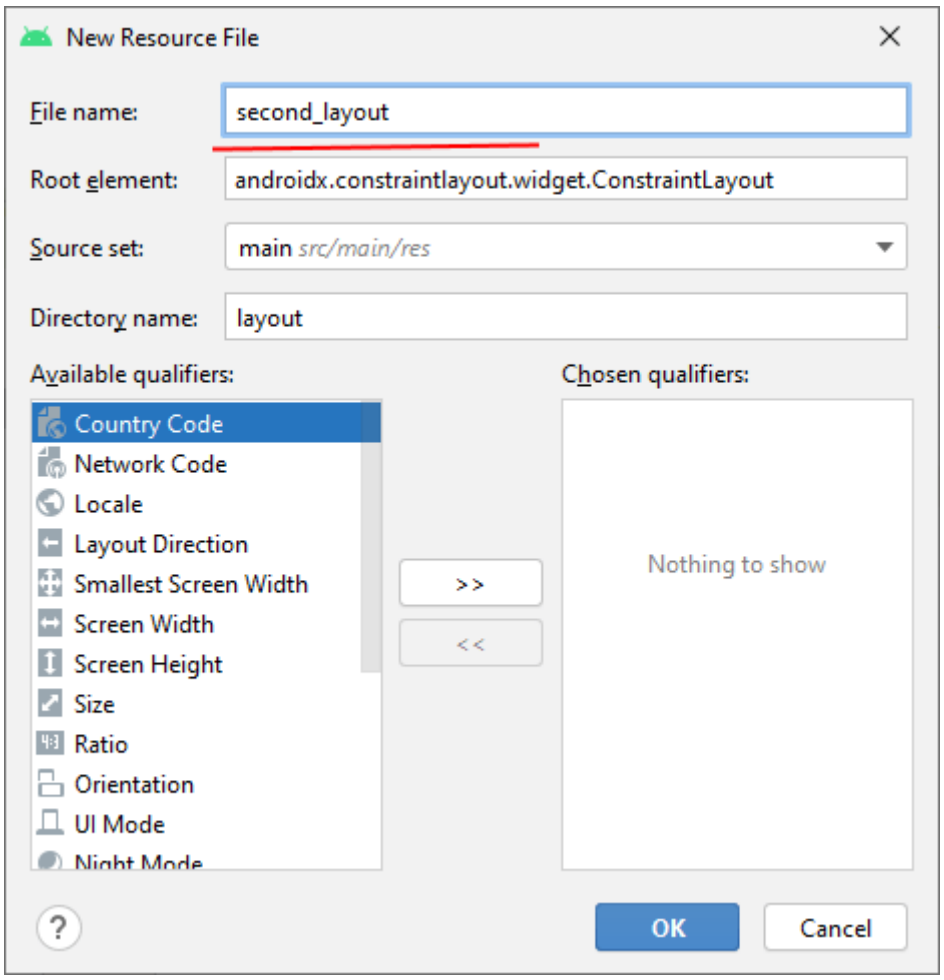
Добавление файла layout

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:



Установка параметров для файла layout в Android Studio

В качестве названия укажем second_layout. Все остальные настройки оставим по умолчанию:

- в поле Root element указывается корневой элемент. По умолчанию это androidx.constraintlayout.widget.ConstraintLayout.

- поле Source set указывает, куда помещать новый файл. По умолчанию это main - область проекта, с которой мы собственно работаем при разаботке приложения.

- поле Directory main указывает папку в рамках каталога, выбранного в предыдущей опции, в который собственно помещается новый файл. По умолчанию для файлов с разметкой интерфейса это layout.

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:

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

    <TextView
        android:id="@+id/header"
        android:text="Welcome to Android"
        android:textSize="26sp"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

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

Здесь определено текстовое поле TextView, которое имеет следующие атрибуты:

- android:id - идентификатор элемента, через который мы сможем ссылаться на него в коде. В записи android:id="@+id/header" символ @ указывает XML-парсеру использовать оставшуюся часть строки атрибута как идентификатор. А знак + означает, что если для элемента не определен id со значением header, то его следует определить.

- android:text - текст элемента - на экран будет выводиться строка "Welcome to Android".

- android:textSize - высота шрифта (здесь 26 единиц)

- android:layout_width - ширина элемента. Значение "match_parent" указывает, что элемент будет растягиваться по всей ширине контейнера ConstraintLayout

- android:layout_height - высота элемента. Значение "match_parent" указывает, что элемент будет растягиваться по всей высоте контейнера ConstraintLayout

Применим этот файл в качестве определения графического интерфейса в классе MainActivity:

```
1  package com.example.viewapp;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.os.Bundle;
5
6  public class MainActivity extends AppCompatActivity {
7
8      @Override
9      protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.second_layout);
12     }
13 }
```

Файл интерфейса называется second_layout.xml, поэтому по умолчанию для него будет создаваться ресурс R.layout.second_layout. Соответственно, чтобы его использовать, мы передаем его в метода setContentView. В итоге мы увидим на экране следующее:

Получение и управление визуальными элементами в коде

Выше определенный элемент TextView имеет один очень важный атрибут - id или идентификатор элемента. Этот идентификатор позволяет обращаться к элементу, который определен в файле xml, из кода Java. Например, перейдем к классу MainActivity и изменим его код:

```
package com.example.viewapp;

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

public class MainActivity extends AppCompatActivity {

    @Override
    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 = findViewById(R.id.header);
        // переустанавливаем у него текст
        textView.setText("Hello from Java!");
    }
}
```

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:

Получить элемент с помощью findViewById в Android Studio

Определение размеров

При разработке приложений под 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)

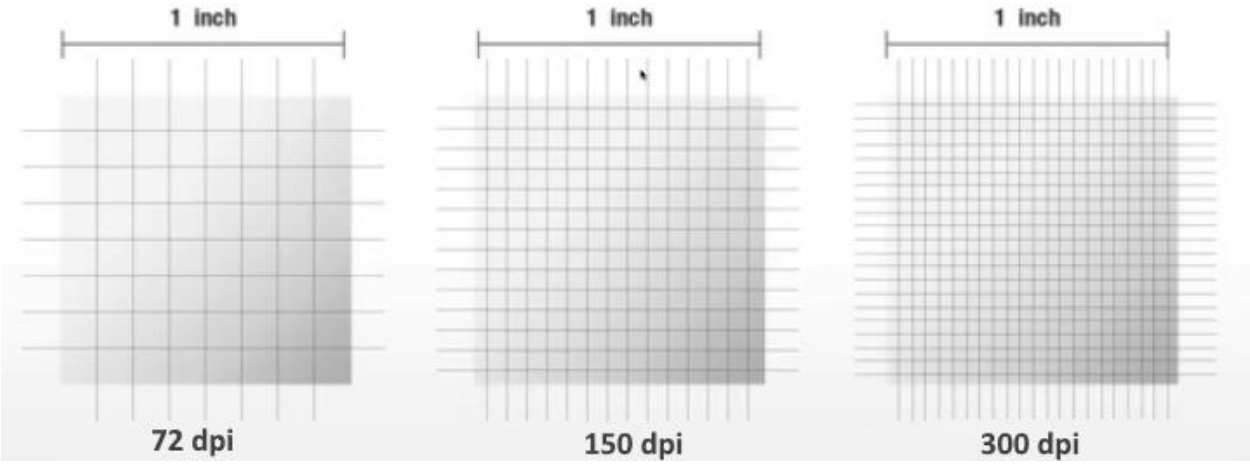
- sp: (scale-independent pixels) независимые от масштабирования пиксели. Допускают настройку размеров, производимую пользователем. Рекомендуются для работы со шрифтами.

- pt: 1/72 дюйма, базируются на физических размерах экрана

- mm: миллиметры

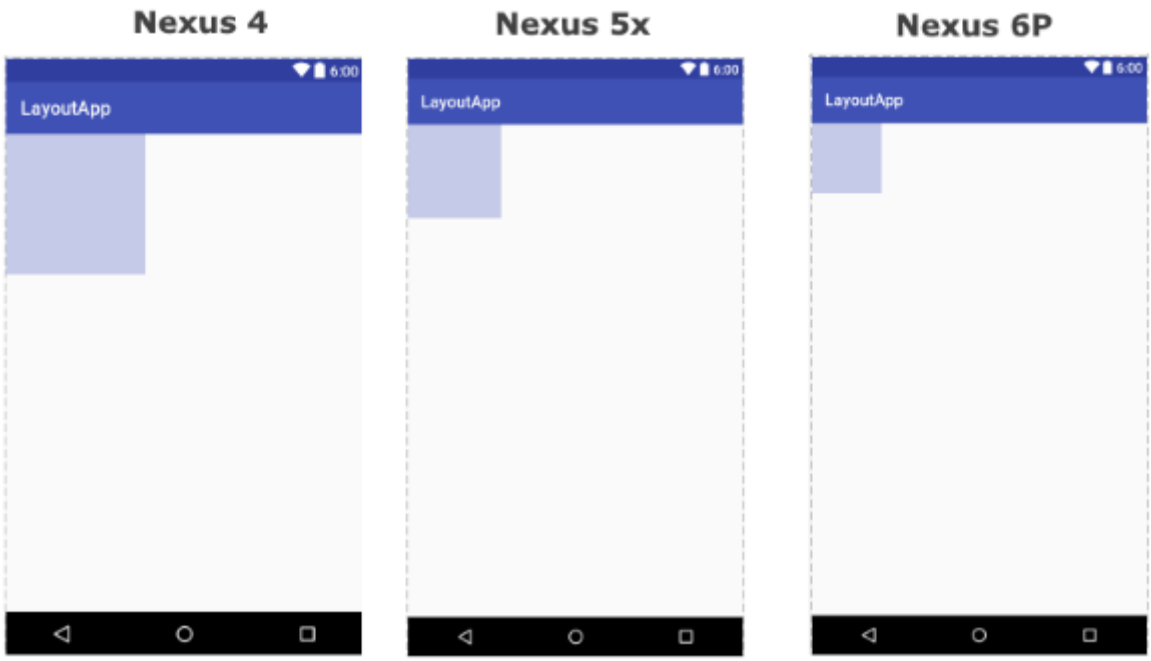
- in: дюймы

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:



Разрешение экрана в Android

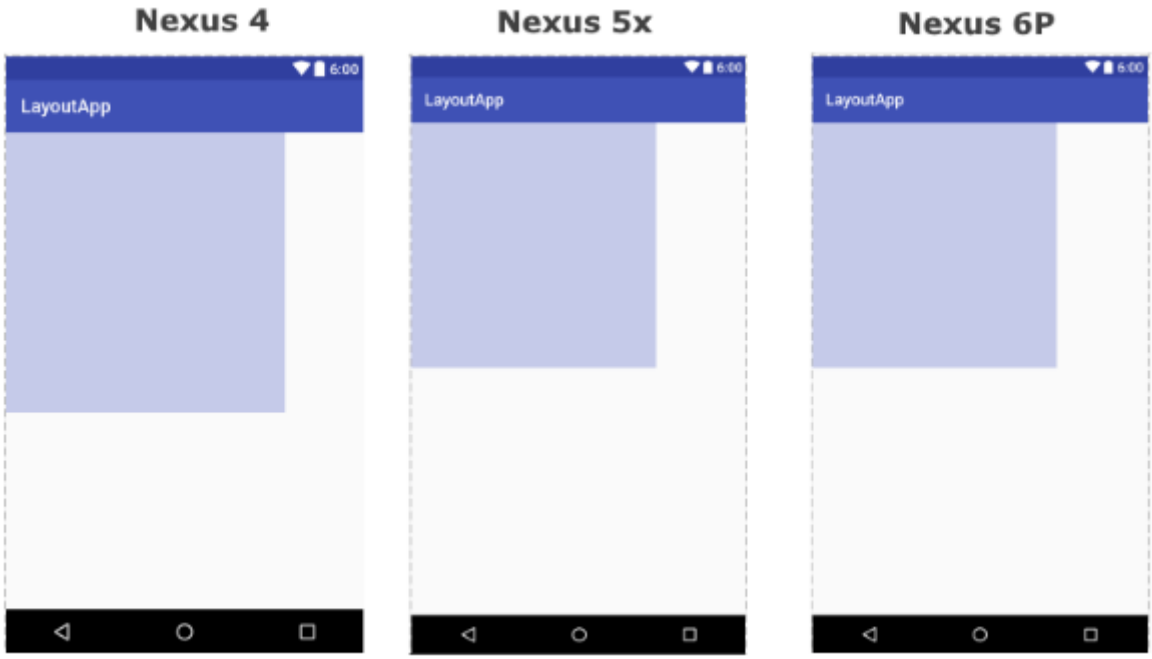
Используя же стандартные физические пиксели, мы можем столкнуться с проблемой, что размеры элементов также будут сильно варьироваться в зависимости от плотности пикселей устройства. Например, возьмем 3 устройства с различными характеристиками экрана Nexus 4, Nexus 5X и Nexus 6P и выведем на экран квадрат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



Физические пиксели в Android

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

Теперь также возьмем квадрат со сторонами 300x300, но теперь вместо физических пикселей используем единицы dp:



независимые от плотности пиксели в Android

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 древнее 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)

Установка размеров

Основная проблема, связанная с размерами, связана с их установкой в коде Java. Например, некоторые методы принимают в качестве значения физические пиксели, а не device-independent pixels. В этом случае может потребоваться перевести значения из одного типа единиц в другой. Для этого требуется применить метод TypedValue.applyDimension(), который принимает три параметра:

```
1 public static float applyDimension(int unit,
2                                    float value,
3                                    android.util.DisplayMetrics metrics)
```

Параметр unit представляет тип единиц, из которой надо получить значение в пикселях. Тип единиц описывается одной из констант **TypedValue**:

- COMPLEX_UNIT_DIP - dp или независимые от плотности экрана пиксели
- COMPLEX_UNIT_IN - in или дюймы
- COMPLEX_UNIT_MM - mm или миллиметры
- COMPLEX_UNIT_PT - pt или точки
- COMPLEX_UNIT_PX - px или физические пиксели
- COMPLEX_UNIT_SP - sp или независимые от масштабирования пиксели (scale-independent pixels)

Параметр value представляет значение, которое надо преобразовать

Параметр metrics представляет информацию о метрике, в рамках которой надо выполнить преобразование.

В итоге метод возвращает преобразованное значение. Рассмотрим абстрактный пример. Например, нам надо получить из 60dp обычные физические пиксели:

```
int valueInDp = 60;
int valueInPx = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, valueInDp, getResources().getDisplayMetrics());
```

В качестве третьего аргумента передается вызов метода getResources().getDisplayMetrics(), который позволяет получить информацию о метрике, связанной с текущим устройством. В итоге мы получим из 60dp некоторое количество пикселей.

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 использовать одну из следующих опций:

Растяжение по всей ширине или высоте контейнера с помощью значения match_parent (для всех контейнеров кроме ConstraintLayout) или 0dp (для ConstraintLayout)

Растяжение элемента до тех границ, которые достаточны, чтобы вместить все его содержимое с помощью значения wrap_content

Точные размеры элемента, например 96 dp

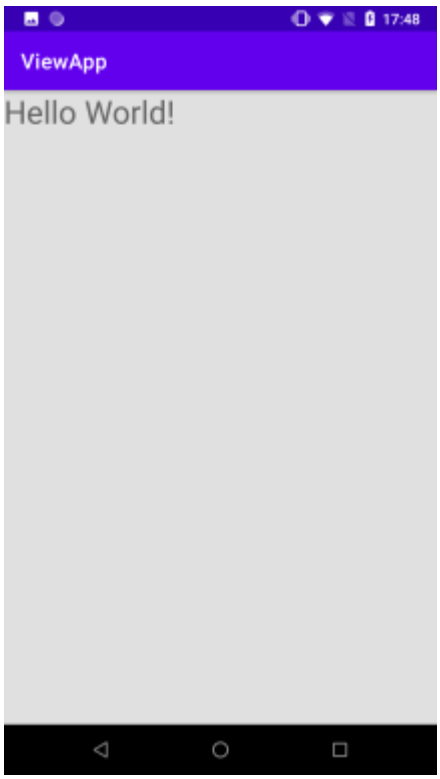
match_parent

Установка значения match_parent позволяет растянуть элемент по всей ширине или высоте контейнера. Стоит отметить, что данное значение применяется ко всем контейнерам, кроме ConstraintLayout. Например, рястянем элемент TextView по всей ширине и высоте контейнера LinearLayout:

```
<?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">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0" />

</LinearLayout>
```

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

И TextView также принимает подобные атрибуты. Значение android:layout_width="match_parent" обеспечивает растяжение по ширине, а android:layout_height="match_parent" - по вертикали. Для наглядности в TextView применяет атрибут android:background, который представляет фон и в данном случае окрашивает элемент в цвет "#e0e0e0", благодаря чему мы можем увидеть занимаемую им область.



Высота и длина match_parent в Android

Следует учитывать, что значение match_parent можно применять почти во всех встроенных контейнерах, типа LinearLayout или RelativeLayout и их элементах. Однако match_parent не рекомендуется применять к элементам внутри ConstraintLayout. Вместо "match_parent" в ConstraintLayout можно использовать значение 0dp, чтобы растянуть элемент по горизонтали или вертикали:

```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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        />

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

Стоит отметить, что ConstraintLayout сама также растягивается по ширине и высоте экрана с помощью значения "match_parent" в атрибутах layout_width и android:layout_height, но к вложенным элементам это значение не рекомендуется применять.

Поскольку ConstraintLayout имеет некоторые особенности при установке размеров, то более подробно работа с размерами элементов именно в ConstraintLayout раскрыта более подробно в одной из следующих тем.

wrap_content

Значение wrap_content устанавливает те значения для ширины или высоты, которые необходимы, чтобы разместить на экране содержимое элемента:

```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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#ffcdd2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />

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

Здесь элемент TextView растягивается до тех значений, которые достаточны для размещения его текста.

Высота и длина wrap_content в Android Studio

Установка точных значений

Также мы можем установить точные значения:

Кроме того, можно комбинировать несколько значений, например, растянуть по ширине содержимого и установить точные значения для высоты:

```
<TextView
        android:layout_height="80dp"
        android:layout_width="wrap_content"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
/>
```

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:

```
<TextView
        android:minWidth="200dp"
        android:maxWidth="250dp"
        android:minHeight="100dp"
        android:maxHeight="200dp"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
/>
```

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:

```java
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

        ConstraintLayout constraintLayout = new ConstraintLayo
        TextView textView = new TextView(this);
        textView.setText("Hello Android");
        textView.setTextSize(26);

        // устанавливаем параметры размеров и расположение эле
        ConstraintLayout.LayoutParams layoutParams = new Const
                (ConstraintLayout.LayoutParams.WRAP_CONTENT, C
        // эквивалент app:layout_constraintLeft_toLeftOf="pare
        layoutParams.leftToLeft = ConstraintLayout.LayoutParam
        // эквивалент app:layout_constraintTop_toTopOf="parent
        layoutParams.topToTop = ConstraintLayout.LayoutParams.
        // устанавливаем параметры для textView
        textView.setLayoutParams(layoutParams);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);
        setContentView(constraintLayout);
    }
}
```

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 (в случае с LinearLayout или RelativeLayout).

Поскольку в данном случае мы имеем дело с контейнером ConstraintLayout, то для установки размеров применяется значение ConstraintLayout.LayoutParams.WRAP_CONTENT. В реальности класс androidx.constraintlayout.widget.ConstraintLayout.LayoutParams, который

предоставляет это значение, наследуется от android.view.ViewGroup.LayoutParams

Программная установка высоты и длины в Android и ViewGroup.LayoutParams.MATCH_PARENT

Также мы можем передать точные значения или комбинировать типы значений:

```
ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT, 200);
```

Внутренние и внешние отступы

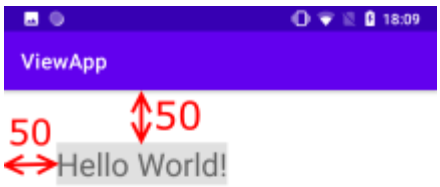
Параметры разметки позволяют задать отступы как от внешних границ элемента до границ контейнера, так и внутри самого элемента между его границами и содержимым.

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.

```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="50dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />

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

У контейнера ConstraintLayout установлен только один общий внутренний отступ в 50 единиц. Вложенный элемент TextView позиционируется в левом верхнем углу контейнера (благодаря атрибутам app:layout_constraintLeft_toLeftOf="parent" и app:layout_constraintTop_toTopOf="parent"). . Поэтому TextView будет отодвигаться от начальной точки (левый верхний угол контейнера ConstraintLayout) вниз и влево на 50 единиц. Кроме того, такие же отступы будут действовать справа и снизу, если элемент будет примыкать к нижней или правой границе контейнера.



Установка одного отступа

```
android:padding="50dp"
```

Будет аналогична установке четырех отступов

```
android:paddingTop="50dp"
android:paddingLeft="50dp"
android:paddingBottom="50dp"
android:paddingRight="50dp"
```

Подобным образом можно установить отступы в других элементах. Например, установим внутри TextView сверху и снизу от внутреннего содержимого (то есть текста) отступы в 60 единиц и отступы слева и справа в 40 единиц:

```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="50dp"
    tools:context=".MainActivity">

    <TextView
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:paddingTop="60dp"
        android:paddingLeft="40dp"
        android:paddingRight="40dp"
        android:paddingBottom="60dp"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        />

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

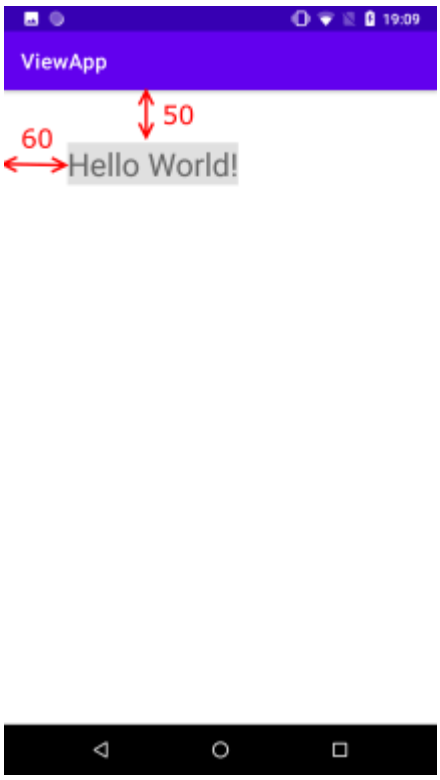
Стоит отметить, что вместо атрибутов android:paddingLeft и android:paddingRight можно применять атрибуты android:paddingStart и android:paddingEnd, которые разработаны специально адаптации приложения для работы как для языков с левосторонней ориентацией, так и правосторонней ориентацией (арабский, фарси).

Margin

Для установки внешних отступов 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):

```
<?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_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginTop="50dp"
        android:layout_marginLeft="60dp"
        android:text="Hello World!"
        android:textSize="30sp"
        android:background="#e0e0e0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"
    />

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

Здесь у TextView задаются отступы от двух сторон ConstraintLayout (слева 60 единиц и сверху 50 единиц):



Программная установка отступов

Для программной установки внутренних отступов у элементы вызывается метод setPadding(left, top, right, bottom), в который передаются четыре значения для каждой из сторон. Также можно по отдельности задать отступы с помощью методов getPaddingLeft(), getPaddingTop(), getPaddingRight() и getPaddingBottom().

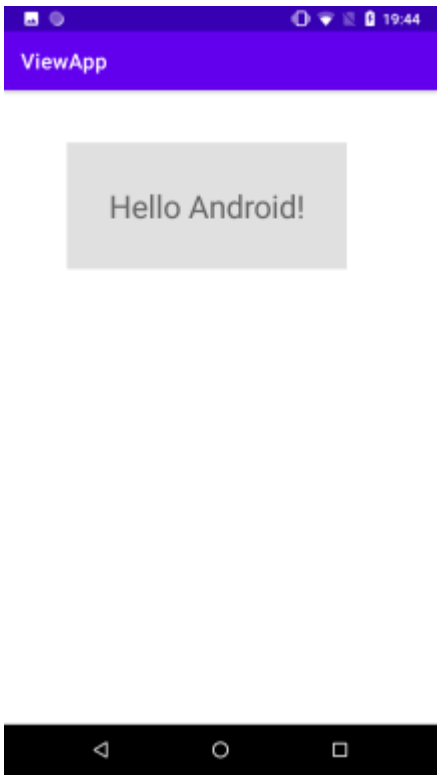
Для установки внешних отступов необходимо реализовать объект LayoutParams для того контейнера, который применяется. И затем вызвать у этого объекта LayoutParams метод setMargins(left, top, right, bottom):

```java
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
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);
        // установка цвета текстового поля
        textView.setBackgroundColor(0xFFE0E0E0);
        // установка текста текстового поля
        textView.setText("Hello Android");
        // установка размера текста
        textView.setTextSize(30);

        ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLayout.LayoutParams.WRAP_CONTENT);
        // установка внешних отступов
        layoutParams.setMargins(60, 50, 60, 50);
        // позиционирование в левом верхнем угду контейнера
        // эквивалент app:layout_constraintLeft_toLeftOf="parent"
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        // эквивалент app:layout_constraintTop_toTopOf="parent"
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        // устанавливаем размеры
        textView.setLayoutParams(layoutParams);
        // установка внутренних отступов
        textView.setPadding(40,40,40,40);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```

Поскольку в данном случае элемент TextView добавляется в контейнер типа ConstraintLayout, то для его позиционирования применяется объект ConstraintLayout.LayoutParams (соответственно для LinearLayout это будет LinearLayout.LayoutParams), у которого вызывается метод setMargins().

Но если посмотреть на последний скриншот, то можно увидеть, что несмотря на то, что отступы вроде бы заданы также, что и в предпоследнем примере в файле layout, однако в реальности на экране мы увидим отступы со совсем другими значениями. Дело в том, что методы setPadding() и setMargins() принимают значения в пикселях, тогда как в файле layout применялись единицы dp. И чтобы использовать dp также в коде, необходимо выполнить преобразования:

```java
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;
import android.util.TypedValue;
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.setBackgroundColor(0xFFE0E0E0);
        textView.setText("Hello Android!");
        textView.setTextSize(30);

        // получаем отступ в пикселях для 50 dp
        int margin50inDp = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 50, getResources().getDisplayMetrics());
        // получаем отступ в пикселях для 60 dp
        int margin60inDp = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 60, getResources().getDisplayMetrics());
        // получаем отступ в пикселях для 40 dp
        int padding40inDp = (int) TypedValue.applyDimension(
                TypedValue.COMPLEX_UNIT_DIP, 40, getResources().getDisplayMetrics());

        ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLayout.LayoutParams.WRAP_CONTENT);
        // установка внешних отступов
        layoutParams.setMargins(margin60inDp, margin50inDp, margin60inDp, margin50inDp);
        // выравнивание по левому краю ConstraintLayout
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        // выравнивание по верхней границе ConstraintLayout
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        // устанавливаем размеры
        textView.setLayoutParams(layoutParams);
        // установка внутренних отступов
        textView.setPadding(padding40inDp, padding40inDp, padding40inDp, padding40inDp);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```



ConstraintLayout

ConstraintLayout представляет контейнер, который позволяет создавать гибкие и масштабируемые визуальные интерфейсы.

Для позиционирования элемента внутри ConstraintLayout необходимо указать ограничения (constraints). Есть несколько типов ограничений. В частности, для установки позиции относительно определенного элемента испльзуются следующие ограничения:

layout_constraintLeft_toLeftOf: левая граница позиционируется относительно левой границы другого элемента

layout_constraintLeft_toRightOf: левая граница позиционируется относительно правой границы другого элемента

layout_constraintRight_toLeftOf: правая граница позиционируется относительно левой границы другого элемента

layout_constraintRight_toRightOf: правая граница позиционируется относительно правой границы другого элемента

layout_constraintTop_toTopOf: верхняя граница позиционируется относительно верхней границы другого элемента

layout_constraintTop_toBottomOf: верхняя граница позиционируется относительно нижней границы другого элемента

layout_constraintBottom_toBottomOf: нижняя граница позиционируется относительно нижней границы другого элемента

layout_constraintBottom_toTopOf: нижняя граница позиционируется относительно верхней границы другого элемента

layout_constraintBaseline_toBaselineOf: базовая линия позиционируется относительно базовой линии другого элемента

layout_constraintStart_toEndOf: элемент начинается там, где завершается другой элемент

layout_constraintStart_toStartOf: элемент начинается там, где начинается другой элемент

layout_constraintEnd_toStartOf: элемент завершается там, где начинается другой элемент

layout_constraintEnd_toEndOf: элемент завершается там, где завершается другой элемент

Возможно, по поводу четырех последних свойств возникло некоторое непонимание, что подразумевается под началом или завершением элемента. Дело в том, что некоторые языки (например, арабский или фарси) имеют правостороннюю ориентацию, то есть символы идут справа налево, а не слева направо, как в европейских языках. И в зависимости от текущей ориентации - левосторонняя или правосторонняя - будет изменяться то, где именно начало, а где завершение элемента. Например, при левосторонней ориентации начало - слева, а завершение - справа, поэтому атрибут layout_constraintStart_toEndOf фактически будет аналогичен атрибуту layout_constraintLeft_toRightOf. А при правосторонней ориентации - атрибуту layout_constraintRight_toLeftOf, так как начало справа, а завершение - слева

Каждое ограничение устанавливает позиционирование элемента либо по горизонтали, либо по вертикали. И для определения позиции элемента в ConstraintLayout необходимо указать как минимум одно ограничение по горизонтали и одно ограничение по вертикали.

Позиционирования может производиться относительно границ самого контейнера ContentLayout (в этом случае ограничение имеет значение parent), либо же относительно любого другого элемента внутри ConstraintLayout, тогда в качестве значения ограничения указывается id этого элемента.

Простейший пример:

```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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="30sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

В данном случае у элемента TextView установлено два ограничение: одно погоризонтальной линии (app:layout_constraintLeft_toLeftOf="parent"), второе - по вертикальной линии (app:layout_constraintTop_toTopOf="parent"). Оба ограничения устанавливаются относительно контейнера ConstraintLayout, поэтому они принимают значение parent, то есть ConstraintLayout.

Ограничение app:layout_constraintLeft_toLeftOf="parent" устанавливает левую границу TextView у левой границы контейнера.

Ограничение app:layout_constraintTop_toTopOf="parent" устанавливает верхнюю границу TextView у верхней границы контейнера.

В итоге TextView будет располагаться в верхнем левом углу контейнера.

Расположение элементов в ConstraintLayout в Android Studio

Стоит обратить внимание, что все эти атрибуты ограничений берутся из пространства имен "http://schemas.android.com/apk/res-auto", которое проецируется на префикс app.

Если необходимо установить ограничение относительно другого элемента, то необходимо указать id этого элемента:

```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"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="180dp"
        android:layout_height="wrap_content"
        android:hint="Введите Email"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Отправить"
        app:layout_constraintLeft_toRightOf="@id/editText"
        app:layout_constraintTop_toTopOf="parent" />

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

Здесь для текстового поля ввода EditText устанавливаются два ограничения относительно родительского контейнера ConstraintLayout, поэтому ограничения имеют значение parent, а сам EditText выравнивается по левой и верхней границе контейнера. Верхняя граница кнопки Button также выравнивается по верхней границе контейнера. А вот левая граница кнопки выравнивается по правой границе EditText. Для этого в качестве значения атрибута передается id EditText:

```xml
app:layout_constraintLeft_toRightOf="@id/editText"
```

Позиционирование в ConstraintLayout в Android

Подобным образом можно составлять различные комбинации атрибутов для определения нужного нам позиционирования. Например, изменим код кнопки:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Отправить"
    app:layout_constraintLeft_toRightOf="@id/editText"
    app:layout_constraintTop_toBottomOf="@id/editText" />
```

В данном случае верхняя граница кнопки выравнивается по нижней границе EditText

Позиционирование в ConstraintLayout в Android

Более того мы можем позиционировать оба элемента один относительно другого:

```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"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Введите Email"
        app:layout_constraintRight_toLeftOf="@+id/button"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Отправить"
        app:layout_constraintLeft_toRightOf="@id/editText"
        app:layout_constraintTop_toTopOf="parent" />

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

Позиционирование в центре

Если необходимо расположить элемент в центре контейнера по вертикали, то надо использовать пару атрибутов

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintBottom_toBottomOf="parent"
```

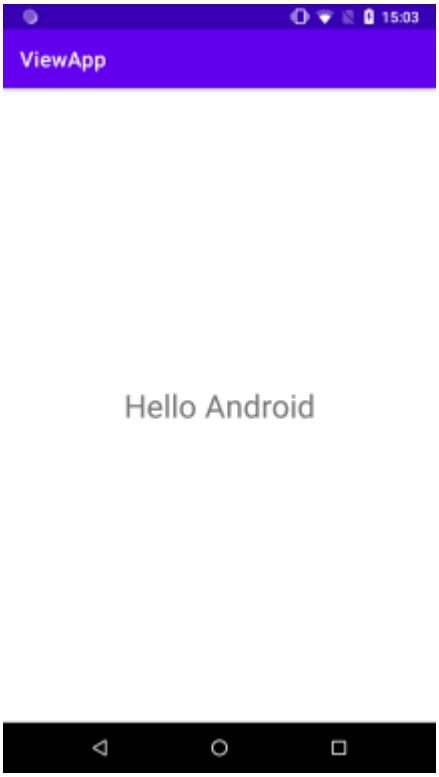
Если необходимо расположить элемент в центре контейнера по горизонтали, то надо использовать следующую пару атрибутов

```
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
```

Соответственно для расположения в центре контейнера по вертикали и горизонтали надо применить все выше указанные четыре атрибута:

```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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello Android"
        android:textSize="30sp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent" />

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

Позиционирование в центре ConstraintLayout в Android



Сдвиг

Если элементы расположены по центру, ConstraintLayout позволяет их сдвигать по горизонтали и по вертикали. Для сдвига по горизонтали применяется атрибут layout_constraintHorizontal_bias, а для сдвига по вертикали - атрибут layout_constraintVertical_bias. В качестве значения они принимают число с плавающей точкой от 0 до 1. Значение по умолчанию - 0.5 (расположение по центру). Например:

```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"
    tools:context=".MainActivity">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Top TextView"
        android:textSize="30sp"
        android:background="#e0e0e0"

        app:layout_constraintHorizontal_bias="0.2"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bottom TextView"
        android:textSize="30sp"
        android:background="#e0e0e0"

        app:layout_constraintHorizontal_bias=".9"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"/>

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

Первый TextView сдвигается на 20% от левой границы контейнера (значение по умолчанию - 0.5, поэтому при значении 0.2 элемент сдвигается влево). Второй TextView сдвигается на 90% от левой границы контейнера. Например, значение 1 означало бы, что элемент придвинут к правой границе, а значение 0 - к левой

Аналогично работает атрибут layout_constraintVertical_bias, который сдвигает по вертикали.

Цепочки элементов в ConstraintLayout

ConstraintLayout позволяет организовать расположение элементов в ряд по горизонтали или по вертикали или то, что в Android называется chains или цепочки. Мы можем по цепочке установить позиционирование одного элемента относительно другого и таким обазом организовать ряд элементов.

Горизонтальная цепочка элементов

Например, ряд элементов по горизонтали:

```
<?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:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"
    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="Third"
        android:textSize="30sp"
        app:layout_constraintLeft_toRightOf="@id/textView2"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

В итоге элементы цепочки равномерно будут растянуты по всей ширине контейнера:

Horisontal chains in ConstraintLayout in Android

Горизонтальная цепочка элементов достигается за счет двух факторов:

- Первый элемент выравнивается относительно левой границы контейнера (app:layout_constraintLeft_toLeftOf="parent"), последний элемент выравнивается относительно правой границы контейнера (app:layout_constraintRight_toRightOf="parent").

- Благодаря установке атрибутов app:layout_constraintLeft_toRightOf и app:layout_constraintRight_toLeftOf располагаем один элемент справа или слева от другого.

Кроме того, ConstraintLayout позволяет настроить положение элементов внутри цепочки. Для этого применяется атрибут layout_constraintHorizontal_chainStyle, который может принимать следующие значения:

- spread: значение по умолчанию, при котором элементы цепочки равномерно растягиваются по всей длине цепочки, как в примере выше

- spread_inside: первый и последний элемент цепочки примыкают к границами контейнера

- packed: элементы цепочки располагаются вплотную друг к другу.

Например, применим значение spread_inside:

```
<?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:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintHorizontal_chainStyle="spread_inside"

    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="Third"
        android:textSize="30sp"
        app:layout_constraintLeft_toRightOf="@id/textView2"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Причем в данном случае достаточно установить атрибут у первого элемента цепочки:

Horisontal chains spread_inside in ConstraintLayout in Android

Значение packed:

```
<?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:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="First"
        android:textSize="30sp"

        app:layout_constraintHorizontal_chainStyle="packed"

        app:layout_constraintRight_toLeftOf="@id/textView2"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#e0e0e0"
```

```
        android:text="Second"
        android:textSize="30sp"
        app:layout_constraintLeft_toRightOf="@id/textView1"
        app:layout_constraintRight_toLeftOf="@id/textView3"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="Third"
        android:textSize="30sp"
        app:layout_constraintLeft_toRightOf="@id/textView2"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Horisontal chains packed in ConstraintLayout in Android

Вес элемента

Стоит отметить, что выше у элементов устанавливалась ширина, необходимая для их содержимого. Но мы могли бы установить и нулевую ширину, тогда элементы равномерно бы распределялись по всей цепочки без образования промежутков между ними.

```
<?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:id="@+id/textView1"
```

```
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"
    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"
    app:layout_constraintLeft_toRightOf="@id/textView2"
    app:layout_constraintRight_toRightOf="parent"
```

app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

Горизонтальный ряд виджетов в ConstraintLayout в Android

В этом случае значение атрибута app:layout_constraintHorizontal_chainStyle не играет никакой роли, так как все элементы итак растягиваются по всей цепочке.

Однако такое поведение может не устраивать, например, мы хотим, чтобы один элемент был два раза больше другого. И в этом случае мы можем с помощью атрибута layout_constraintHorizontal_weight. Однако следует учитывать, что при применении весов у элементов, они должны иметь нулевую ширину:

<?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:id="@+id/textView1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintHorizontal_weight="1"

    app:layout_constraintRight_toLeftOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"

    app:layout_constraintHorizontal_weight="2"

    app:layout_constraintLeft_toRightOf="@id/textView1"
    app:layout_constraintRight_toLeftOf="@id/textView3"
```

```
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="Third"
        android:textSize="30sp"

        app:layout_constraintHorizontal_weight="1"

        app:layout_constraintLeft_toRightOf="@id/textView2"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

В качестве значения атрибут layout_constraintHorizontal_weight принимает число - вес элемента. Так, в данном случае вес первого элемента - 1, вес второго - 2, а вес третьего - 1. Поэтому вся ширина контейнера будет условно поделена на 1 + 2 + 1 = 4 частей, из которых по одной части займут первый и третий элемент, а второй займет 2 части, то есть второй элемент будет в два раза больше первого и третьего элемента.

Вес виджетов и layout_constraintHorizontal_weight в ConstraintLayout в Android

В принципе мы можем оставить элементы и с шириной "wrap_content" или конкретным значением, отличным от "0dp", просто в этом случае они не будут участвовать в распределении пространства контейнера и вес у такого элемента роли играть не будет.

Вертикальная цепочка

Для образования вертикальной цепочки также должно соблюдаться два условия:

- Первый элемент выравнивается относительно верхней границы контейнера (app:layout_constraintTop_toTopOf="parent"), последний элемент выравнивается относительно нижней границы контейнера (app:layout_constraintBottom_toBottomOf="parent").

- Благодаря установке атрибутов app:layout_constraintBottom_toTopOf и app:layout_constraintBottom_toTopOf располагаем один элемент поверх другого.

Чтобы настроить положение элементов внутри цепочки, применяется атрибут layout_constraintVertical_chainStyle, который может принимать следующие значения:

- spread: значение по умолчанию, при котором элементы цепочки равномерно растягиваются по всей длине цепочки
- spread_inside: первый и последний элемент цепочки примыкают к границами контейнера
- packed: элементы цепочки прилегают вплотную друг к другу.

Например, вертикальная цепочка со значением по умолчанию - **spread**:

```
<?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:id="@+id/textView1"
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:background="#efefef"
        android:text="First"
        android:textSize="30sp"
```

```
    app:layout_constraintBottom_toTopOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintTop_toBottomOf="@id/textView1"
    app:layout_constraintBottom_toTopOf="@id/textView3"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

<TextView
    android:id="@+id/textView3"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="Third"
    android:textSize="30sp"

    app:layout_constraintTop_toBottomOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toBottomOf="parent" />
```

</androidx.constraintlayout.widget.ConstraintLayout>

Вертикальный ряд элементов в ConstraintLayout в Android

Также достаточно применить к первому элементу цепочки атрибут layout_constraintVertical_chainStyle, чтобы изменить положение элементов:

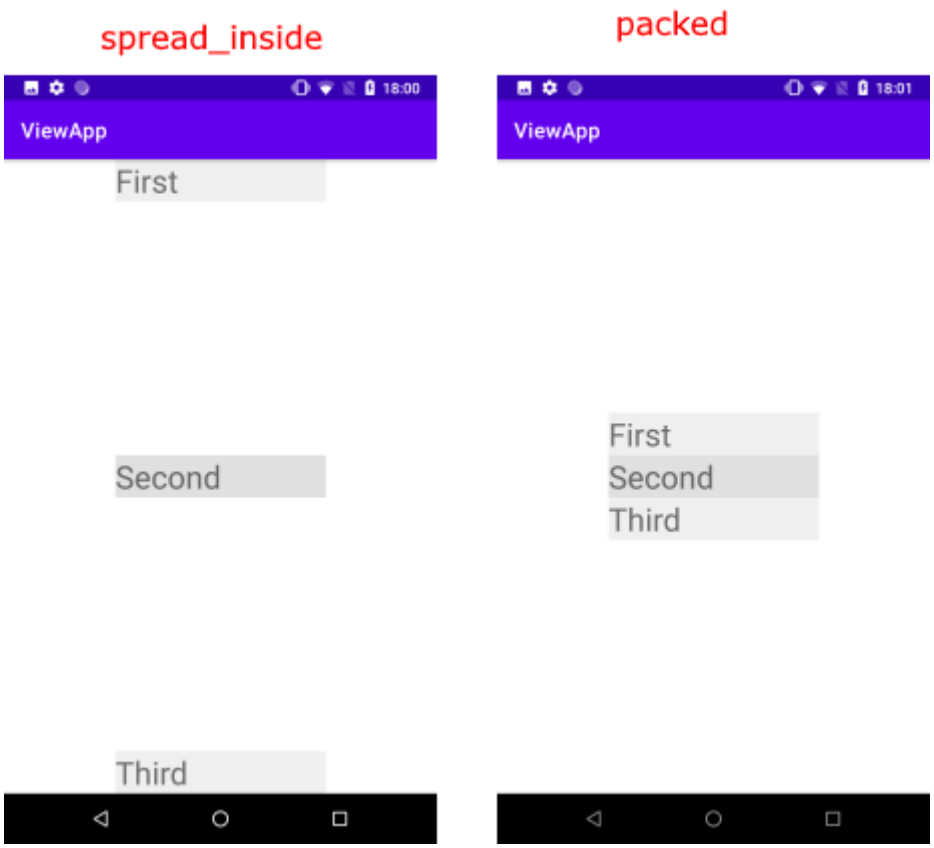
```
<TextView
    android:id="@+id/textView1"
    android:layout_width="200dp"
    android:layout_height="wrap_content"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"

    app:layout_constraintVertical_chainStyle="spread_inside"
```

app:layout_constraintBottom_toTopOf="@id/textView2"

app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toTopOf="parent" />



layout_constraintVertical_chainStyle in ConstraintLayout in Android

И как при горизонтальной ориентации в вертикальной цепочке можно использовать вес элементов с помощью атрибута layout_constraintVertical_weight. Для установки веса у элемента в качестве высоты должно быть установлено значение 0dp

<?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"

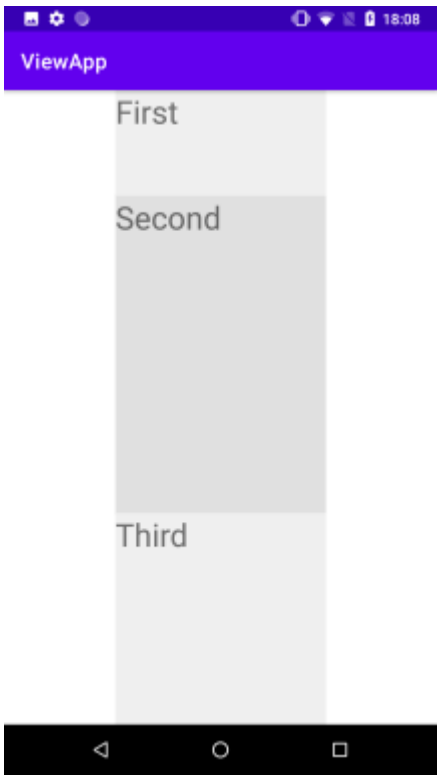
android:layout_width="match_parent"

android:layout_height="match_parent"

```
tools:context=".MainActivity">

<TextView
    android:id="@+id/textView1"
    android:layout_width="200dp"
    android:layout_height="0dp"
    android:background="#efefef"
    android:text="First"
    android:textSize="30sp"
    app:layout_constraintVertical_weight="1"
    app:layout_constraintBottom_toTopOf="@id/textView2"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="200dp"
    android:layout_height="0dp"
    android:background="#e0e0e0"
    android:text="Second"
    android:textSize="30sp"
    app:layout_constraintVertical_weight="3"
    app:layout_constraintTop_toBottomOf="@id/textView1"
    app:layout_constraintBottom_toTopOf="@id/textView3"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

<TextView
```

```
android:id="@+id/textView3"
android:layout_width="200dp"
android:layout_height="0dp"
android:background="#efefef"
android:text="Third"
android:textSize="30sp"
app:layout_constraintVertical_weight="2"
app:layout_constraintTop_toBottomOf="@id/textView2"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintBottom_toBottomOf="parent" />

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

Совокупный вес элементов в данном случае 1 + 3 + 2 = 6. Поэтому вся высота контейнера будет делиться на 6 частей, из которых первый элемент займет 1 часть, второй - 3 части и третий - 2 части в соответствии со своим весом.



layout_constraintVertical_weight in ConstraintLayout in Android

Программное создание ConstraintLayout и позиционионирование.

Для создания контейнера в коде Java применяется одноименный класс ConstraintLayout, для создания объекта которого в конструктор передаются значения для ширины и высоты элемента:

```
ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams

        (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
```

Первый параметр устанавливает щирину элемента, а второй - высоту. ConstraintLayout.LayoutParams.WRAP_CONTENT указывает, что элемент будет иметь те размеры, которые необходимы для того, чтобы вывести на экран его содержимое. Кроме ConstraintLayout.LayoutParams.WRAP_CONTENT можно применять константу ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, которая аналогична применению значения "0dp" в атрибутах layout_width и layout_height и которая растягивает элемент по ширине или высоте контейнера.

Также можно использовать точные размеры, например:

```
ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams

        (ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, 200);
```

Для настройки позиционирования внутри ConstraintLayout применяется класс ConstraintLayout.LayoutParams. Он имеет довольно много функционала. Рассмотрим в данном случае только те поля, которые позволяют установить расположение элемента:

- baselineToBaseline: выравнивает базовую линию элемента по базовой линии другого элемента, id которого присваивается свойству.

- bottomToBottom: выравнивает нижнюю границу элемента по нижней границе другого элемента.

- bottomToTop: выравнивает нижнюю границу элемента по верхней границе другого элемента.

- leftToLeft: выравнивает левую границу элемента по левой границе другого элемента.

- leftToRight: выравнивает левую границу элемента по правой границе другого элемента.

- rightToLeft: выравнивает правую границу элемента по левой границе другого элемента.

- rightToRight: выравнивает правую границу элемента по правой границе другого элемента.

- startToEnd: выравнивает начало элемента по завершению другого элемента.

- startToStart: выравнивает начало элемента по началу другого элемента.

- topToBottom: выравнивает верхнюю границу элемента по нижней границе другого элемента.

- topToTop: выравнивает верхнюю границу элемента по верхней границе другого элемента.

- endToEnd: выравнивает заврешение элемента по завершению другого элемента.

- endToStart: выравнивает завершение элемента по началу другого элемента.

В качестве значения эти поля принимают id (идентификатор) элемента, относительно которого выполняется позиционирование. Если расположение устанавливается относительно контейнера ConstraintLayout, то применяется константа **ConstraintLayout.LayoutParams.PARENT_ID**

Рассмотрим простейший пример:

```
package com.example.viewapp;
 import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

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

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        TextView textView = new TextView(this);
        // установка текста текстового поля
        textView.setText("Hello Android");
        // установка размера текста
        textView.setTextSize(30);

        ConstraintLayout.LayoutParams layoutParams = new
ConstraintLayout.LayoutParams
```

```
            (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        // позиционирование в левом верхнем углу контейнера
        // эквивалент app:layout_constraintLeft_toLeftOf="parent"
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        // эквивалент app:layout_constraintTop_toTopOf="parent"
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        // устанавливаем размеры
        textView.setLayoutParams(layoutParams);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```

В данном случае значение ConstraintLayout.LayoutParams.WRAP_CONTENT для ширины и высоты указывает, что элемент будет иметь те размеры, которые необходимы для того, чтобы вывести на экран его содержимое.

Далее выравниваем левую границу элемента по левой стороне контейнера:

```
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
```

Эта установка аналогична использованию атрибута app:layout_constraintLeft_toLeftOf="parent".

Затем выравниваем верхнюю границу элемента по верхней стороне контейнера:

```
layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
```

Эта установка аналогична использованию атрибута app:layout_constraintTop_toTopOf="parent".

И в конце применяем объект ConstraintLayout.LayoutParams к TextView:

constraintLayout.addView(textView);

В итоге элемент TextView будет расположен в верхнем левом углу ConstraintLayout:

Расположение элементов в ConstraintLayout в Android Studio

Рассмотрим другой пример - установку расположения элементов относительно друг друга:

```
package com.example.viewapp;

 import androidx.appcompat.app.AppCompatActivity;

import androidx.constraintlayout.widget.ConstraintLayout;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;
```

```
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

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

        ConstraintLayout constraintLayout = new ConstraintLayout(this);

        EditText editText = new EditText(this);
        editText.setHint("Введите Email");
        editText.setId(View.generateViewId());

        Button button = new Button(this);
        button.setText("Отправить");
        button.setId(View.generateViewId());

        ConstraintLayout.LayoutParams editTextLayout = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        editTextLayout.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        editTextLayout.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        editTextLayout.rightToLeft = button.getId();
        editText.setLayoutParams(editTextLayout);
        constraintLayout.addView(editText);
```

```
        ConstraintLayout.LayoutParams buttonLayout = new
ConstraintLayout.LayoutParams
                (ConstraintLayout.LayoutParams.WRAP_CONTENT ,
ConstraintLayout.LayoutParams.WRAP_CONTENT);
        buttonLayout.leftToRight = editText.getId();
        buttonLayout.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        button.setLayoutParams(buttonLayout);
        constraintLayout.addView(button);


        setContentView(constraintLayout);
    }
}
```

При расположении одного элемента относительно другого, нам нужно знать id вторрого элемента. Если элемент определен в коде Java, то вначале надо сгенерировать идентификатор:

```
editText.setId(View.generateViewId());
button.setId(View.generateViewId());
```

Затем можно применять идентификаторы элементов для установки позиционирование позиционионирование. Так, правая граница EditText выравнивается по левой границе кнопки:

```
editTextLayout.rightToLeft = button.getId();
```

А левая граница кнопки выравнивается по правой границе элемента EditText:

```
buttonLayout.leftToRight = editText.getId();
```

Позиционирование в ConstraintLayout в Android

Задание

1. Изучите порядок выполнения приложения в проекте Android Studio, создание и применение графического интерфейса. Режим кода и графический режим. Файл activity_main.xml. Определение пространств имен XML. Текстовое поле TextView
2. Для работы с визуальными элементами создать новый проект. В качестве шаблона проекта выбераем **Empty Activity**
3. Создать интерфейс в коде java. (создать элементы управления программно в коде java).
4. Объявить элементы интерфейса в XML. Определить интерфейс в файле XML **activity_main.xml**
5. Добавить в проект новый файл разметки интерфейса.
6. Реализовать получение и управление визуальными элементами в коде.
7. Определить размеры экрана вашего устройства на Android.
8. Установить размеры в коде Java.

9. Установить ширину и высоту визуальных элементов в XML (match_parent, wrap_content)
10. Реализовать программную установку ширины и высоты
11. Реализовать внутренние и внешние отступы в XML(Padding, Margin)
12. Реализовать программную установку внутренних и внешних отступов
13. Реализуйте примеры применения контейнера ConstraintLayout. (позиционирование,сдвиг, горизонтальная цепочка элементов, вес элемента, горизонтальный ряд виджетов, вертикальная цепочка, программное создание ConstraintLayout)