

Курс по Android разработке

Яборов Андрей Владимирович

avyaborov@gravity-group.ru

Марквирер Владлена Дмитриевна

vdmarkvirer@hse.ru

НИУ ВШЭ 2022



Разработка под Android

Урок 2

- Синтаксис языка
- Основные компоненты Android. Обзор Activity и Fragment
- Demo проект
- Base проект



Синтаксис языка

Объектно-ориентированность в языке Java выражена аналогично, как и в языке C#. Т.е. мы можем создавать иерархию классов, внутри которых можно создавать поля, конструкторы, методы и т.п. причём синтаксис будет идентичен.

В языке Java имеются те же самые операции над числами, строками, что и в том же C#, а также они обозначены теми же символами. Приоритет действий тоже остаётся без изменений. Имеются укороченные формы записи операций (например, “+=” или “/=”), а также операции инкремента и декремента.

Логические операторы и операции такие же, как и в C#.

Синтаксис языка

Java-программа - совокупность объектов, которые вызывают друг друга.

Ключевые элементы **Java-программы**:

- **Класс** – сущность, которая характеризуется полями (атрибутами) и поведением (методами).
- **Объект** – это экземпляр класса.
- **Метод** – это сущность, которая описывает поведение класса.
- **Переменная** – это значение, которое характеризует поле (атрибут) класса.

```
public class MyFirstProgram{  
    public static void main(String[] args){  
        System.out.println("My first Java program.");  
    }  
}
```

<- Java-программа

**Java-программа
для Android ->**

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Log.d("message", "My first Android app.");  
    }  
}
```

Синтаксис языка

Идентификаторы

Все компоненты программы должны иметь имя.

- Все идентификаторы должны начинаться с букв (A – Z или a – z), знака доллара (\$) или подчеркивания (_)
- После первого символа идентификатор может иметь любую последовательность символов.
- В качестве идентификатора не могут использоваться ключевые слова
- Все идентификаторы чувствительны к регистру
- Пример неправильного идентификатора:
123Developer, +first, int
- Пример правильного идентификатора:
developer, \$second, _third

Переменные

- локальными
- статическими
- нестатическими

Модификаторы

- модификаторы доступа: *default, public, protected, private*
- другие модификаторы: *final, abstract, strictfp*

Синтаксис языка

Перечисления

В простейшей форме перечисление - это список именованных констант. Но в Java перечисления имеют более сложный функционал, чем в других языках программирования. Они могут иметь конструкторы, методы и переменные экземпляра.

Использование перечисляемых переменных позволят избежать ошибок. Например, мы хотим использовать только числа 1, 2, 3, такой способ не позволит использовать числа 0, 5, 9 и т.д.

```
// занимает 1112 байт
public static enum Things {
    THING_1,
    THING_2
};

// другой вариант, занимает 128 байт
public static int THING_1 = 0;
public static int THING_2 = 1;
```

Массивы

Массивы – это объекты, которые хранят несколько переменных одного типа. Массив является объектом и хранится в heap.

Комментарии

Язык Java поддерживает комментарии. Комментарии - это фрагменты кода, которые не обрабатываются компилятором и являются информацией для объяснения необходимости того или иного фрагмента кода.

```
/* Многострочный
   комментарий */
//!!!!This is comments!!!
```

Синтаксис языка

Ключевые слова - слова, которые нельзя использовать в качестве идентификаторов.

Keywords in Java				
abstract	default	if	private	this
assert	do	implements	protected	throw
boolean	double	import	public	throws
break	else	instanceof	return	transient
byte	enum	int	short	try
case	extends	interface	static	void
catch	final	long	strictfp	volatile
char	finally	native	super	while
class	float	new	switch	
continue	for	package	synchronized	

- + goto, const (хотя и не используются в языке Java)
- + true, false и null

Синтаксис языка

Примитивные типы данных

byte	short	int	long	float	double	boolean	char
<p>8-битное целое число Мин. зн.: -128 Макс. зн.: 127</p> <p>Значение по умолчанию: 0 Используется для экономии места в больших массивах. Чаще всего вместо int.</p> <p>Пример: byte c = 65</p>	<p>16-битное целое число Мин. зн.: -32768 Макс. зн.: 32767</p> <p>Значение по умолчанию: 0 Используется для экономии места вместо int.</p> <p>Пример: short a = 2000 short b = -1000</p>	<p>32-битное целое число Мин. зн.: -2147483648 Макс. зн.: 2147483647</p> <p>Значение по умолчанию: 0 Используется для целых значений в случае, если нет дефицита памяти.</p> <p>Пример: int i = 200000 int h = -150000</p>	<p>64-битное целое число Мин. зн.: -2^{63} Макс. зн.: $2^{63}-1$</p> <p>Значение по умолч.: 0L Используется для хранения больших целочисл. значений.</p> <p>Пример: long l = 5000000000L long k = -4000000000L</p>	<p>32-битное число с плавающей точкой</p> <p>Значение по умолч.: 0.0f Используется для экономии памяти в больших массивах чисел с плавающей точкой. Никогда не используется для хранения точных значений (например, денег).</p> <p>Пример: float f = 112.3f</p>	<p>64-битное число двойной точности с плавающей точкой</p> <p>Значение по умолч.: 0.0d Используется для хранения чисел с плавающей точкой (в большинстве случаев). Никогда не используется для хранения точных значений (например, денег).</p> <p>Пример: double d = 21.5</p>	<p>В спецификации размер не указан. Зависит от типа JVM. Возможные значения: true/false</p> <p>Значение по умолчанию: false Используется для определения того, является ли условие истинным.</p> <p>Пример: boolean flag = true</p>	<p>Символ кодировки Unicode 16-bit Мин. зн.: 'u0000' (или 0) Макс. зн.: 'uffff' (или 65535) Используется для хранения любого символа</p> <p>Пример: char c = 'C'</p>

Синтаксис языка

Ссылочные типы данных

- К ссылочным типам данных относятся все типы данных, которые создаются с помощью конструкторов. К ним также относятся все классы, создаваемые разработчиками, например, Developer, Car, Person и т.д.
- Массивы являются ссылочными типами данных.
- Ссылочная переменная может использоваться в качестве ссылки на любой объект определённого типа данных.
- Все ссылочные типы имеют значение по умолчанию: null.
- Пример: Developer developer = new Developer("Java Developer");

Литералы

Литералы - это представление фиксированных значений в виде кода. Они не требуют каких-либо вычислений. Литералы могут быть целочисленными, с плавающей точкой,, символьными, строковыми и булевыми.

```
char c = 'C';

// для int, long, short, byte:
int decimal = 500;
int octal = 0168;
int hexa = 0x32;

// String может содержать простые символы,
// а также Unicode
char c = '\uffff';
String str = "\uffff";
```

Управляющие последовательности:

\n	Перевод на новую строку (0x0a)
\r	Возврат каретки к началу строки (0x0d)
\f	Перевод на новую страницу (0x0c)
\b	Возврат на один символ назад (0x08)
\t	Табуляция
\"	Двойная кавычка
'	Одинарная кавычка
\\	Обратный слэш
\xxx	Восьмеричный символ (xxx)
\uxxxx	Шестнадцатиричный символ UNICODE (xxxx)

Синтаксис языка

Объявление переменных

Переменная предоставляется нам именем хранения, чтобы нашей программой можно было манипулировать. Каждая переменная в Java имеет конкретный тип, который определяет размер и размещение её в памяти; диапазон значений, которые могут храниться в памяти; и набор операций, которые могут быть применены к переменной.

Java: `<тип данных> <имя переменной> [= значение], [переменная [= значение], ...] ;`

Kotlin: `var/val <имя переменной>: [тип данных] [= значение]`

```
// Java
final String name = "Васька";

// Kotlin
val name = "Васька"
```

```
// Java
public static final String CAT_TALK = "meow";

// Kotlin
const val CAT_TALK = "meow"
```

```
// Java
String name = "Васька";
name = name + " - кот";

// Kotlin
var name: String = "Васька"
name = name + " - кот"
```

Синтаксис языка

Условные операторы

Условный оператор if часто применяется программистами и имеется во всех языках программирования. Оператор if позволяет вашей программе в зависимости от условий выполнить оператор или группу операторов, основываясь на значении булевой переменной или выражения. Оператор if является основным оператором выбора в Java и позволяет выборочно изменять ход выполнения программы - и это одно из основных отличий между программированием и простым вычислением.

```
if (условие)
{
    оператор1;
    оператор2;
}
else
{
    оператор1;
    оператор2;
}
```

Синтаксис языка

Оператор выбора

В отличие от операторов if-then и if-then-else, оператор switch применим к известному числу возможных ситуаций.

Можно использовать простые типы byte, short, char, int. Также можно использовать Enum и String (начиная с JDK7), и специальные классы, которые являются обёрткой для примитивных типов: Character, Byte, Short, Integer.

Дублирование значений case не допускается. Тип каждого значения должен быть совместим с типом выражения.

Java

```
switch (<выражение>)  
{  
    case <значение1>: [оператор1;] break;  
    case <значение2>: [оператор2;] break;  
    case <значение3>: [оператор3;] break;  
    [default: [оператор4;] break;  
}
```

Kotlin

```
when (<выражение>)  
{  
    <значение1> -> [оператор1]  
    <значение2> -> [оператор2]  
    <значение3> -> [оператор3]  
    [else -> [оператор4]]  
}
```

Синтаксис языка

Циклы

Выполнение операции или блока операций некоторое количество раз, в зависимости от условия.

while - выполняет выражение или группу выражений до тех пор, пока указанное условие истинно (true). Если условие больше не выполняется (false), то программа выходит из цикла (выполнение цикла прекращается).

do while - выполняет выражение или группу выражений до тех пор, пока указанное условие истинно (true). Если условие больше не выполняется (false), то программа выходит из цикла (выполнение цикла прекращается). Главное отличие от цикла while заключается в том, что проверка истинности условия находится в конце цикла, а это означает, что цикл будет выполнен минимум один раз, независимо от истинности условия.

for - выполняет выражение или группу выражений несколько раз и сокращает количество кода, необходимого для управления цикла.

Java~Kotlin

```
int i = 0;
int sum = 0;
while (i < 10){
    sum += i;
    i++;
}
```

```
int i = 0;
int sum = 0;
do {
    sum += i;
    i++;
}while (i < 10);
```

Java

```
int sum = 0;
for (int i = 0; i < 10; i++)
    sum += i;
String[] strings = {"1 курс",
                    "2 курс", "3 курс"};
for (String str : strings)
    Log.d("Курсы", str);
```

Kotlin

```
var sum : Int = 0
for (i in 1 to 10) // (i in 1..10)
    sum += i
val strings = listOf("1 курс", "2 курс",
                    "3 курс")
strings.forEach{
    Log.d("Курсы", it)
}
repeat(3){
    println("Спать!")
}
```

Синтаксис языка

Исключения

Используя комбинацию ключевых слов `try` и `catch` мы имеем возможность “ловить” исключения, которые возникают во время работы нашей программы. Внутри блока `try/catch` размещается код, который может вызвать исключение.

Другими словами, мы “пытаемся” (`try`) выполнить кусок кода и “ловим” (`catch`) исключения, которые могут возникнуть.

```
try{  
    //Code that can call an exception  
} catch ([<тип исключения> <имя переменной для исключения>]) {  
    //Sequence of actions in case of exception  
} finally {  
    //Sequence of actions required in any cases  
}
```

// в Kotlin
// блок try может использоваться как выражение
[<имя переменной для исключения> : <тип исключения>]

Синтаксис языка

Функции / методы

Метод в Java — это комплекс выражений, совокупность которых позволяет выполнить определенную операцию.

```
// Java
[модификатор] <тип возвращаемого значения> <имя метода>([список параметров]) {
    // тело метода
}
// Kotlin
fun <имя функции>([список параметров])[: тип возвращаемого значения] {
    // тело метода
}
```

Java

```
public static int add(int x, int y) {
    return x + y;
}
```

Kotlin

```
fun add(x: Int, y: Int): Int {
    return x + y
}
```

Синтаксис языка

<https://www.learnjavaonline.org/>

<https://kotlinlang.org/docs/tutorials/getting-started.html>

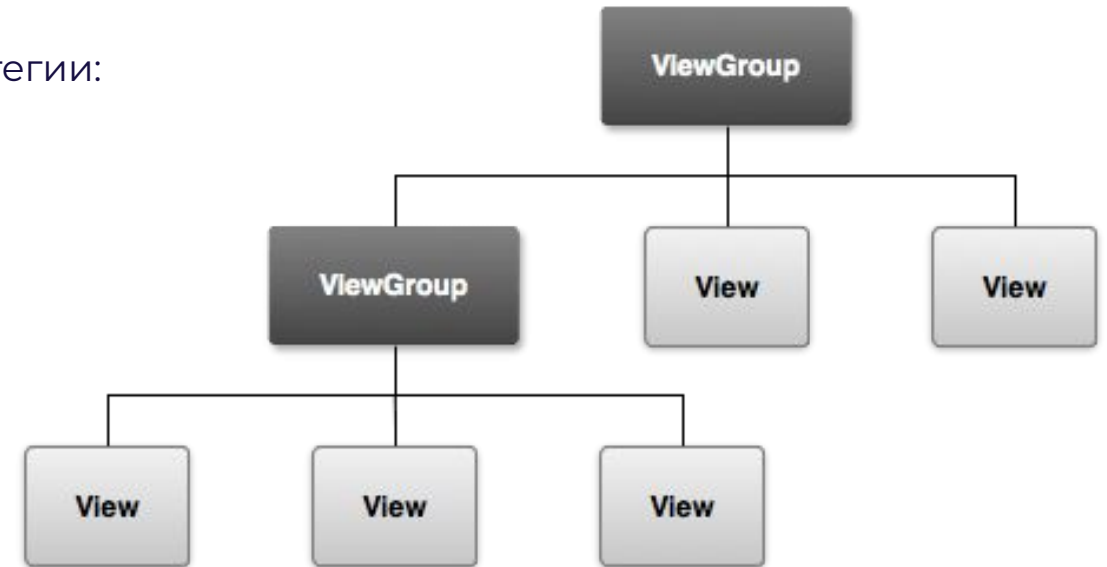
Синтаксис языка

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

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

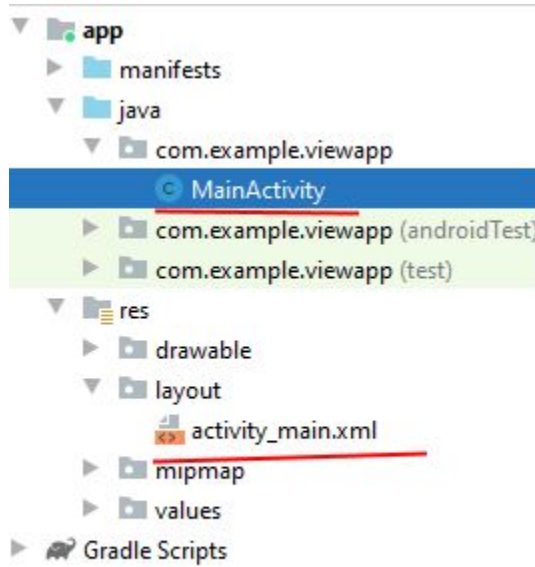
При определении визуального интерфейса у нас есть три стратегии:

- 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



Синтаксис языка

Программное создание элемента управления

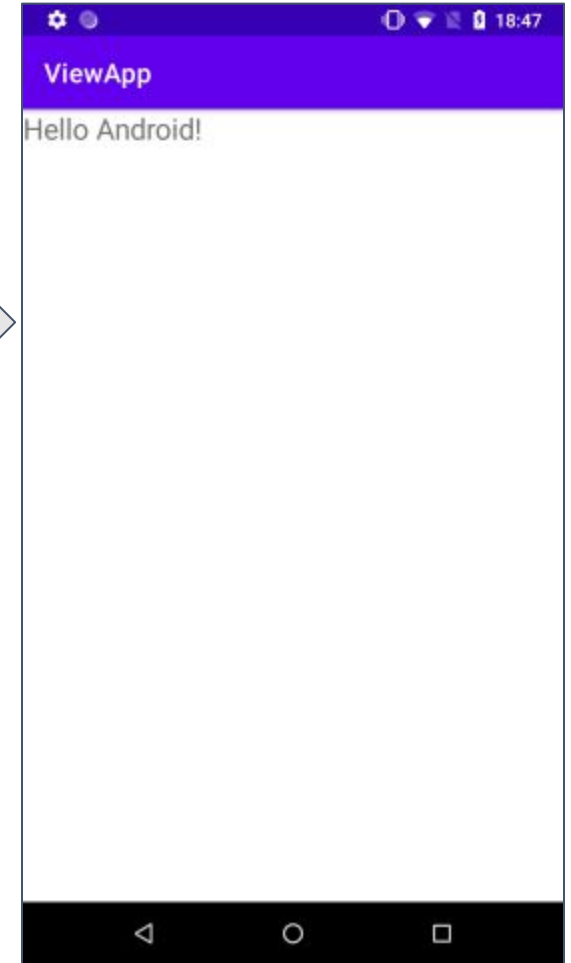


```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

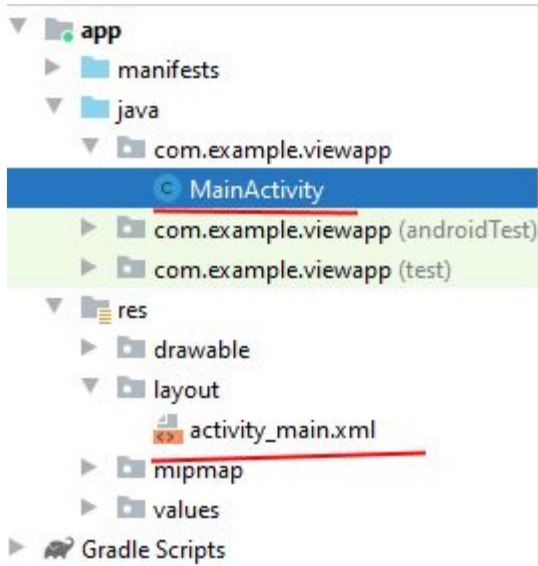
    @Override
    protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);

        // 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);
    }
}
```



Синтаксис языка

Создание элемента управления в XML

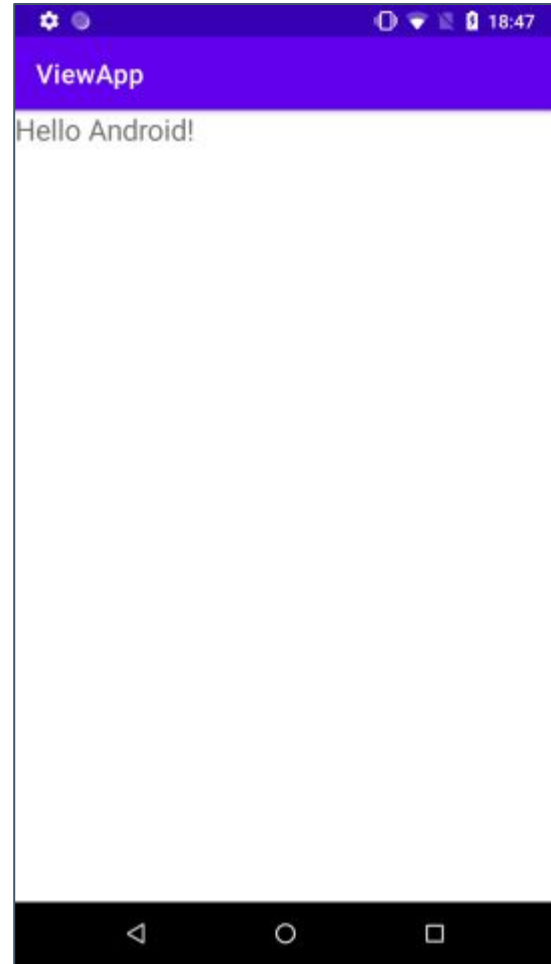


```
package com.example.viewapp;

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);
    }
}
```

```
<?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!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



Синтаксис языка

Простейшие элементы управления в Android-приложениях, общение с ними

Самые простые элементы управления:

- **TextView** - текстовое поле, отображает текст без возможности изменения пользователем напрямую.
- **Button** - кнопка, с помощью которой можно обрабатывать различные события/действия пользователя.
- **EditText** - поле ввода (в Android Studio есть несколько разновидностей для упрощения разработки, для простоты используем PlainText). В поле ввода пользователь может вводить данные, которые разрешит разработчик (буквы / цифры / все символы и т.п.), а также может реагировать на действия пользователя по каждому новому введённому символу, после смены фокуса и т.п. и передавать данные в обработку или передавать управление другому элементу.

Элементы размещаются на макете экрана в дизайнера или программно, настраиваются необходимые свойства. Для корректной работы элементов управления необходимо каждому элементу давать свои имена - идентификаторы (id).

XML: `android:id="@+id/editText1"`

В java файле элемент можно получить по id следующим образом:

```
EditText et = (EditText) findViewById(R.id.editText1);
```

Синтаксис языка

Простейшие элементы управления в Android-приложениях, общение с ними

Для того, чтобы задавать поведение элементам управления, необходимо работать с событиями, переопределять методы и пользоваться свойствами элементов управления.

Например, если есть некоторая кнопка, полученная с помощью метода `findViewById` и сохранённая в переменную с именем `button`, то для задания команд, которые будут выполняться в случае нажатия на кнопку, необходимо написать следующий шаблон:

Java

```
button.setOnClickListener(new  
View.OnClickListener() {  
    public void onClick(View v) {  
        // Обработка нажатия  
    }  
});
```

Kotlin

```
button.setOnClickListener{  
    // Обработка нажатия  
}
```

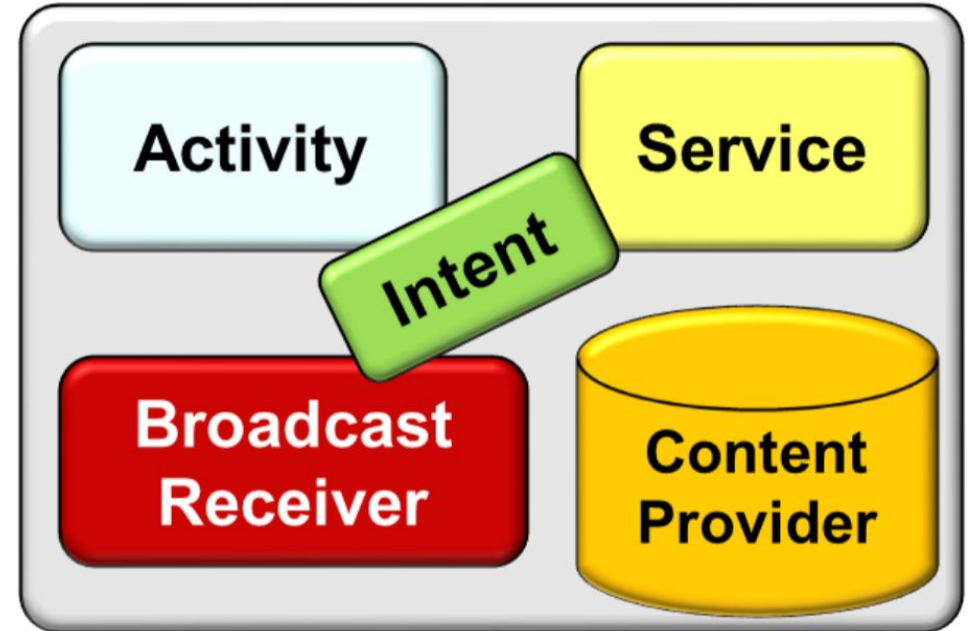
Для установки свойства в программе, необходимо также обратиться к объекту (элементу управления), выбрать необходимое свойство и установить необходимое значение, например:

```
textView.setText(et.getText());
```

Основные компоненты Android

Компоненты приложения являются кирпичиками, из которых состоит приложение для Android. Каждый компонент представляет собой отдельную точку, через которую система может войти в приложение. Не все компоненты являются точками входа для пользователя, а некоторые из них зависят друг от друга. При этом каждый компонент является самостоятельной структурной единицей и играет определенную роль — каждый из них представляет собой уникальный элемент структуры, который определяет работу приложения в целом.

Компоненты приложения можно отнести к одному из четырех типов. Компоненты каждого типа предназначены для определенной цели, они имеют собственный жизненный цикл, который определяет способ создания и прекращения существования компонента.



Основные компоненты Android

- Операция (**Activity**) представляет собой один экран с пользовательским интерфейсом. Например, в приложении для работы с электронной почтой одна операция может служить для отображения списка новых сообщений, другая – для составления сообщения и третья операция – для чтения сообщений.
- Служба (**Service**) представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов. Служба не имеет пользовательского интерфейса.
- Поставщик контента (**Content provider**) управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в Интернете или любом другом постоянном месте хранения, к которому у вашего приложения имеется доступ. Посредством поставщика контента другие приложения могут запрашивать или даже изменять данные (если поставщик контента позволяет делать это).

Основные компоненты Android

- Приемник широковещательных сообщений (**Broadcast receiver**) *представляет собой компонент, который реагирует на объявления распространяемые по всей системе.*
- Компоненты трех из четырех возможных типов — операции, службы и приемники широковещательных сообщений — активируются асинхронным сообщением, которое называется **Intent** (намерение). Объекты Intent связывают друг с другом отдельные компоненты во время выполнения, будь то это компоненты вашего или стороннего приложения

Activity

Ключевым компонентом для создания визуального интерфейса в приложении Android является activity (активность). Нередко activity ассоциируется с отдельным экраном или окном приложения, а переключение между окнами будет происходить как перемещение от одной activity к другой. Приложение может иметь одну или несколько activity. Например, класс MainActivity:

```
public class MainActivity extends AppCompatActivity {
```

```
    // содержимое класса
}
```

Все объекты activity представляют собой объекты класса **android.app.Activity**, которая содержит базовую функциональность для всех activity. Класс AppCompatActivity, хоть и не напрямую, наследуется от базового класса Activity.

Жизненный цикл

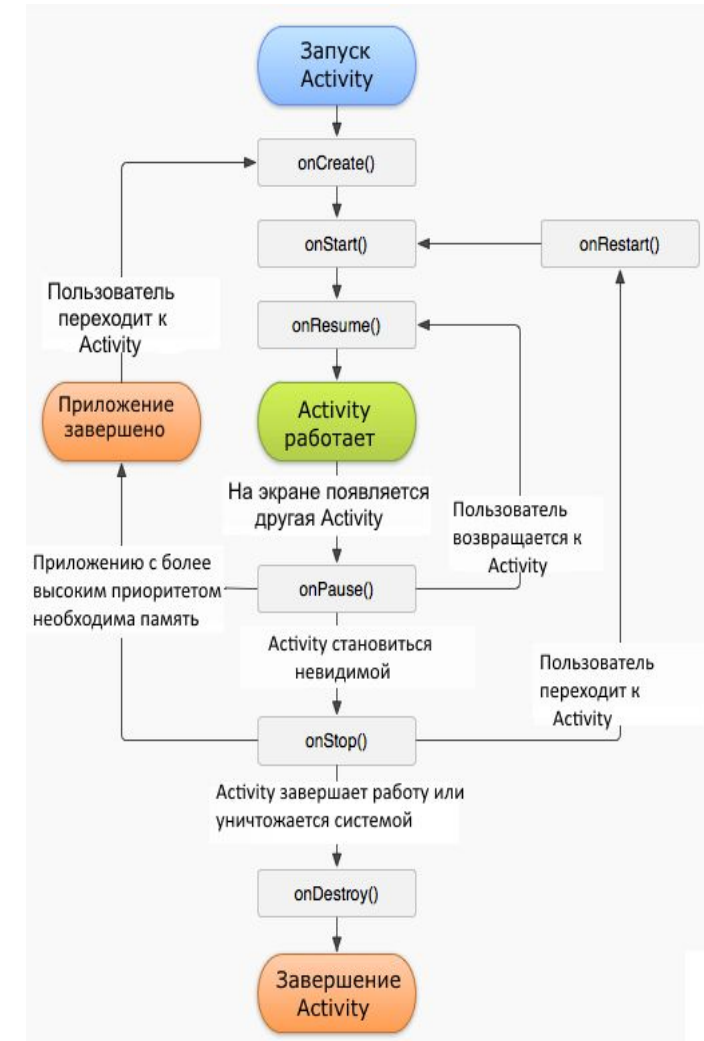
Все приложения Android имеют строго определенный системой жизненный цикл. При запуске пользователем приложения система дает этому приложению высокий приоритет. Каждое приложение запускается в виде отдельного процесса, что позволяет системе давать одним процессам более высокой приоритет, в отличие от других. Благодаря этому, например, при работе с одними приложениями не блокировать входящие звонки. После прекращения работы с приложением, система освобождает все связанные ресурсы и переводит приложение в разряд низкоприоритетного и закрывает его.

Activity

Все объекты activity, которые есть в приложении, управляются системой в виде стека activity, который называется **back stack**. При запуске новой activity она помещается поверх стека и выводится на экран устройства, пока не появится новая activity. Когда текущая activity заканчивает свою работу (например, пользователь уходит из приложения), то она удаляется из стека, и возобновляет работу та activity, которая ранее была второй в стеке.

onCreate - первый метод, с которого начинается выполнение activity. В этом методе activity переходит в состояние Created. Этот метод обязательно должен быть определен в классе activity. В нем производится первоначальная настройка activity. В частности, создаются объекты визуального интерфейса. Этот метод получает объект Bundle, который содержит прежнее состояние activity, если оно было сохранено. Если activity заново создается, то данный объект имеет значение null. Если же activity уже ранее была создана, но находилась в приостановленном состоянии, то bundle содержит связанную с activity информацию.

В методе **onStart()** осуществляется подготовка к выводу activity на экран устройства. Как правило, этот метод не требует переопределения, а всю работу производит встроенный код. После завершения работы метода activity отображается на экране, вызывается метод **onResume**, а activity переходит в состояние Resumed.



Activity

onRestoreInstanceState

После завершения метода `onStart()` вызывается метод `onRestoreInstanceState`, который призван восстанавливать сохраненное состояние из объекта `Bundle`, который передается в качестве параметра. Но следует учитывать, что этот метод вызывается только тогда, когда `Bundle` не равен **null** и содержит ранее сохраненное состояние. Так, при первом запуске приложения этот объект `Bundle` будет иметь значение `null`, поэтому и метод `onRestoreInstanceState` не будет вызываться.

onResume

А при вызове метода `onResume` `activity` переходит в состояние `Resumed`, а пользователь может с ней взаимодействовать. И собственно `activity` остается в этом состоянии, пока она не потеряет фокус, например, вследствие переключения на другую `activity` или просто из-за выключения экрана устройства.

onPause

Если пользователь решит перейти к другой `activity`, то система вызывает метод **onPause**. В этом методе можно освобождать используемые ресурсы, приостанавливать процессы, например, воспроизведение аудио, анимаций, останавливать работу камеры (если она используется) и т.д., чтобы они меньше сказывались на производительность системы.

Activity

onSaveInstanceState

Метод **onSaveInstanceState** вызывается после метода **onPause()**, но до вызова **onStop()**. В **onSaveInstanceState** производится сохранение состояния приложения в передаваемый в качестве параметра объект **Bundle**.

onStop

В этом методе **activity** переходит в состояние **Stopped**. В методе **onStop** следует освобождать используемые ресурсы, которые не нужны пользователю, когда он не взаимодействует с **activity**. Здесь также можно сохранять данные, например, в базу данных.

При этом во время состояния **Stopped** **activity** остается в памяти устройства, сохраняется состояние всех элементов интерфейса. К примеру, если в текстовое поле **EditText** был введен какой-то текст, то после возобновления работы **activity** и перехода ее в состояние **Resumed** мы вновь увидим в текстовом поле ранее введенный текст.

Если после вызова метода **onStop** пользователь решит вернуться к прежней **activity**, тогда система вызовет метод **onRestart**. Если же **activity** вовсе завершила свою работу, например, из-за закрытия приложения, то вызывается метод **onDestroy**.

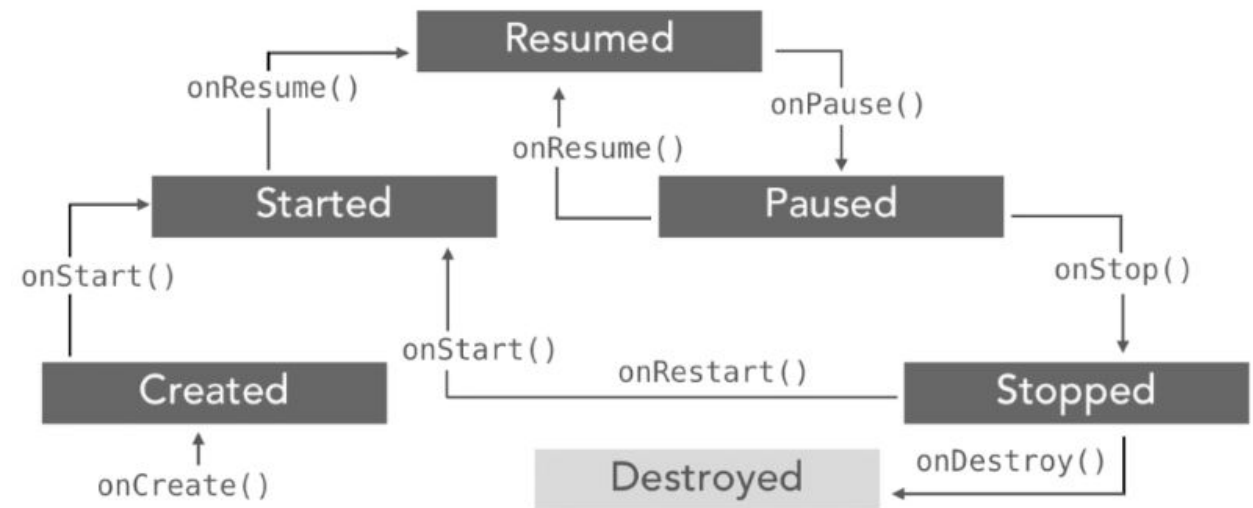
Activity

onDestroy

Ну и завершается работа активности вызовом метода **onDestroy**, который возникает либо, если система решит убить activity, либо при вызове метода **finish**.

Также следует отметить, что при изменении ориентации экрана система завершает activity и затем создает ее заново, вызывая метод **onCreate**.

В целом переход между состояниями activity можно выразить следующей схемой:



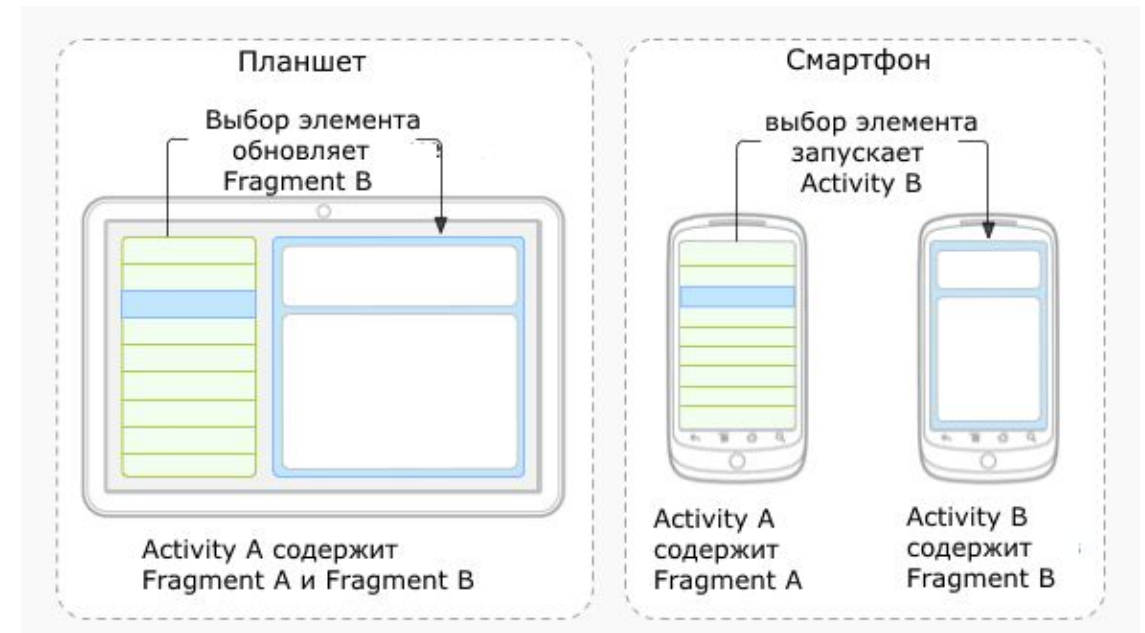
Fragment

Организация приложения на основе нескольких activity не всегда может быть оптимальной. Мир ОС Android довольно сильно фрагментирован и состоит из многих устройств. И если для мобильных аппаратов с небольшими экранами взаимодействие между разными activity выглядит довольно неплохо, то на больших экранах - планшетах, телевизорах окна activity смотрелись бы не очень в силу большого размера экрана. Собственно поэтому и появилась концепция фрагментов.

Фрагмент существует в контексте activity и имеет свой жизненный цикл, вне activity обособлено он существовать не может. Каждая activity может иметь несколько фрагментов.

Существует подклассы фрагментов:

ListFragment, DialogFragment, PreferenceFragment, WebViewFragment, MapFragment и др.



Fragment

Есть три основных класса:

android.app.Fragment — от него, собственно говоря, и будут наследоваться наши фрагменты

android.app.FragmentManager — с помощью экземпляра этого класса происходит все взаимодействие между фрагментами

android.app.FragmentTransaction — ну и этот класс, как понятно по названию, нужен для совершения транзакций.

Перед началом транзакции нужно получить экземпляр `FragmentManager` через метод `FragmentManager.beginTransaction()`. Далее вызываются различные методы для управления фрагментами.

В конце любой транзакции, которая может состоять из цепочки вышеперечисленных методов, следует вызвать метод `commit()`.

```
FragmentManager fragmentManager = getFragmentManager()

fragmentManager.beginTransaction()

    .remove(fragment1)

    .add(R.id.fragment_container, fragment2)

    .show(fragment3)

    .hide(fragment4)

    .commit();
```

Fragment

Для создания визуального интерфейса фрагмент переопределяет родительский метод `onCreateView()`. Он принимает три параметра:

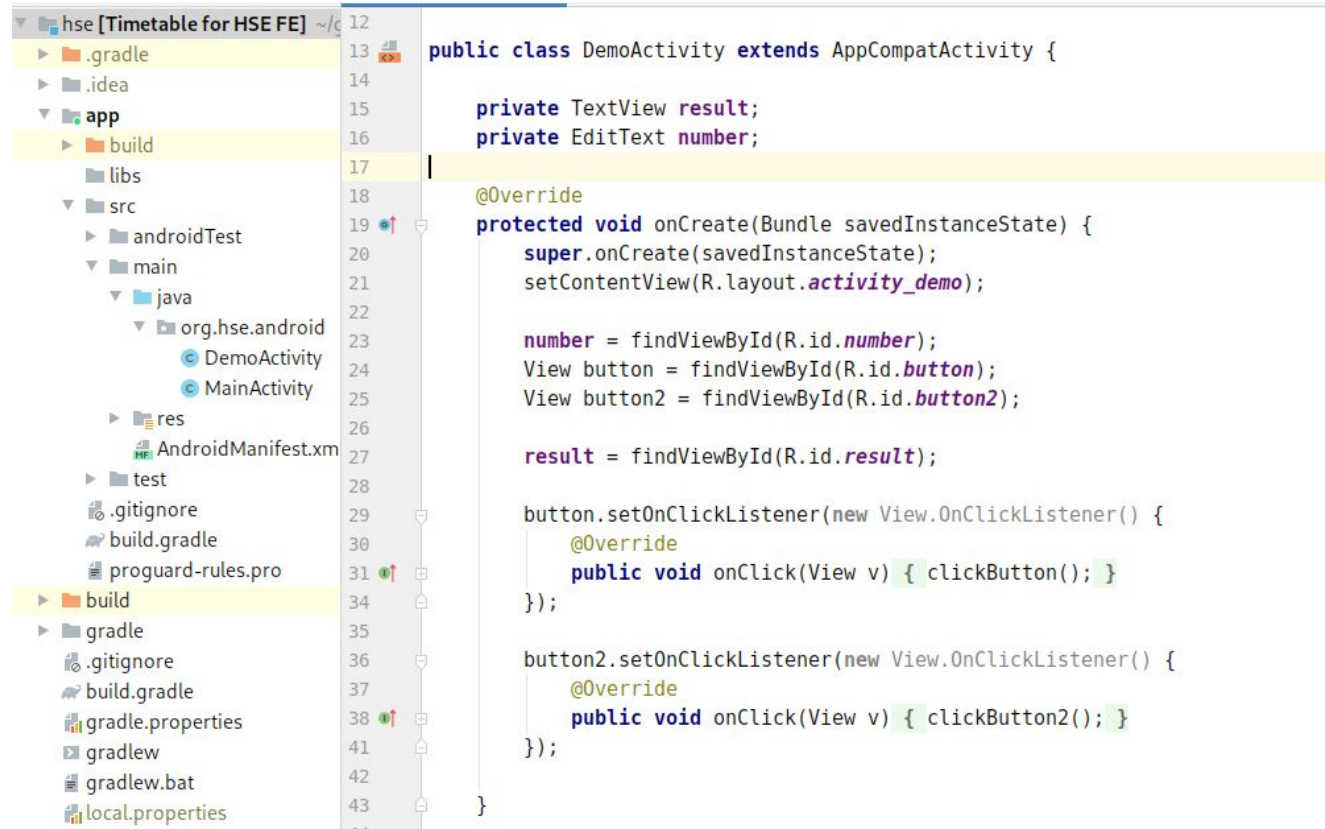
- Объект `LayoutInflater` используется для установки ресурса разметки для создания интерфейса
- Параметр `ViewGroup container` устанавливает контейнер интерфейса
- Параметр `Bundle savedInstanceState` передает ранее сохраненное состояние

Для создания интерфейса применяется метод `inflate()` объекта `LayoutInflater`. Он получает ресурс разметки `layout` для данного фрагмента, контейнер, в который будет заключен интерфейс, и третий булевый параметр указывает, надо ли прикреплять разметку к контейнеру из второго параметра.

```
public class ExampleFragment extends Fragment {  
  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        View view = inflater.inflate(R.layout.you_layout_for_fragment,  
                                     container, false);  
  
        return view;  
    }  
}
```

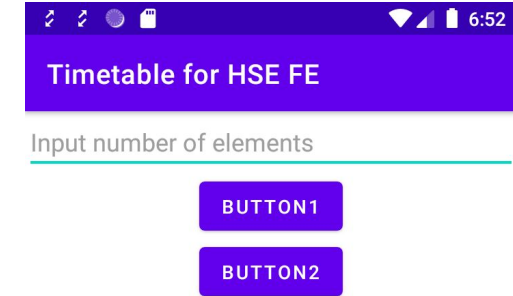
```
public class FragmentDemoActivity extends Activity {  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        // получаем экземпляр FragmentTransaction  
        FragmentManager fragmentManager = getFragmentManager();  
        FragmentTransaction fragmentTransaction = fragmentManager  
            .beginTransaction();  
  
        // добавляем фрагмент  
        MyFragment myFragment = new MyFragment();  
        fragmentTransaction.add(R.id.container, myFragment);  
        fragmentTransaction.commit();  
    }  
}
```


Demo проект



The screenshot shows an IDE with a project named "hse [Timetable for HSE FE]". The project structure on the left includes folders like .gradle, .idea, app, build, libs, src, androidTest, main, java, org.hse.android, res, test, .gitignore, build.gradle, and proguard-rules.pro. The main file, DemoActivity.java, is open in the editor. It is a Java class that extends AppCompatActivity. It has two private fields: TextView result and EditText number. The onCreate method is overridden, and it contains logic to find the result TextView and the number EditText, and to set click listeners for two buttons (button and button2) that call clickButton() and clickButton2() respectively.

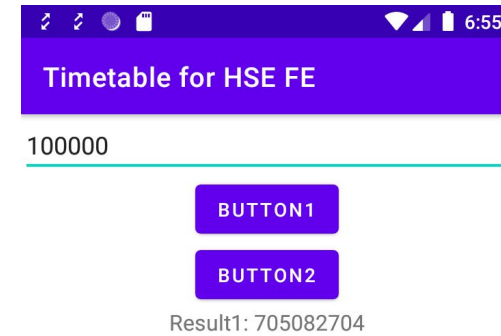
```
12 public class DemoActivity extends AppCompatActivity {
13
14     private TextView result;
15     private EditText number;
16
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_demo);
22
23         number = findViewById(R.id.number);
24         View button = findViewById(R.id.button);
25         View button2 = findViewById(R.id.button2);
26
27         result = findViewById(R.id.result);
28
29         button.setOnClickListener(new View.OnClickListener() {
30             @Override
31             public void onClick(View v) { clickButton(); }
32         });
33
34         button2.setOnClickListener(new View.OnClickListener() {
35             @Override
36             public void onClick(View v) { clickButton2(); }
37         });
38     }
39 }
```



Демо проект

Посчитаем все элементы в списке при нажатии на кнопку Button1

```
private void clickButton() {  
    String numberVal = number.getText().toString();  
    if (numberVal.isEmpty()) {  
        numberVal = "0";  
    }  
    int count = Integer.parseInt(numberVal);  
    // init list  
    List<Integer> list = new ArrayList<>();  
    for (int i = 0; i < count; i++) {  
        list.add(i + 1);  
    }  
  
    // Count all items in list  
    int sum = list.stream().mapToInt(Integer::intValue).sum();  
    result.setText(String.format("Result1: %d", sum));  
}
```



Демо проект

Задания

1. Ограничить ввод в поле для ввода только числовыми значениями
2. Добавить логику для нажатия на кнопку Button2. Считать произведение всех четных чисел (от 0 до вводимого значения) и вывести результат на экран по аналогии с кнопкой Button1
3. Добавить валидацию на диапазон вводимого значения, диапазон придумать самостоятельно. В случае выхода за диапазон выводить Toast с текстом о выходе за диапазон

Base проект

Создадим первый экран приложения

1. Файл с логотипом скачать по ссылке
https://www.hse.ru/data/2012/01/19/1263884289/logo_%D1%81_hse_cmyk_e.png
2. Поместить скачанный файл в папку **drawable-xxxhdpi**
3. Сделать верстку экрана как на картинке справа. Создать новую activity и файл с версткой для нее **activity_main.xml**. Запускать по умолчанию данную activity
4. Основные цвета приложения указать в файле стилей **themes.xml**
`colorPrimary #003399`
`colorPrimaryVariant #011F5A`
`colorSecondary #4080ff`
`colorSecondaryVariant #0C5DFD`
5. Тексты для кнопок вынести в файл **strings.xml**
6. Добавить показ Toast при нажатии на кнопки с информацией о нажатой кнопке



Литература

<https://developer.android.com/guide/components/activities/intro-activities>

<https://developer.android.com/guide/components/intents-filters>

<https://developer.android.com/guide/components/services>

<https://developer.android.com/guide/topics/ui/declaring-layout>

<https://material.io/develop/android/theming/color>