

Основы создания интерфейса

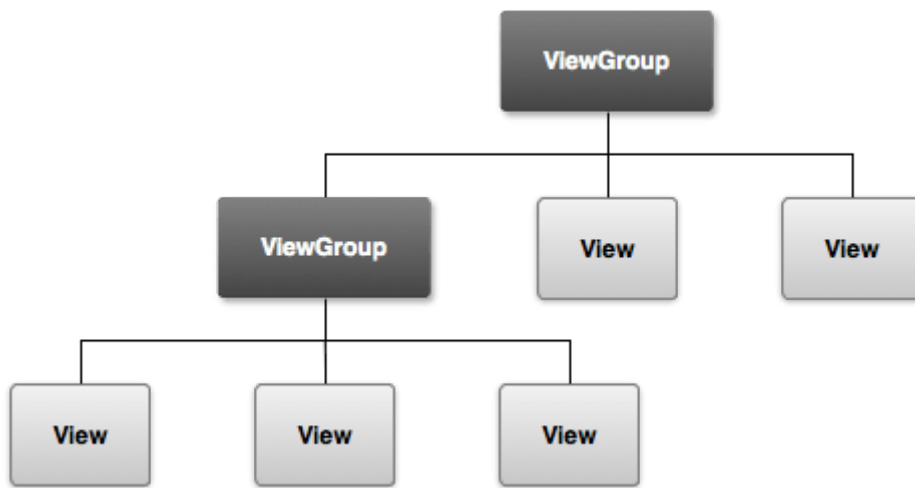
Введение в создание интерфейса

Последнее обновление:



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

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



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

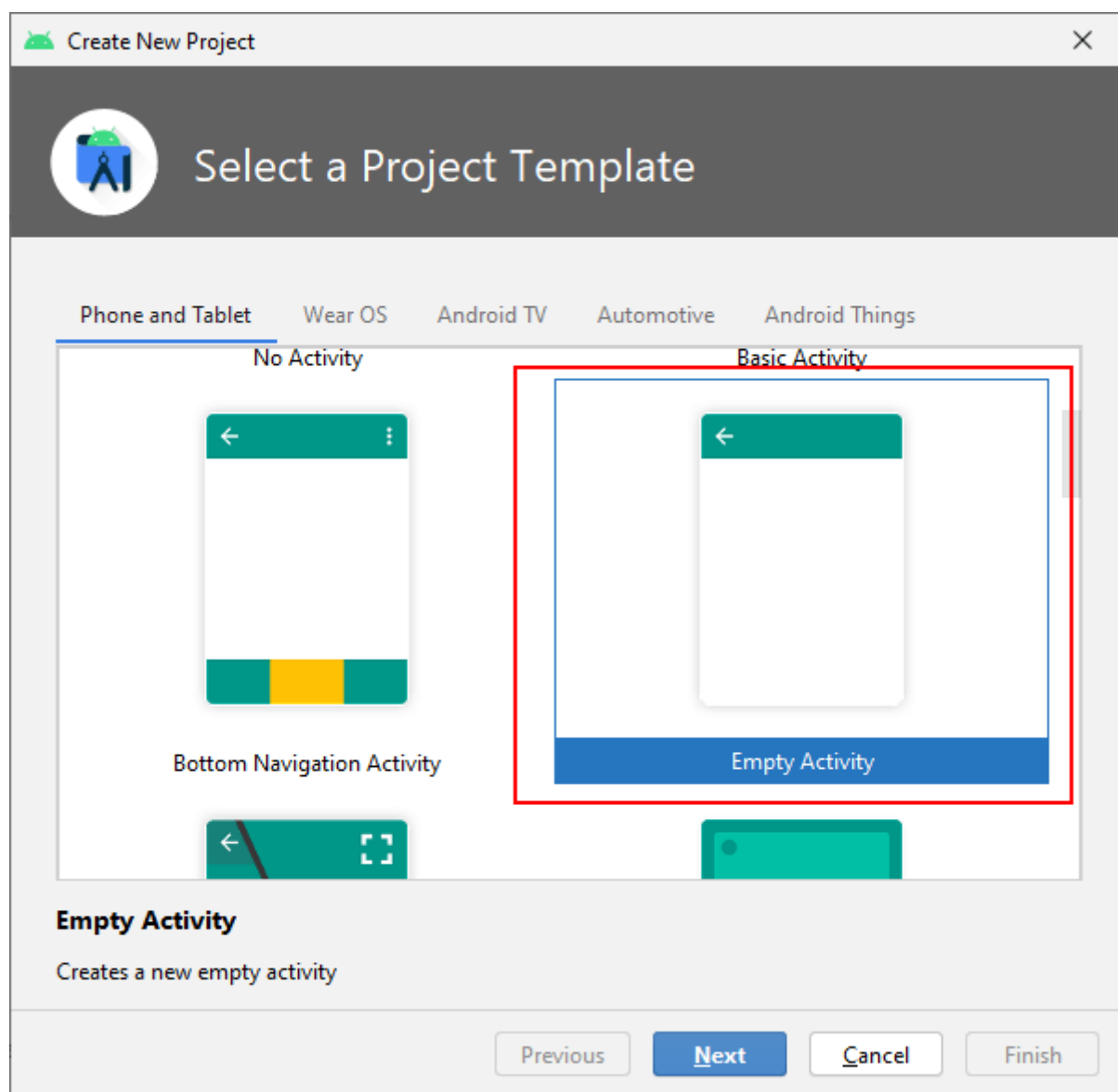
При определении визуального у нас есть три стратегии:

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

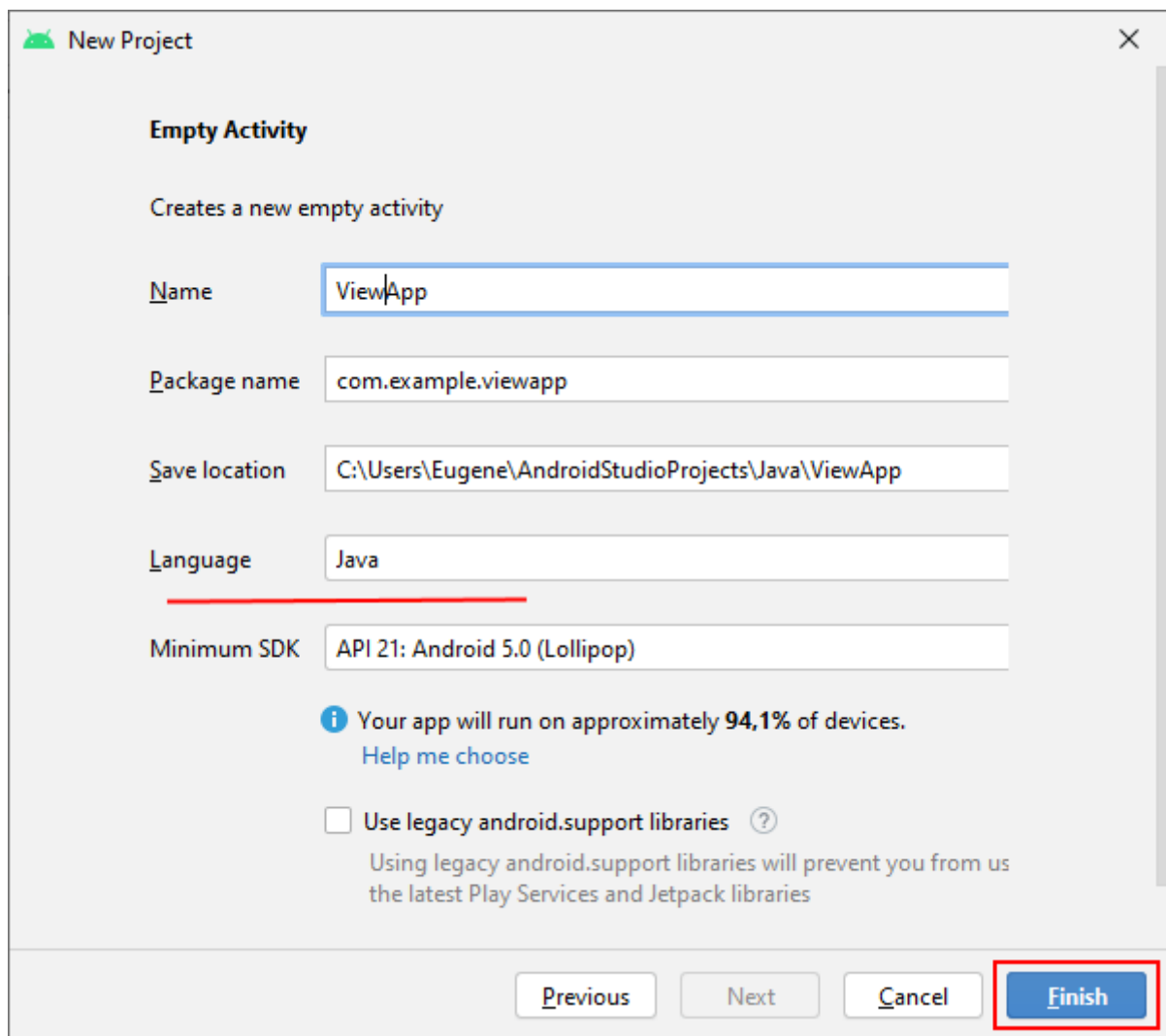
Сначала рассмотрим первую стратегию - определение интерфейса в коде `Java`.

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:



New Project

Empty Activity

Creates a new empty activity

Name: ViewApp

Package name: com.example.viewapp

Save location: C:\Users\Eugene\AndroidStudioProjects\Java\ViewApp

Language: Java

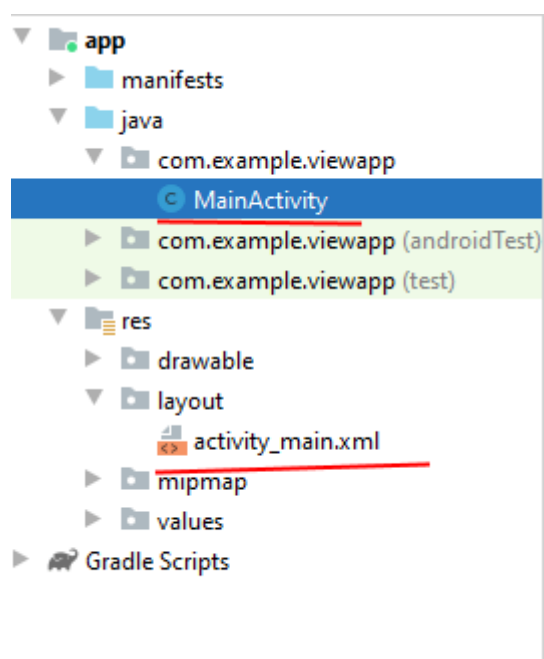
Minimum SDK: API 21: Android 5.0 (Lollipop)

i Your app will run on approximately **94,1%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries **?**
Using legacy android.support libraries will prevent you from using the latest Play Services and Jetpack libraries

[Previous](#) [Next](#) [Cancel](#) **Finish**

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:

```

1 package com.example.viewapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.widget.TextView;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12
13        // создание TextView
14        TextView textView = new TextView(this);
15        // установка текста в TextView
16        textView.setText("Hello Android!");
17        // установка высоты текста
18        textView.setTextSize(22);
19        // установка визуального интерфейса для activity
20        setContentView(textView);
21    }
22 }

```

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.

```

1 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`:

```
1 package com.example.viewapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4 import androidx.constraintlayout.widget.ConstraintLayout;
5
6 import android.os.Bundle;
7 import android.widget.TextView;
8
9 public class MainActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14
15         ConstraintLayout constraintLayout = new ConstraintLayout(this);
16         TextView textView = new TextView(this);
17         textView.setText("Hello Android!");
18         textView.setTextSize(26);
19         // устанавливаем параметры размеров и расположение элемента
20         ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.La
21             (ConstraintLayout.LayoutParams.WRAP_CONTENT, ConstraintLayo
22         // выравнивание по левому краю ConstraintLayout
```

```

23     layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
24     // выравнивание по верхней границе ConstraintLayout
25     layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
26     // устанавливаем параметры для textView
27     textView.setLayoutParams(layoutParams);
28     // добавляем TextView в ConstraintLayout
29     constraintLayout.addView(textView);
30     // в качестве корневого
31     setContentView(constraintLayout);
32 }
33 }

```

Для каждого контейнера конкретные действия по добавлению и позиционированию в нем элемента могут отличаться. В данном случае контейнером 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

```

1 layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;

```

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

А строка кода

```

1 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`).

```

1 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`:

```
1 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 интерфейс 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.