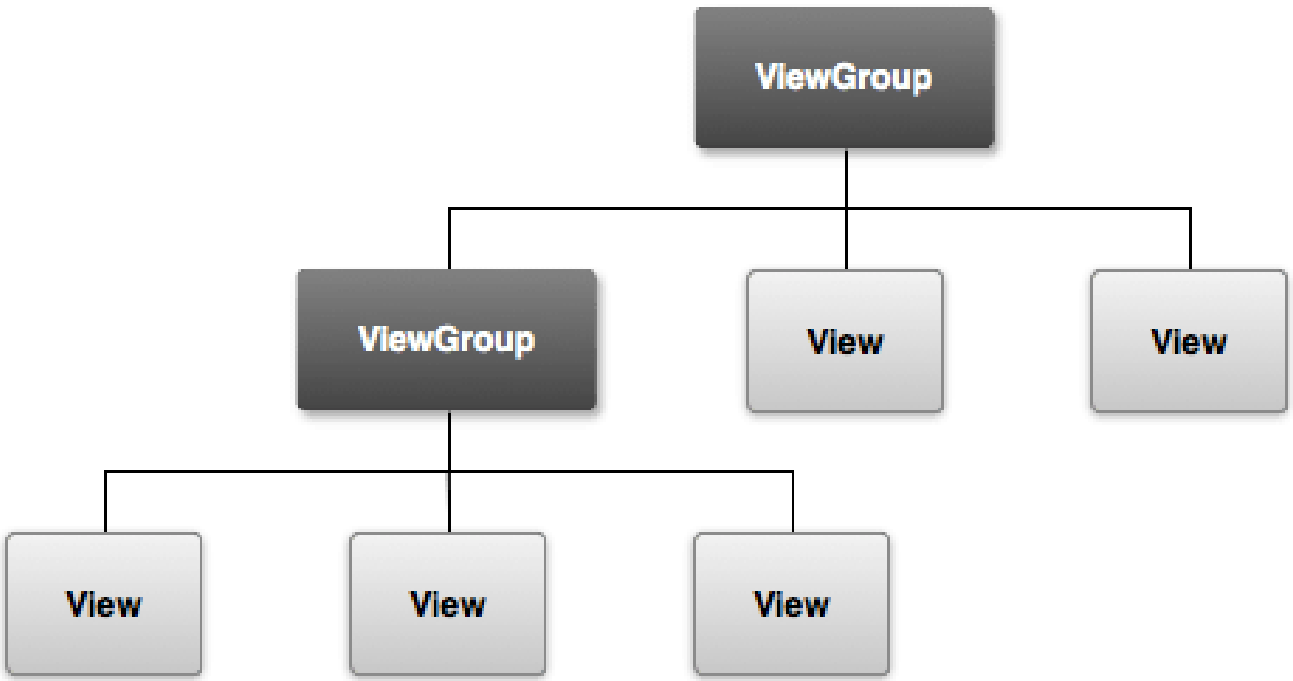
Графический интерфейс пользователя представляет собой иерархию объектов **android.view.View** и **android.view.View Group**. Каждый объект ViewGroup представляет контейнер, который содержит и упорядочивает дочерние объекты **View**. В частности, к контейнерам относят такие элементы, как RelativeLayout,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

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



Большинство визуальных элементов, наследующихся от класса View, такие как кнопки,

текстовые поля и другие, располагаются в пакете `android.widget`

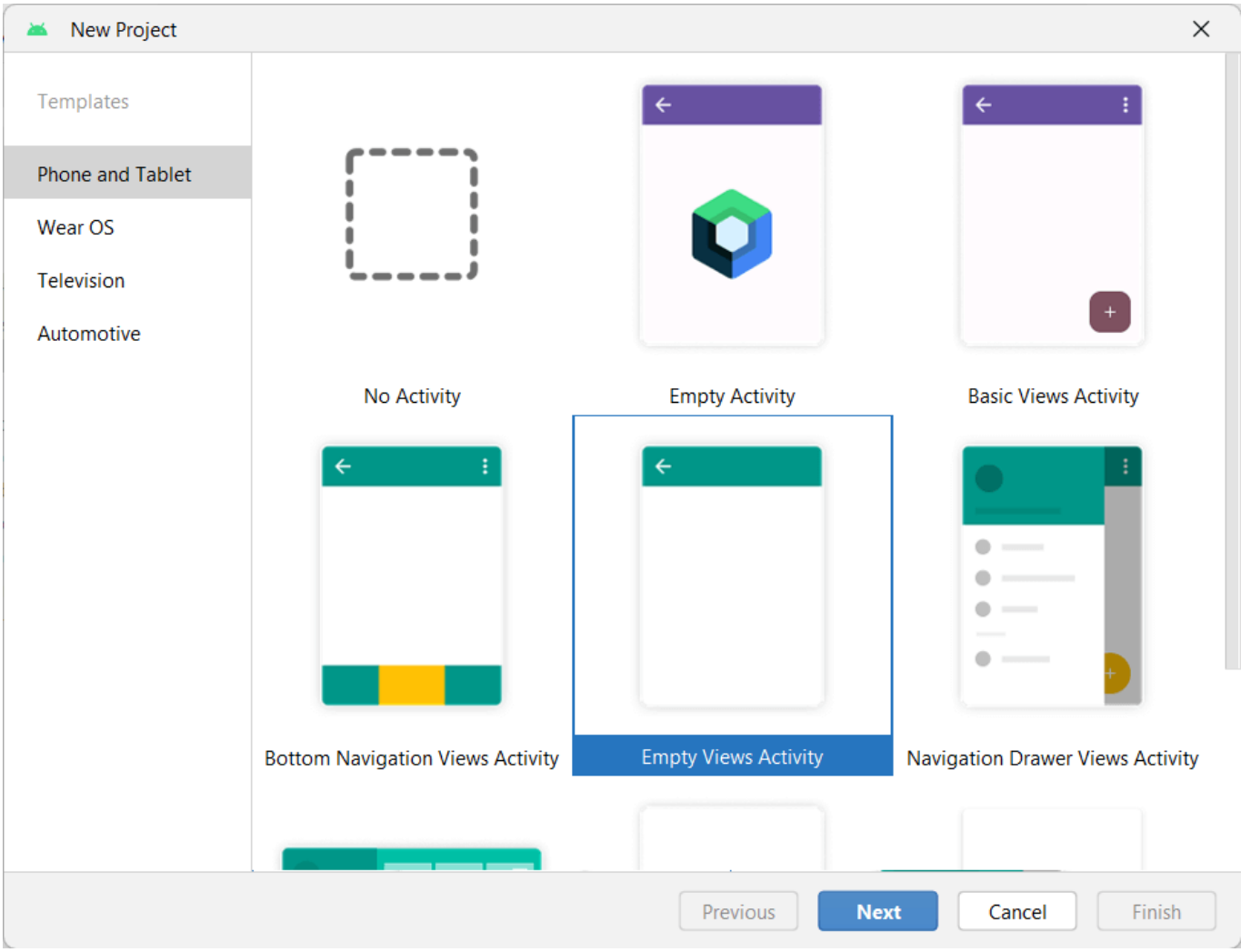
При определении визуального у нас есть три стратегии:

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

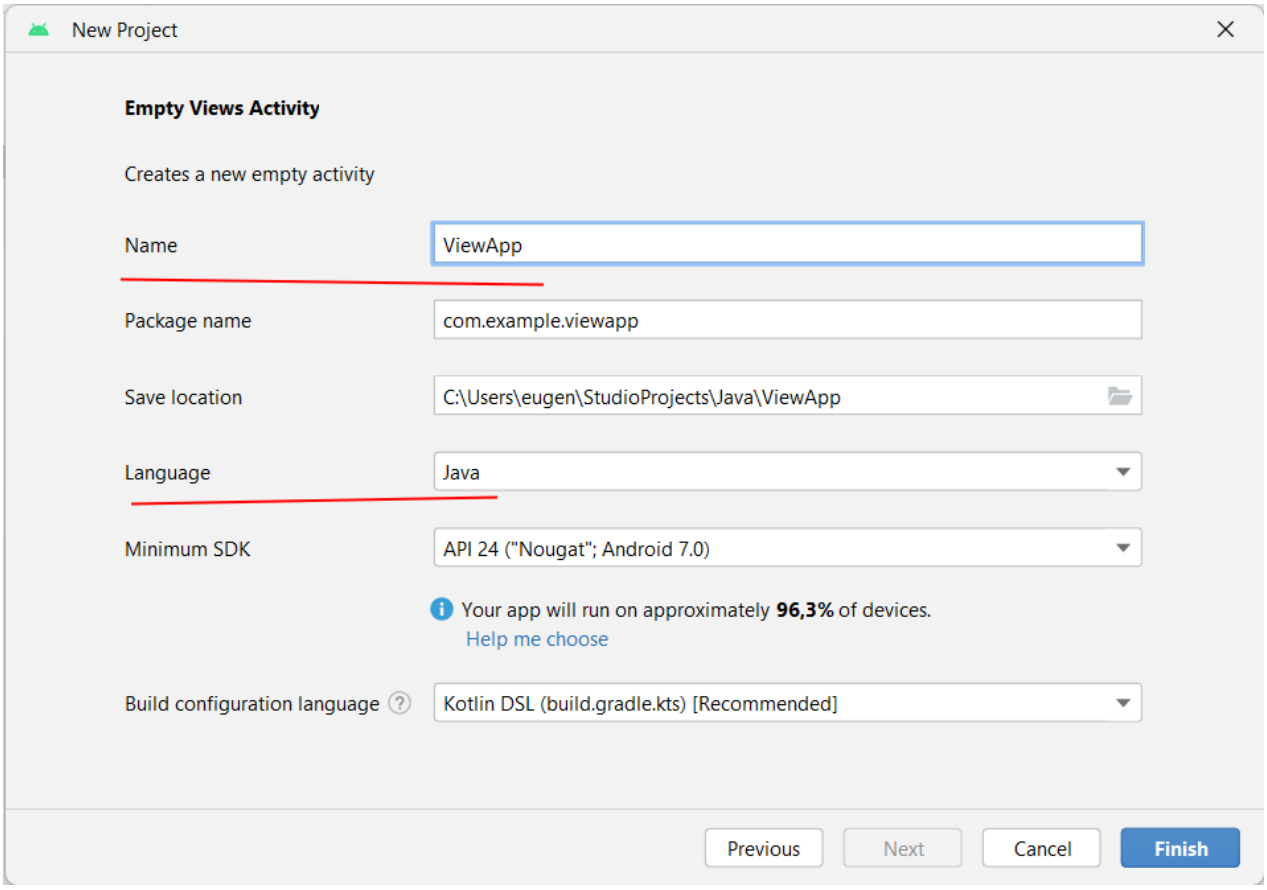
Сначала рассмотрим первую стратегию - определение интерейса в коде Java.

Создание интерфейса в коде java

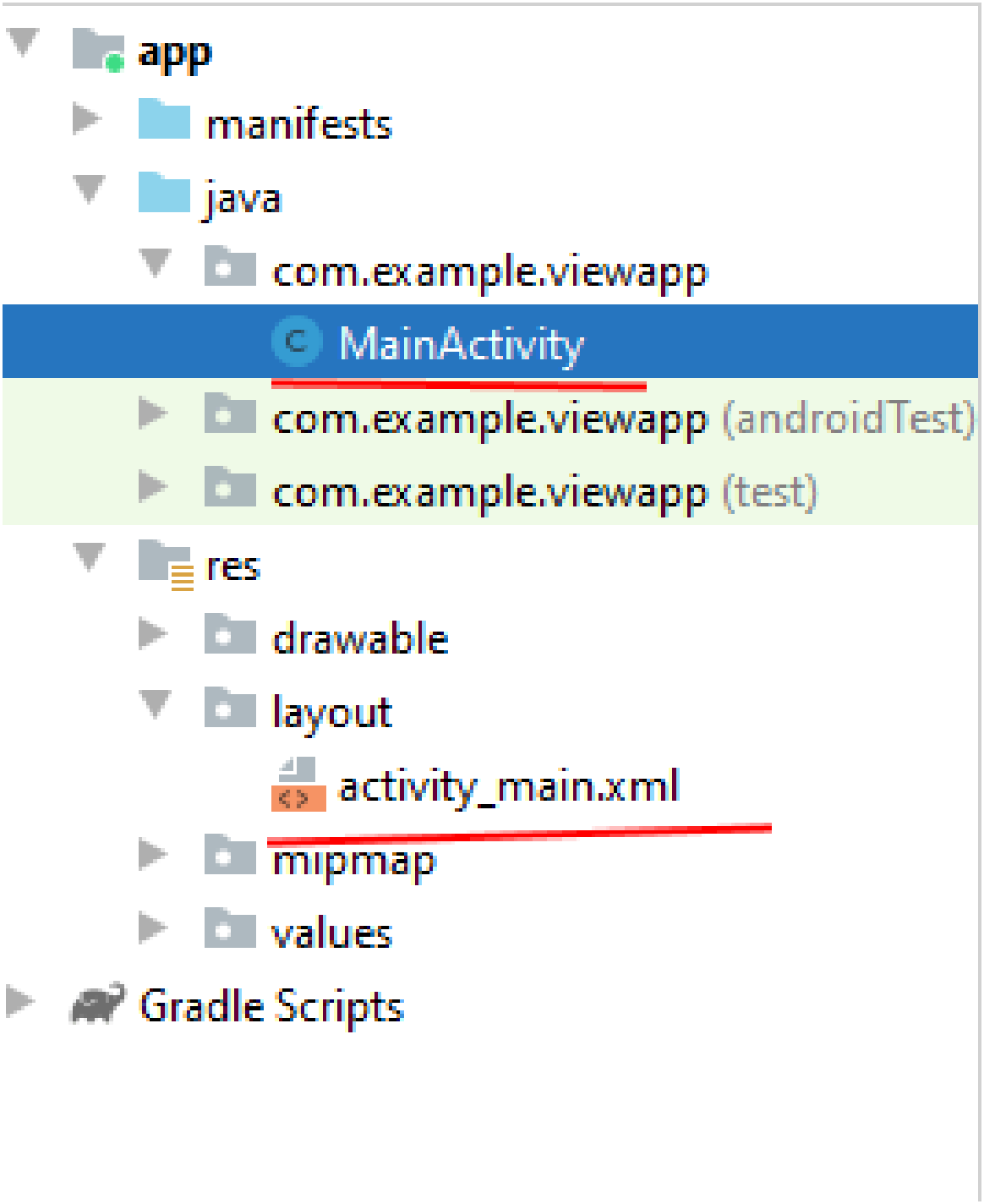
Для работы с визуальными элементами создадим новый проект. В качестве шаблона проекта выберем **Empty Views Activity**:



Пусть новый проект 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:

```
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
       protected void onCreate(Bundle savedInstanceSta
10         super.onCreate(savedInstanceState);
11
12         // создание TextView
13         TextView textView = new TextView(this);
14         // установка текста в TextView
15         textView.setText("Hello METANIT.COM!");
16         // установка высоты текста
           textView.setTextSize(28);
17         // установка визуального интерфейса для ac
18         setContentView(textView);
19     }
20 }
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


14:04
Hello METANIT.COM!

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

```
1  package com.example.viewapp;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import androidx.constraintlayout.widget.ConstraintL
5
6  import android.os.Bundle;
   import android.widget.TextView;
7
8  public class MainActivity extends AppCompatActivity
9
10     @Override
11     protected void onCreate(Bundle savedInstanceSta
12         super.onCreate(savedInstanceState);
13
14         ConstraintLayout constraintLayout
   ConstraintLayout(this);
15         TextView textView = new TextView(this);
16         textView.setText("Hello METANIT.COM!");
17         textView.setTextSize(28);
```

```
18          // устанавливаем параметры размеров и
элемента
19          ConstraintLayout.LayoutParams    layoutPara
ConstraintLayout.LayoutParams
20                  (ConstraintLayout.LayoutParams.WRA
ConstraintLayout.LayoutParams.WRAP_CONTENT);
21          // выравнивание по левому краю ConstraintL
22          layoutParams.leftToLeft
ConstraintLayout.LayoutParams.PARENT_ID;
23          // выравнивание по верхней границе Constra
24          layoutParams.topToTop
ConstraintLayout.LayoutParams.PARENT_ID;
25          // устанавливаем параметры для textView
26          textView.setLayoutParams(layoutParams);
27          // добавляем TextView в ConstraintLayout
28          constraintLayout.addView(textView);
29          // в качестве корневого
30          setContentView(constraintLayout);
31      }
32  }
33
```

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

```
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.LayoutPara
```

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 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.