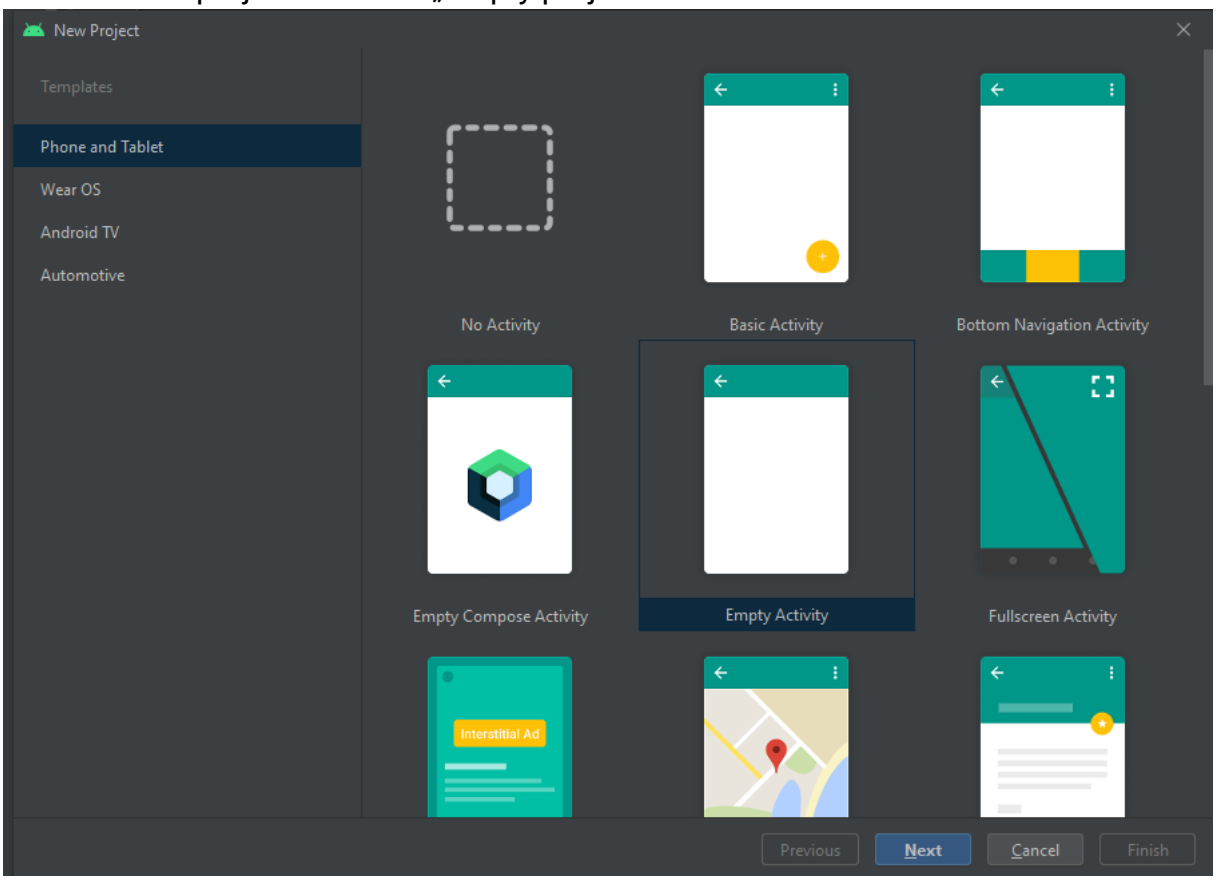
User Interface

A part of a programme, application or operating system responsible for communication with the user. Its most common form is the Graphical User Interface (GUI). It consists of graphic elements that standardize the appearance of the application and present it in a recognizable and predictable form. When designing the interface, apart from the graphic layer (icons, menus, text fields, lists), it is necessary to remember about designing user movement paths, information architecture, interaction processes. That is why the combination of UI and UX (user experience) is now so important in the application design process. In the Android system a complete UI and UX system is represented by Material Design (https://material.io/) Currently implemented version 3.

This lab is designed to show you the basics of creating interfaces along with an introduction to using tools to help you create interfaces. We first create a typical Hello World application and then a Calculator application.

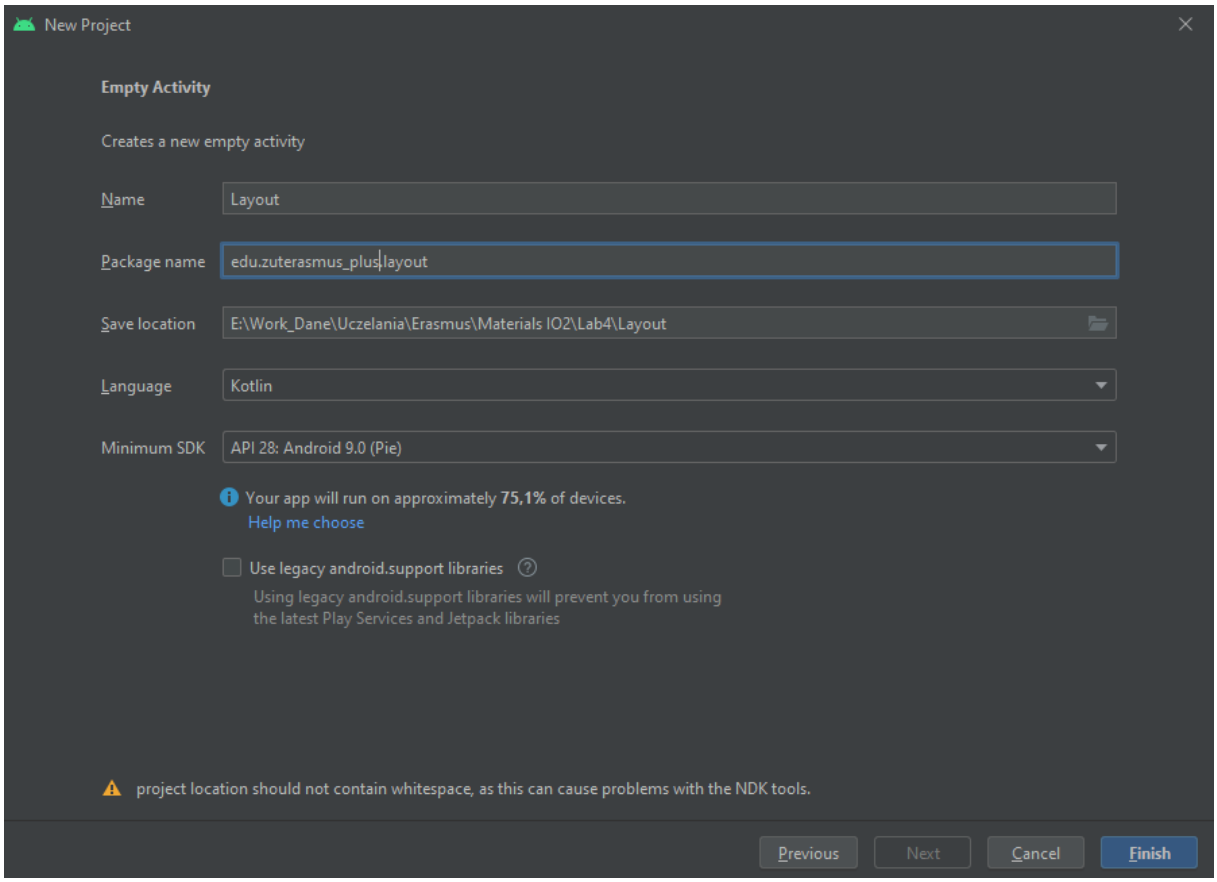
Project Hello World

1. Launch Android Studio
2. Create a new project - choose „Empty project”



3. Enter project name, package name, select API version and path

Funded by
the European Union



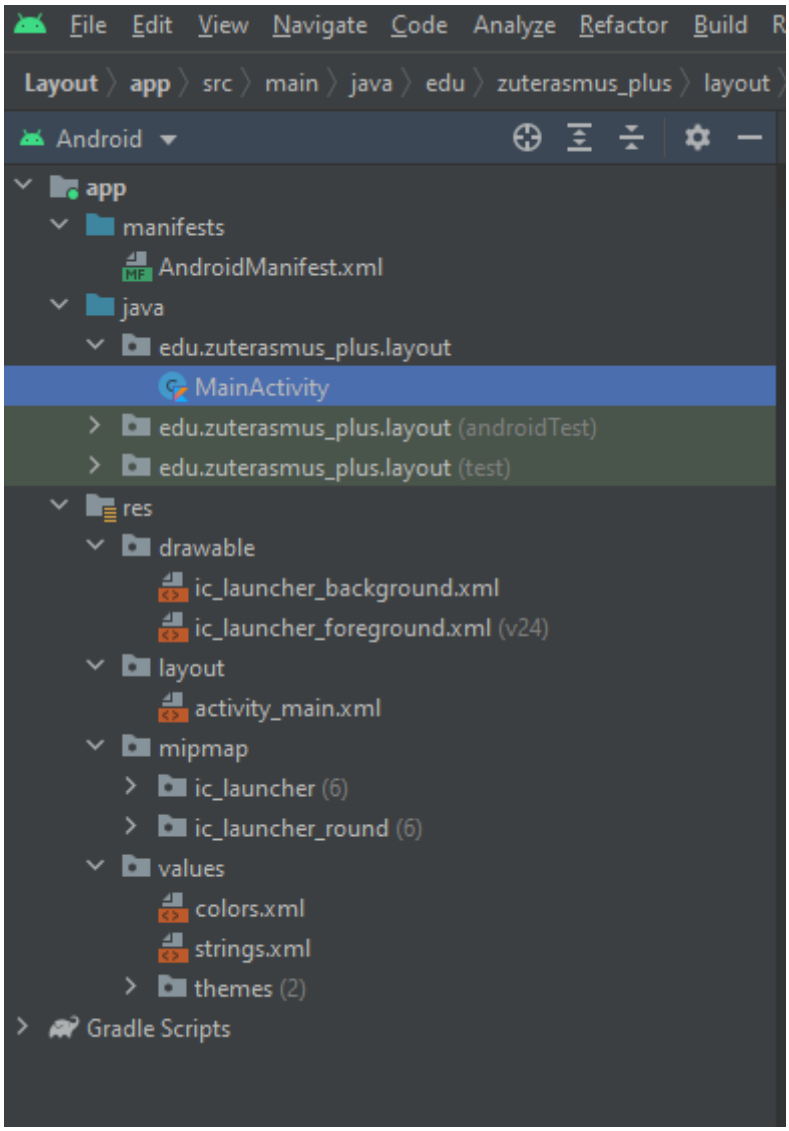
4. Press Finish and wait for the project skeleton to be created.

The following is a starting view of the project and also shows the project structure

We store the code in the JAVA directory (also when writing using the Kotlin language). The manifest directory stores **AndroidManifest.xml** file containing important information for the compiler, including definitions of application components or permissions.

The res directory is used to store information about the application resources, including the layouts directory, where we defile the appearance of the application in an xml file.

Funded by
the European Union

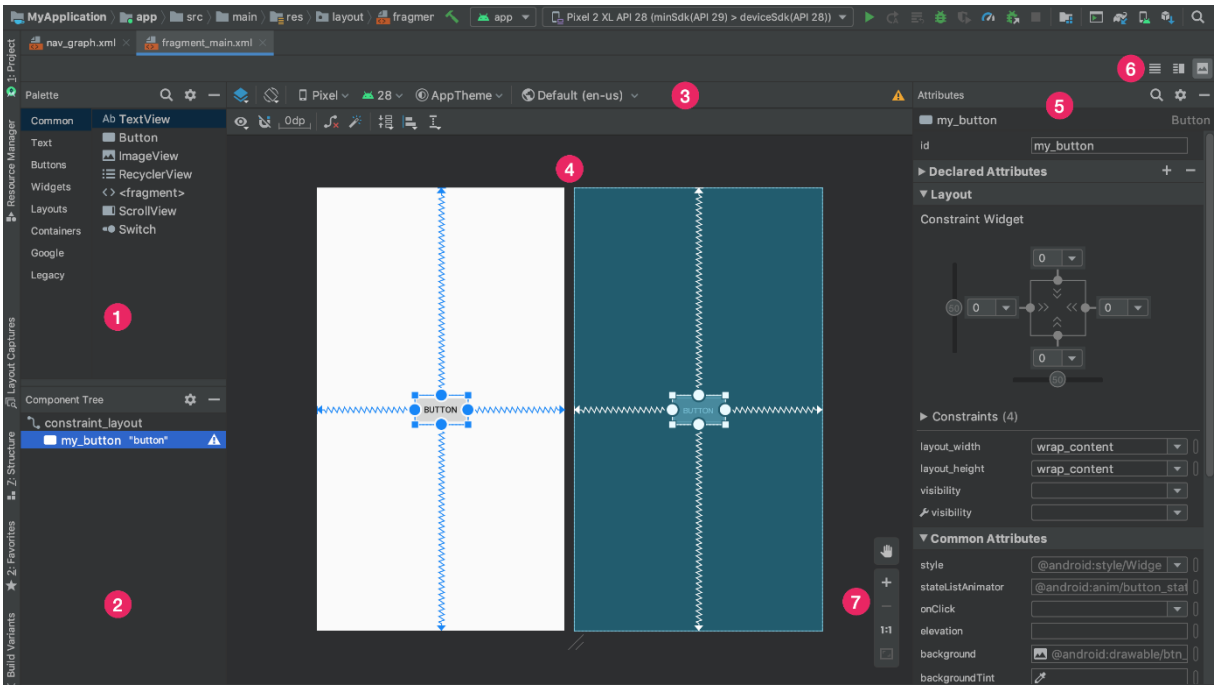


5. Creating the user interface

The interface is created in xml files. Android Studio allows you to create the code directly or using the Graphics Editor.

Select the **activity_main.xml** file from the layout directory. The Layout Editor tool will open (https://developer.android.com/studio/write/layout-editor) :

Funded by
the European Union

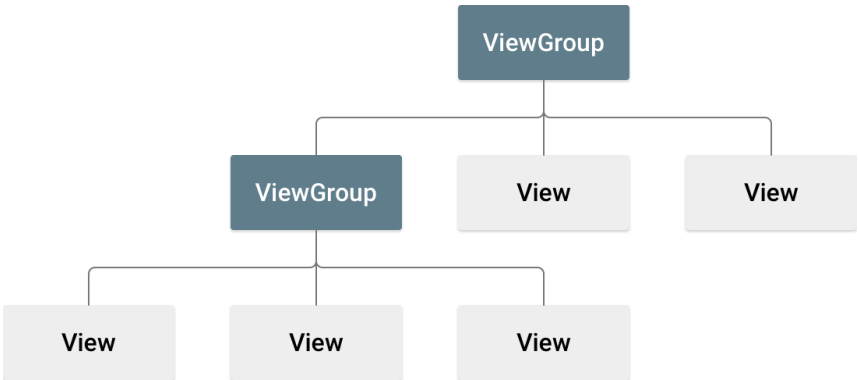


1) **Palette**: Contains various views and view groups that you can drag into your layout.
2) **Component Tree**: Shows the hierarchy of components in your layout.
3) **Toolbar**: Click these buttons to configure your layout appearance in the editor and change layout attributes.
4) **Design editor**: Edit your layout in Design view, Blueprint view, or both.
5) **Attributes**: Controls for the selected view's attributes.
6) **View mode**: View your layout in either Code code mode icon, Design design mode icon, or Split split mode icon modes. Split mode shows both the Code and Design windows at the same time.
7) **Zoom and pan controls**: Control the preview size and position within the editor.

For more information about the interface, see:
https://developer.android.com/studio/write/layout-editor

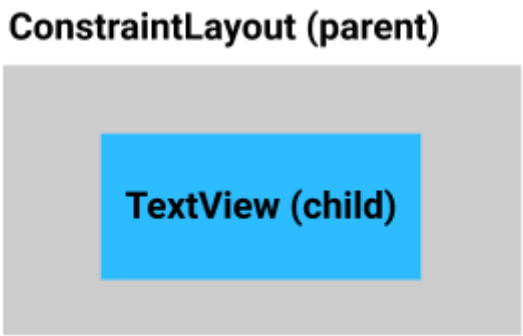
The user interface is built hierarchically using **ViewGroup** (Layout) and **View** (widget) objects

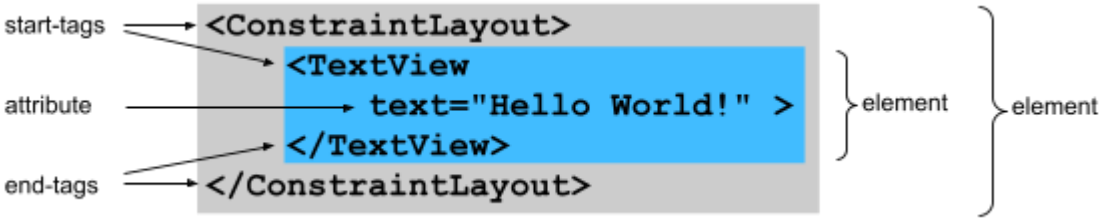
Funded by
the European Union



As can be seen in the figure above **ViewGroup** objects allow you to create hierarchies in which individual objects are contained

6. The user interface in the created project is defined as follows





The xml form is as follows

Funded by
the European Union

```
<?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>
```

We use attributes to define the appearance of objects. Individual XML tags (e.g. TextView) w pliku XML odpowiadają nazwą klas w których te obiekty zdefiniowano.

Look at the tag for the **ConstraintLayout**, and notice that it use **androidx.constraintlayout.widget.ConstraintLayout** instead of just **ConstraintLayout** This is because ConstraintLayout is part of Android Jetpack, which contains libraries of code which offers additional functionality on top of the core Android platform. Jetpack has useful functionality you can take advantage of to make building apps easier. You'll recognize this UI component is part of Jetpack because it starts with "**androidx**".

Excecise 1

a) Replace "Hello World" with your name.
b) Add a new TextView object with the name of the city you are from.
c) Place all the captions in the strings.xml file (guide - https://developer.android.com/guide/topics/resources/string-resource)

Exercise 2 (Additional)

Do the exercise on this page

https://developer.android.com/codelabs/constraint-layout

Funded by
the European Union

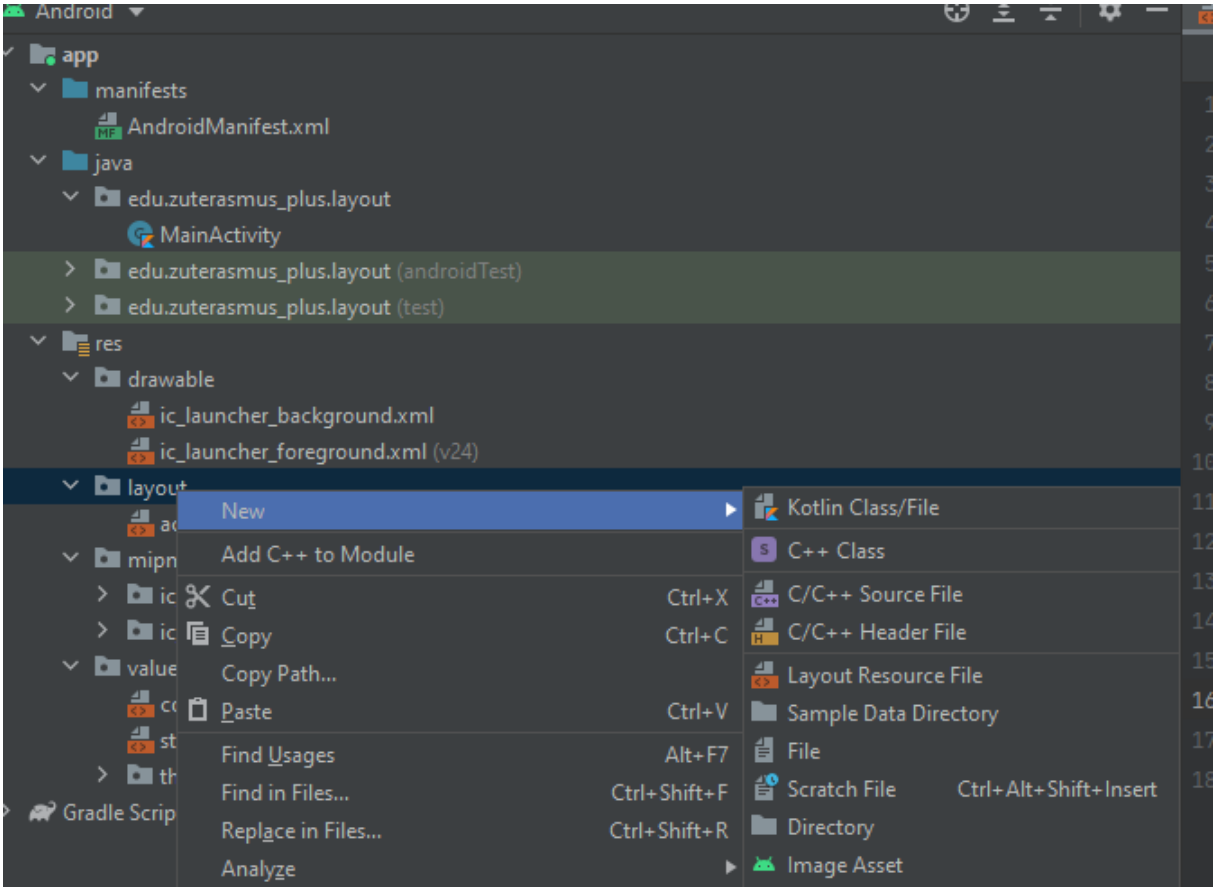
Project Calculator

1. Create a new layout or download new project from https://github.com/rmaciaszczyk/SummerSchool_Lab1

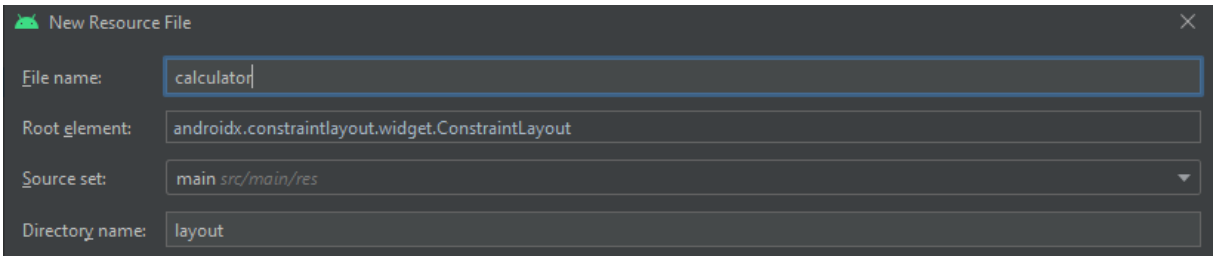
If you download the project from the repository, go to no. 2

Right-click on the folder

layout->New->Layout Resourse File



Create a new layout file "**calculator.xml**"



Funded by
the European Union

Copy content from file:
https://raw.githubusercontent.com/rmaciaszczyk/SummerSchool_Lab1/main/app/src/main/res/layout/calculator.xml

Download Icon Backspace 24 px:
https://github.com/rmaciaszczyk/SummerSchool_Lab1/blob/main/app/src/main/res/drawable/ic_baseline_backspace_24.xml

Import SVG file https://developer.android.com/studio/write/vector-asset-studio#svg

2. Look inside calculator.xml. Analyse its structure

Exercise 3

a) What and how many **ViewGroup** type objects were used
b) What and how many **View** type objects were used

3. The next step is to create the calculator code.

Layout assignment

a) Navigate to file ***MainActivity.kt***
b) Inside ***onCreate()*** change

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

activity_main.xml -> ***calculator.xml***

b) Launch the application

4. Definition of objects
It is now necessary to define all the buttons to which we will assign actions.

In Kotlin all variables must be initialised or you have to explicitly specify that they can take the value null. It is also possible to specify that they will be initialized later before the first use(lateinit).

For example:

```
//Regular initialization means non-null by default
private var myName: String = "Erasmus"
//To allow nulls, you can declare a variable as a nullable string by writing String?: This
private var myNameNullable: String? = null
//You should be very sure that your lateinit variable will be initialized before
accessing
private lateinit var lateMyName: String
```

Funded by
the European Union

Read-only local variables are defined using the keyword ***val***. They can be assigned a value only once. Variables that can be reassigned use the ***var*** keyword.

a) Definition of variables in the code
Please write below definition of class

```
class MainActivity : AppCompatActivity() {
private var one: TextView? =null
private lateinit var two:TextView
private var three:TextView? = null
private lateinit var four:TextView
private var five:TextView? = null
private var six:TextView? = null
private var seven:TextView? = null
private var eight:TextView? = null
private var nine:TextView? = null
private var zero:TextView? = null
private var div:TextView? = null
private var multi:TextView? = null
private var sub:TextView? = null
private var plus:TextView? = null
private var dot:TextView? = null
private var equals:TextView? = null
private lateinit var display:TextView
private var clear:TextView? = null
private var backDelete: ImageButton? = null
```

If class names are underlined then an import should be added.
You can also use the key combination **ALT + ENTER** and automatically add import

5. Bind layout with code
Now we must bind object form layout with object form code.

6. Add ***OnClick()*** event to button
   a. Add interface ***View.OnClickListener*** to the definition of class

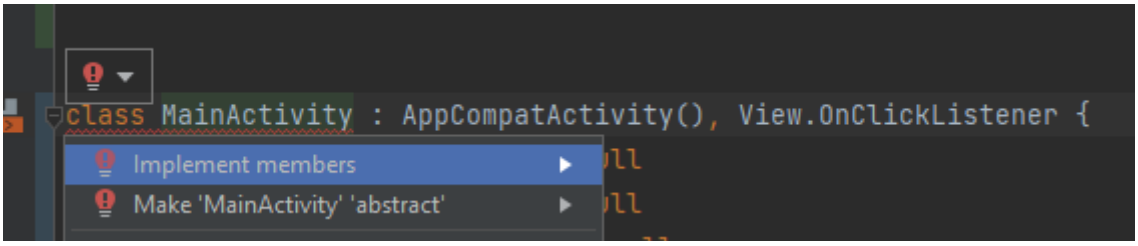
```
class MainActivity : AppCompatActivity(), View.OnClickListener {
```

   b. Define a listener inside the class

Adding an interface forces you to implement its methods. Click on the underlined name of the **MainActivity** class (underlining in red means error), a red light bulb appears, after selecting it you have the possibility to use quick actions. In this case we select **Implement members**

Funded by
the European Union

You can also use the key combination **ALT + ENTER** to bring up this pop-up menu



After action we receive:

```
override fun onClick(p0: View?) {
    TODO("Not yet implemented")
}
```

c. Assign the listener to the buttons (inside **onCreate()** poniżej **set**)

```
one?.setOnClickListener(this)
two.setOnClickListener(this)
three?.setOnClickListener(this)
four.setOnClickListener(this)
five?.setOnClickListener(this)
six?.setOnClickListener(this)
seven?.setOnClickListener(this)
eight?.setOnClickListener(this)
nine?.setOnClickListener(this)
zero?.setOnClickListener(this)
div?.setOnClickListener(this)
multi?.setOnClickListener(this)
div?.setOnClickListener(this)
multi?.setOnClickListener(this)
sub?.setOnClickListener(this)
plus?.setOnClickListener(this)
dot?.setOnClickListener(this)
equals?.setOnClickListener(this)
display?.setOnClickListener(this)
clear?.setOnClickListener(this)
backDelete?.setOnClickListener(this)
```

d. Finish defining the listener

Funded by
the European Union

```
override fun onClick(p0: View?) {
  if (isError) {
     display.text=""
     isError=false
  }

  when  (p0?.id){
     R.id.one-> display.append("1")
     R.id.two-> display.append("2")
     R.id.three-> display.append("3")
     R.id.four-> display.append("4")
     R.id.five-> display.append("5")
     R.id.six-> display.append("6")
     R.id.seven-> display.append("7")
     R.id.eight-> display.append("8")
     R.id.nine-> display.append("9")
     R.id.zero-> display.append("0")
     R.id.div-> display.append("/")
     R.id.multi-> display.append("*")
     R.id.sub-> display.append("-")
     R.id.plus-> display.append("+")
     R.id.dot-> display.append(".")
     R.id.clear-> display.text=""
     R.id.equals-> evaluateExpression(display.text.toString())
     R.id.backDelete -> {
          display.text =
            if((display.text.length -1 )>=0)
                display.text.subSequence(0,display.text.length -1)
           else display.text
           }
  }
}
```

The code above contains a variable that has not yet been defined (**isError**). It is used to determine if an expression is erroneous. It must be declared globally.

```
private var isError: Boolean = true
```

Calculations will be performed using the **exp4j** library - https://github.com/fasseg/exp4j, It is used to calculate mathematical expressions described as a string.

In the code above the calculation will be performed if the user selects the "=" button (R.id.equals), then the value of the entered string will be passed to the evaluateExpression() method, it uses the library mentioned above.

7. Defining the method **evaluateExpression()**,

Funded by
the European Union

a) Adding a library to your application

Go to **build.gradle(app)** and in the dependencies section add

```
implementation 'net.objecthunter:exp4j:0.4.8'
```

after that, synchronise the project. Press: ***Sync Now***

```
Gradle files have changed since last project sync. A project sync may be necessary for the l... Sync Now
        dataBinding true
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-andr
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {

    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'com.google.android.material:material:1.6.0
    implementation 'androidx.constraintlayout:constraintlayout
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'
    androidTestImplementation 'androidx.test.espresso:espresso-

    implementation 'net.objecthunter:exp4j:0.4.8'
}
```

b) Add import

```
import net.objecthunter.exp4j.ExpressionBuilder
```

Funded by
the European Union

c) Add evaluateExpression method

```
private fun evaluateExpression(inputString: String) {
    val expression = ExpressionBuilder(inputString).build()
    try {
        // Calculate the result and display
        val result = expression.evaluate()
        display.text = result.toString()
        //lastDot = true // Result contains a dot
    } catch (ex: Exception)
    {
        when(ex) {
            is IllegalArgumentException, is ArithmeticException -> {
                display.text = "Error"
                isError = true
            }
            else -> throw ex
        }

    }

}
```

8. Run application

View Binding

View binding is a feature that allows you to more easily write code that interacts with views. Once view binding is enabled in a module, it generates a binding class for each XML layout file present in that module. An instance of a binding class contains direct references to all views that have an ID in the corresponding layout.

In most cases, view binding replaces ***findViewById()***.

1. Enabling View Binding

   To enable view binding in a module, set the **viewBinding** build option to true in the module-level **build.gradle** file, as shown in the following example:

```
android {
 . . .
    buildFeatures {
 . . .
        viewBinding true
    }
 . . .
}
```

2. Usage

   If view binding is enabled for a module, a binding class is generated for each XML layout file that the module contains. Each binding class contains references to the root view and all views that have an ID. The name of the

Funded by
the European Union

binding class is generated by converting the name of the XML file to Pascal case and adding the word "Binding" to the end.
For example: calculator.xml -> generated binding class CalculatorBinding

3. Use view binding in activities

ViewBinding allows us to replace the definitions of all objects from the layout.

a) We replace them with a single object. In our code, let us create an object (**MainActivity.kt**)

```
private lateinit var binding: CalculatorBinding
```

All previously defined screen related objects should be removed from the code

To set up an instance of the binding class for use with an activity, perform the following steps in the activity's **onCreate()** method:

b) Call the static **inflate()** method included in the generated binding class. This creates an instance of the binding class for the activity to use.
c) Get a reference to the root view by either calling the **getRoot()** method or using Kotlin property syntax.
d) Pass the root view to **setContentView()** to make it the active view on the screen.

```
class MainActivity : AppCompatActivity(), View.OnClickListener {

    private var isError: Boolean = true
    private lateinit var binding: CalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = CalculatorBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
```

e) We can now also remove the code that binds the layout elements to the code (this is done automatically by the compiler). We remove the line containing the **findViewById()** method call from the code and all declared object
f) Now we will refer to the individual objects using the **binding** object

Funded by
the European Union

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = CalculatorBinding.inflate(layoutInflater)
    val view = binding.root
    setContentView(view)

    binding.one.setOnClickListener(this)
    binding.two.setOnClickListener(this)
. . .
```

g) References should be changed throughout the code

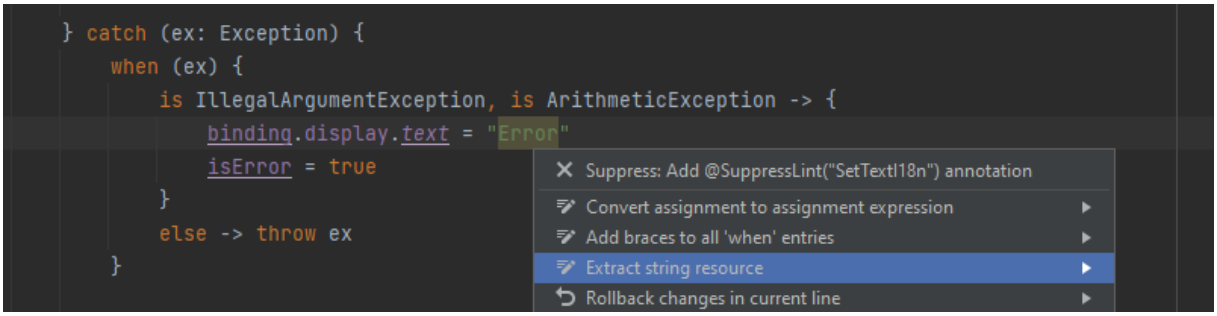
4. Run application
5. Remove unnecessary import

   After these operations, you can delete unnecessary imports, either manually or using a keyboard shortcut. To optimize imports in a file, you can also press Ctrl+Alt+Shift+L , select Optimize imports, and click Run.
   Note that the abbreviation refers to the reformatting of the code and allows also Code Cleaup

6. Removal of warnings

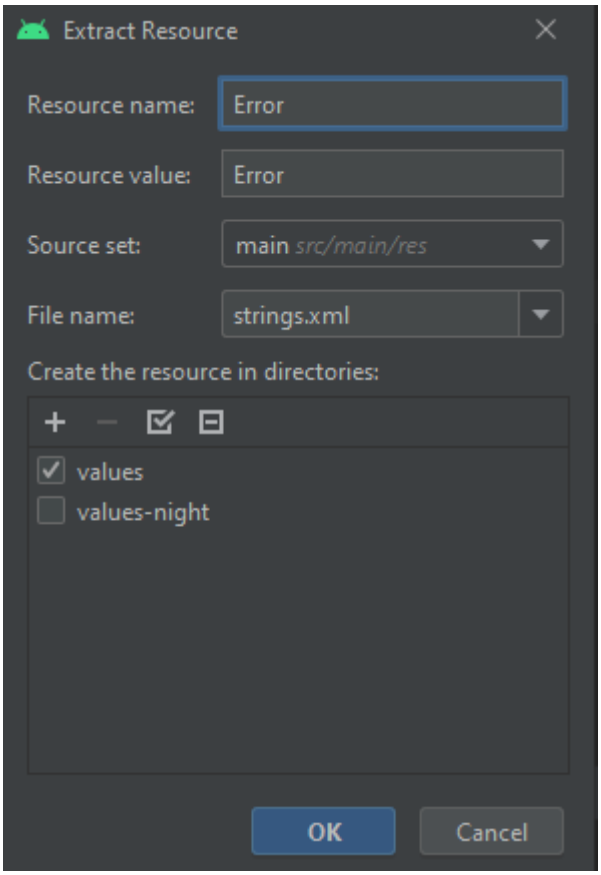
   The compiler analyzing the code tells us to use good practices. One of them is to place all strings in a dedicated resource file. (*res/values/strings.xml*). This solution allows us to translate our application into another language without any problems. More information (https://developer.android.com/training/basics/supporting-devices/languages)

   a) Creating constants using, AndroidStudio prompts:
   In the evaluateExpression method we have hardcoded the String "Error". Hover over this object and use **ALT+Enter** to select **Extract string resource,**



Then fill in the name of the resource

Funded by
the European Union



Code after changes

```
} catch (ex: Exception) {
    when (ex) {
        is IllegalArgumentException, is ArithmeticException -> {
            binding.display.text = getString(R.string.Error)
            isError = true
        }
        else -> throw ex
```

b) Improve the code in your application so that there are no warnings.

Exercise 4

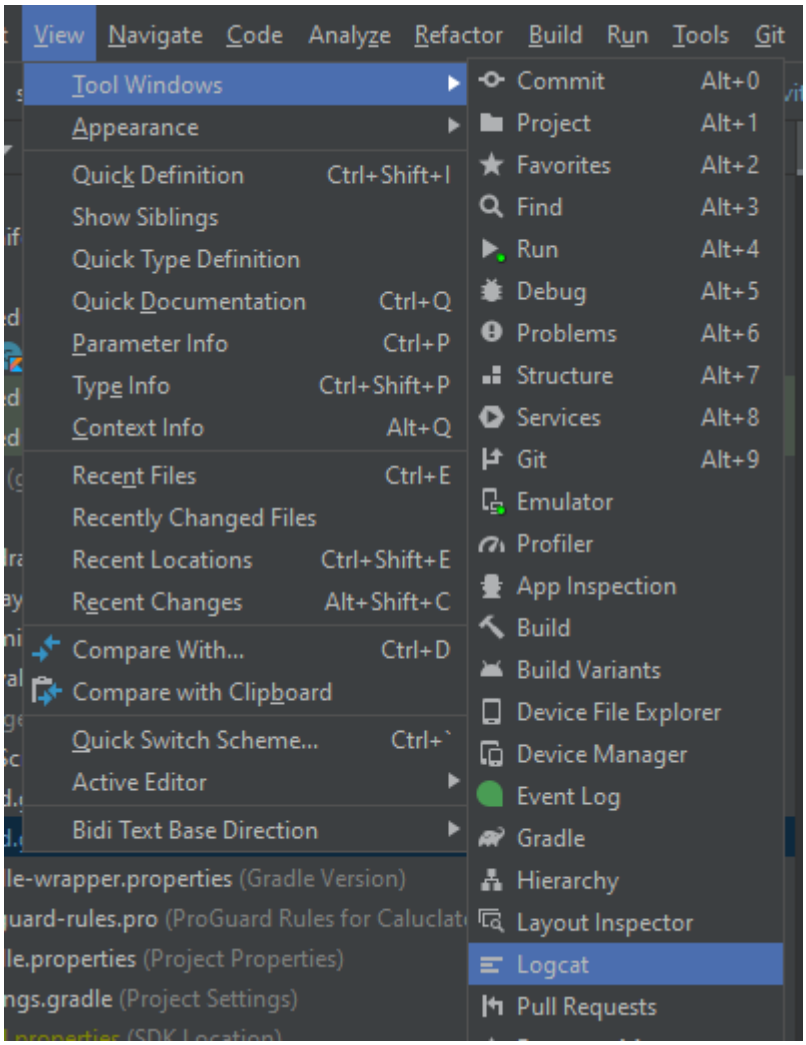
Please test application. Fix the error when a user in the application performs the following actions:

a) Press “2”
b) Press “5”
c) Press “/”
d) Press “=”
e) Press “=”

Funded by
the European Union

☹ ☹ ☹

To see what is causing the error use the Logcat tool



Funded by
the European Union