

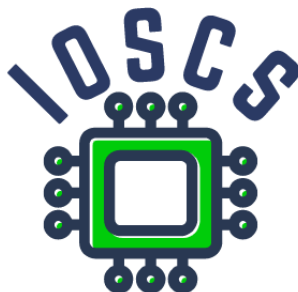
Project: Innovative Open Source Courses for Computer Science

Mobile Application Development Laboratory

**Radosław Maciaszczyk
West Pomeranian University of Technology in Szczecin**

30.05.2021

Innovative Open Source Courses for Computer Science



This teaching material was written as one of the outputs of the project “Innovative Open Source Courses for Computer Science”, funded by the Erasmus+ grant no. 2019-1-PL01-KA203-065564. The project is coordinated by West Pomeranian University of Technology in Szczecin (Poland) and is implemented in partnership with Mendel University in Brno (Czech Republic) and University of Žilina (Slovak Republic). The project implementation timeline is September 2019 to December 2022.

Project information

Project was implemented under the Erasmus+.

Project name: **“Innovative Open Source courses for Computer Science curriculum”**

Project nr: **2019-1-PL01-KA203-065564**

Key Action: **KA2 – Cooperation for innovation and the exchange of good practices**

Action Type: **KA203 – Strategic Partnerships for higher education**

Consortium

ZACHODNIOPOMORSKI UNIWERSYTET TECHNOLOGICZNY W SZCZECINIE

MENDELOVA UNIVERZITA V BRNE

ZILINSKA UNIVERZITA V ZILINE

Erasmus+ Disclaimer

This project has been funded with support from the European Commission. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Copyright Notice

This content was created by the IOSCS consortium: 2019–2022. The content is Copyrighted and distributed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0).



Co-funded by the
Erasmus+ Programme
of the European Union

First App – Hello World

This lab contains:

- Install IDE (Android Studio)
- Create first project
- Show environment
- Implement Activity Lifecycle
- Fire new component

Task 1. Download and install Android Studio

From page <https://developer.android.com/studio> download and install latest version Android Studio

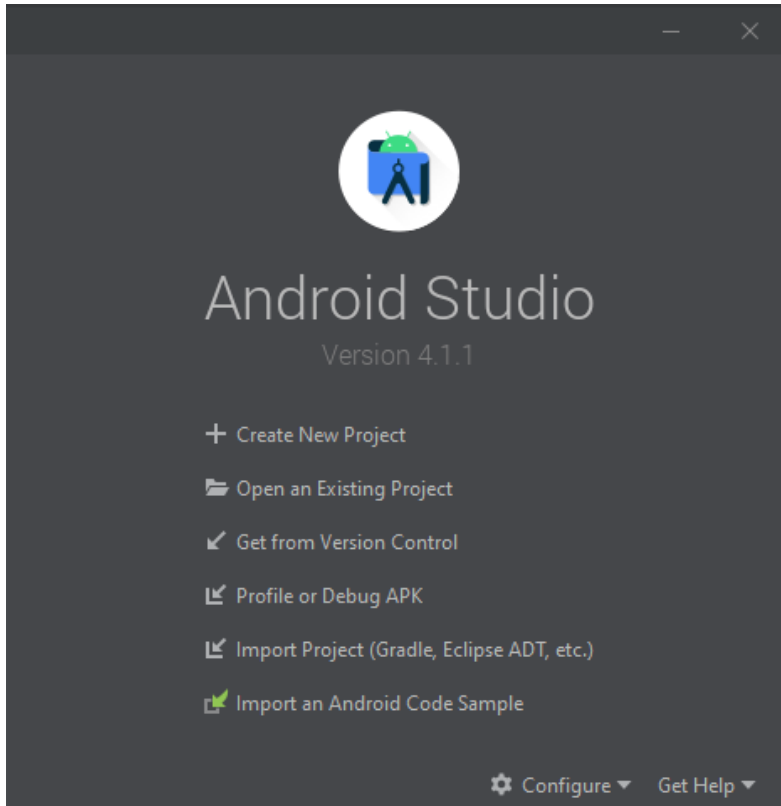
System requirements[1]:

Windows	Mac	Linux
<ul style="list-style-type: none">• 64-bit Microsoft® Windows® 8/10• x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor• 8 GB RAM or more• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)• 1280 x 800 minimum screen resolution	<ul style="list-style-type: none">• MacOS® 10.14 (Mojave) or higher• ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework• 8 GB RAM or more• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)• 1280 x 800 minimum screen resolution	<ul style="list-style-type: none">• Any 64-bit Linux distribution that supports Gnome, KDE, or Unity DE; GNU C Library (glibc) 2.31 or later.• x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD processor with support for AMD Virtualization (AMD-V) and SSSE3• 8 GB RAM or more• 8 GB of available disk space minimum (IDE + Android SDK + Android Emulator)• 1280 x 800 minimum screen resolution

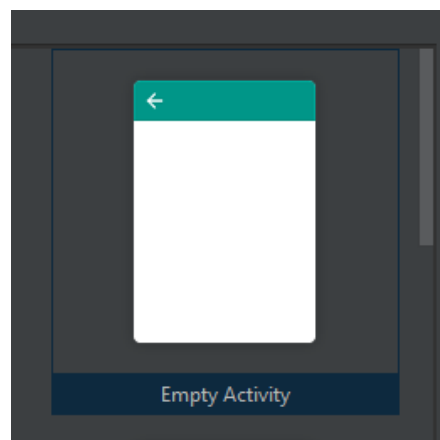


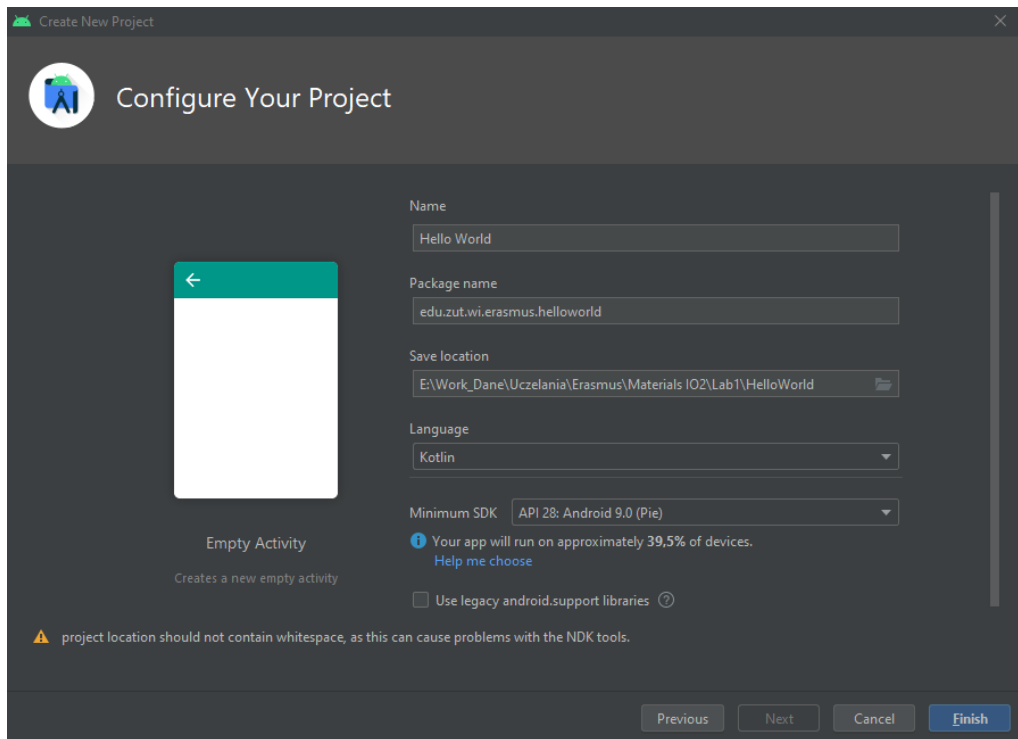
Task 2. Create Hello World

- I. Create project
 1. Select „Empty Activity” -> next



2. Select „Empty Activity” -> next
3. Configure project:
Configure:
 - Name
 - Package name
 - Location project
 - Chose Language
 - Choose SDKFinish





II. Create Android Virtual Device

The Android Studio environment allows the emulation of Android devices. This allows seamless testing of the applications created.

1. Choose from menu: Tools->AVD Manager
2. If not exist AVD create Virtual Device
 - a. Choose a device e.g Pixel 2
 - b. Select a system image e.g. Android 9
 - c. Define name
 - d. If necessary define advanced settings
 - e. Finish

III. Run project

1. Choose from menu: Run -> Run 'app' or Shift + F10

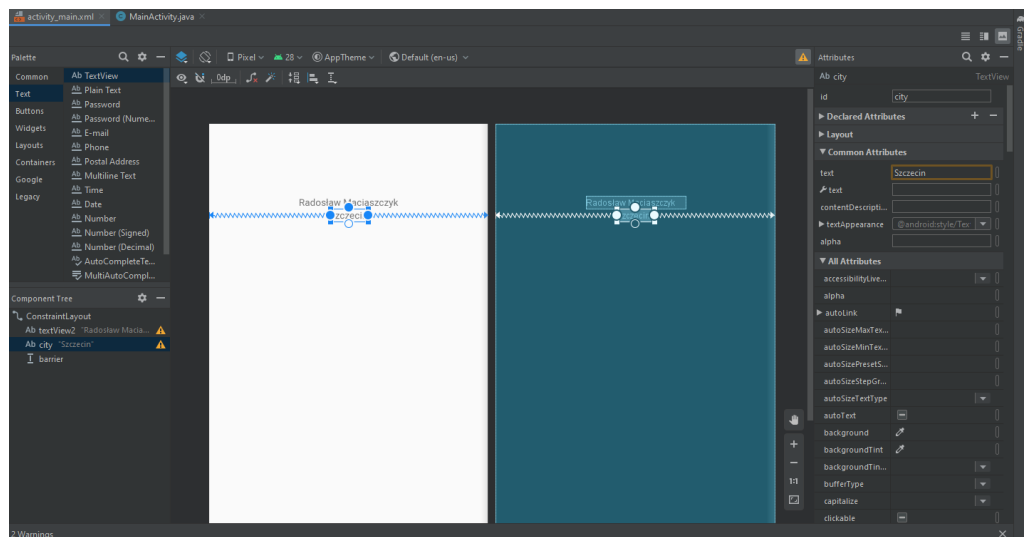
IV. Change text on screen and add new TextView

1. Search file on project **R.layout.activity_main.xml** and show (double click)
2. Click on text "Hello word" and change text
3. For TextView change **andorioid:id** -> **android:id="@+id/name"**
4. From Pallete add new text widget TextView
5. Configure new TextView: **id** and **text**

Fragment code: **R.layout.activity_main.xml**

```
<TextView
    android:id="@+id/name"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Radosław Maciaszczyk"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.148" />

<TextView
    android:id="@+id/city"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Szczecin"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/name" />
```



Task 3. Activity Lifecycle

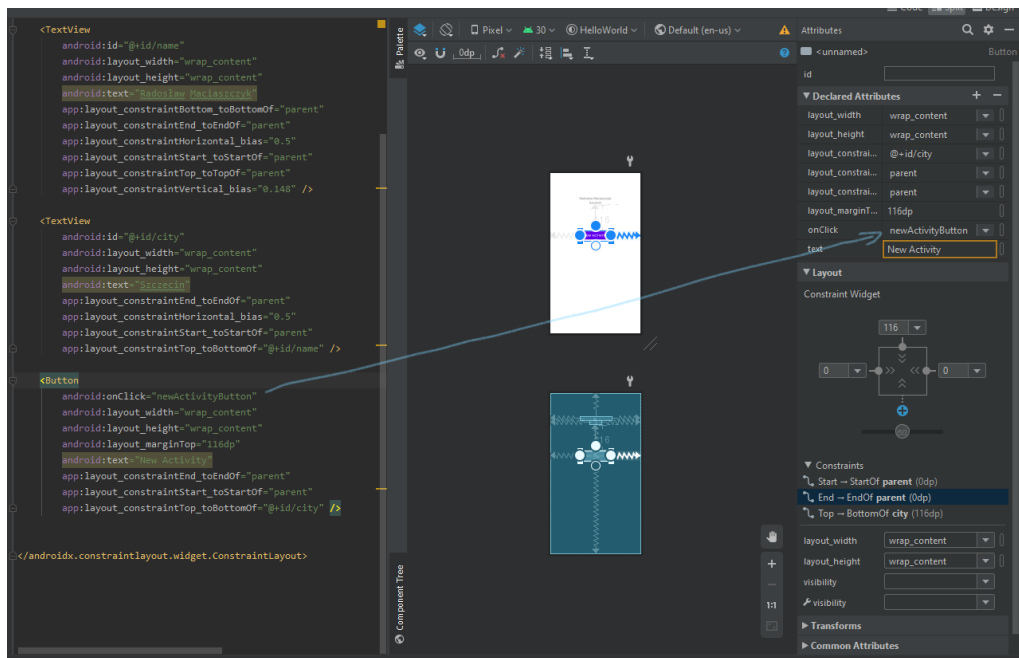
This task show all activity lifecycle, how implement intent and add button. Show how use Log and logcat.

Knowing the life cycle of an activity is crucial to create a properly working application that can store application states between runs.

- V. Add Button to screen
 1. On file **R.layout.activity_main.xml** add from widget new button
 2. Configure button:
id: newActivity
text: New Activity
 3. Go to **MainActivity.kt** and define onClick methods.

```
fun newActivityButton(view: View){  
}
```

4. Go to **activity_main.xml** and add onclick methods to button.



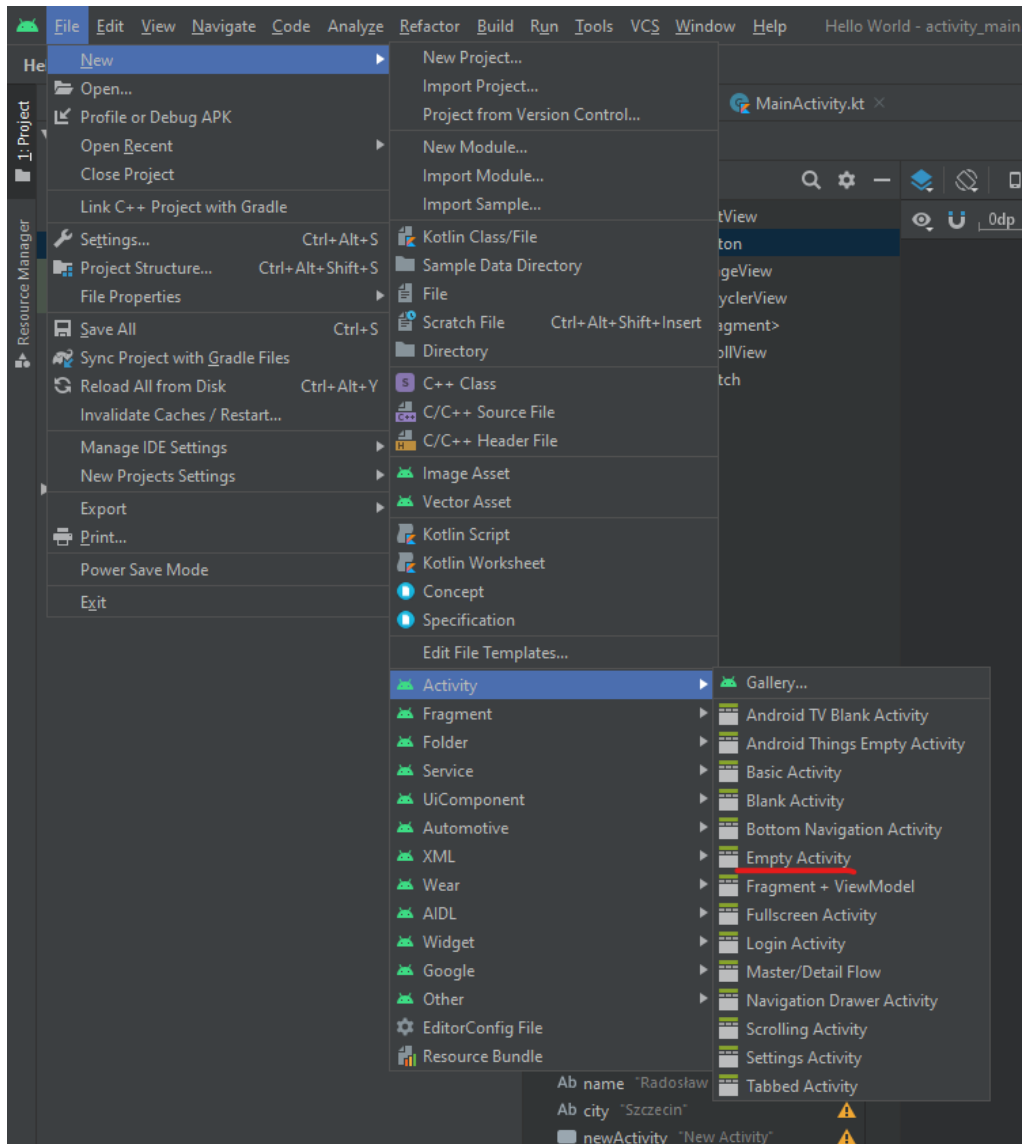
5. Final xml for button

```
<Button  
    android:onClick="newActivityButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"
```

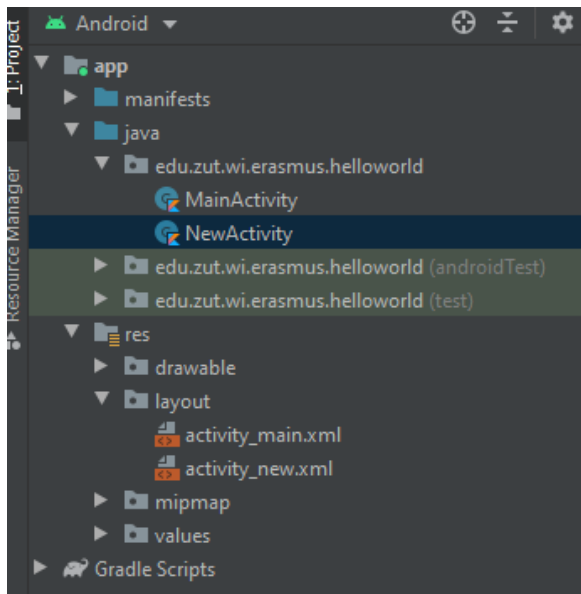
```
android:layout_marginTop="116dp"  
android:text="New Activity"  
app:layout_constraintEnd_toEndOf="parent"  
app:layout_constraintStart_toStartOf="parent"  
app:layout_constraintTop_toBottomOf="@+id/city" />
```

VI. Create new Activity

1. Click on menu File-> New -> Activity -> Empty Activity
2. Configure Activity
Activity name: NewActivity
Click Finish



After created, we have the following directory structure



VII. Add activity lifecycle methods

1. Use keyboard shortcut CTRL+O (Show Override Members)
2. Choose onPause, onStart, onResume, onStop, onDestroy or
or
2. Write yourself in code (step VIII.3 – example)

VIII. Add Log information to all methods

1. Define constant TAG – (Good practice: define TAG with the same name as class name)

```
private val TAG = "NewActivity"
```

2. Add Log to all callback methods
e.g

```
Log.i(TAG, "OnPause")
```

3. Final code NewActivity.kt

```
package edu.zut.wi.erasmus.helloworld

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log

class NewActivity : AppCompatActivity() {
    private val TAG = "NewActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_new)
        Log.i(TAG, "OnCreate")
    }
}
```

```
}

    override fun onPause() {
        super.onPause()
        Log.i(TAG, "OnPause")
    }

    override fun onRestart() {
        super.onRestart()
        Log.i(TAG, "OnRestart")
    }

    override fun onStart() {
        super.onStart()
        Log.i(TAG, "OnStart")
    }

    override fun onStop() {
        super.onStop()
        Log.i(TAG, "OnStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i(TAG, "OnDestroy")
    }
}
```

IX. Open AndroidManifest.xml

1. Add the **Up** button to the app bar

```
<activity android:name=".NewActivity"
    android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

X. Return to MainActivity.kt

1. Add all Lifecycle Methods to this activity, just like the previous one.
2. Add Log information to this methods

XI. Inside **newActivityButton** method

1. Define intent
2. Start intent

```
val intent = Intent(this,NewActivity::class.java)
startActivity(intent)
```

3. Add also Log information

XII. Final MainActivity.kt

```
package edu.zut.wi.erasmus.helloworld

import android.content.Intent
```



```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.util.Log
import android.view.View

class MainActivity : AppCompatActivity() {
    private val TAG = "MainActivity"
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.i(TAG, "OnCreate")
    }
    fun newActivityButton(view: View){
        Log.i(TAG, "newActivityButton")
        val intent = Intent(this,NewActivity::class.java).apply { }
        startActivity(intent)
    }

    override fun onPause() {
        super.onPause()
        Log.i(TAG, "OnPause")
    }

    override fun onRestart() {
        super.onRestart()
        Log.i(TAG, "OnRestart")
    }

    override fun onStart() {
        super.onStart()
        Log.i(TAG, "OnStart")
    }

    override fun onStop() {
        super.onStop()
        Log.i(TAG, "OnStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.i(TAG, "OnDestroy")
    }
}
```

XIII. Run project

Please look on **logcat** and observe log message.

Additional task:

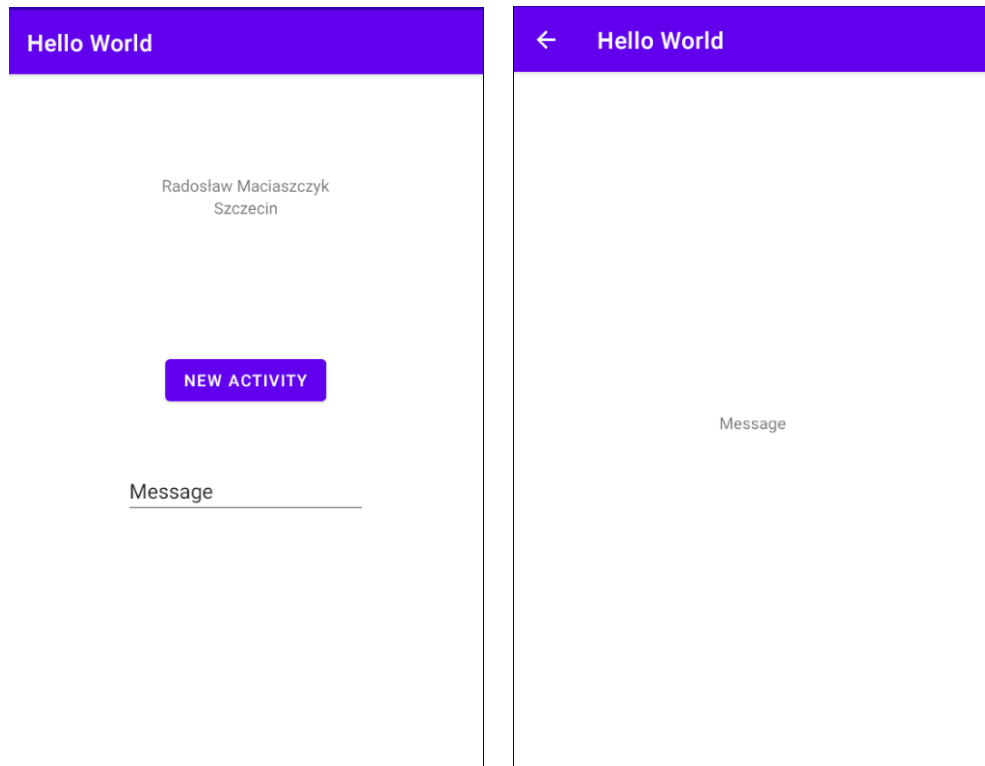
Add to first Activity **Input Text** and using **Intent** send message to second Activity, and add display this message.





Like this:

First Activity send message after click button to Second Activity



More information how to send data with intent:

<https://developer.android.com/training/basics/firstapp/starting-activity>

The resulting application code can be found at https://github.com/matam/Erasmus_Lab1.

Bibliography

[1] - <https://developer.android.com/studio#downloads>

[2] - <https://developer.android.com/studio/intro/keyboard-shortcuts>

More information:

<https://developer.android.com/studio/intro>

<https://developer.android.com/training/basics/firstapp>

<https://developer.android.com/training/basics/firstapp/starting-activity>



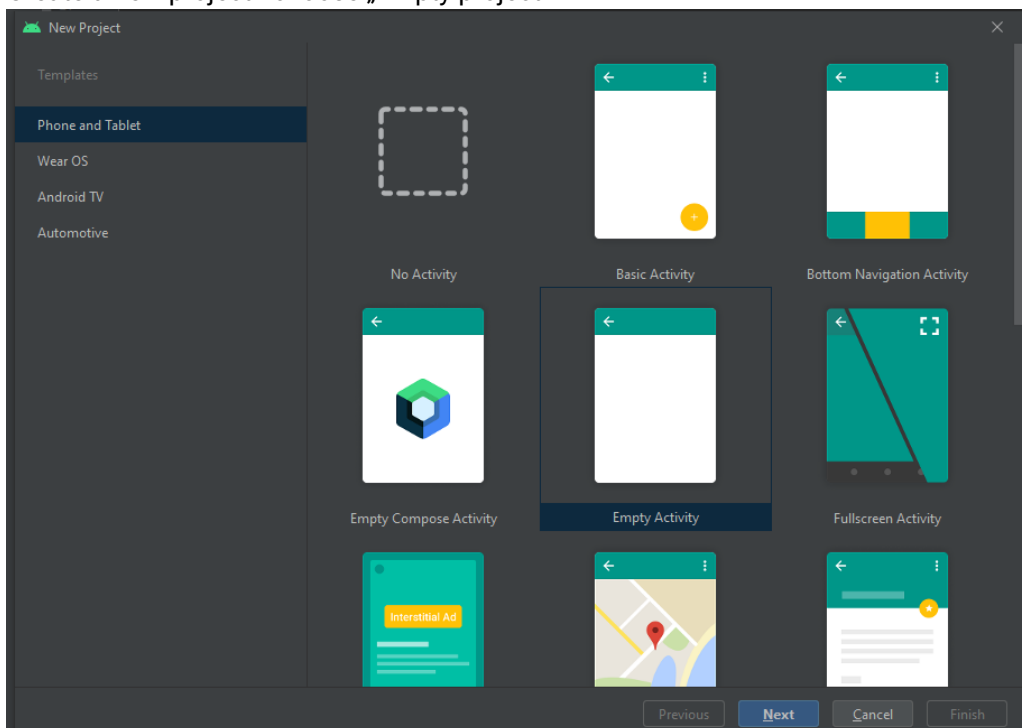
User Interface

User Interface - 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 standardise the appearance of the application and present it in a recognisable and predictable form. When designing the interface, apart from the graphic layer (icons, menus, text fields, lists), it is necessary to remember about creating user movement paths, information architecture, and 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 in version 3.

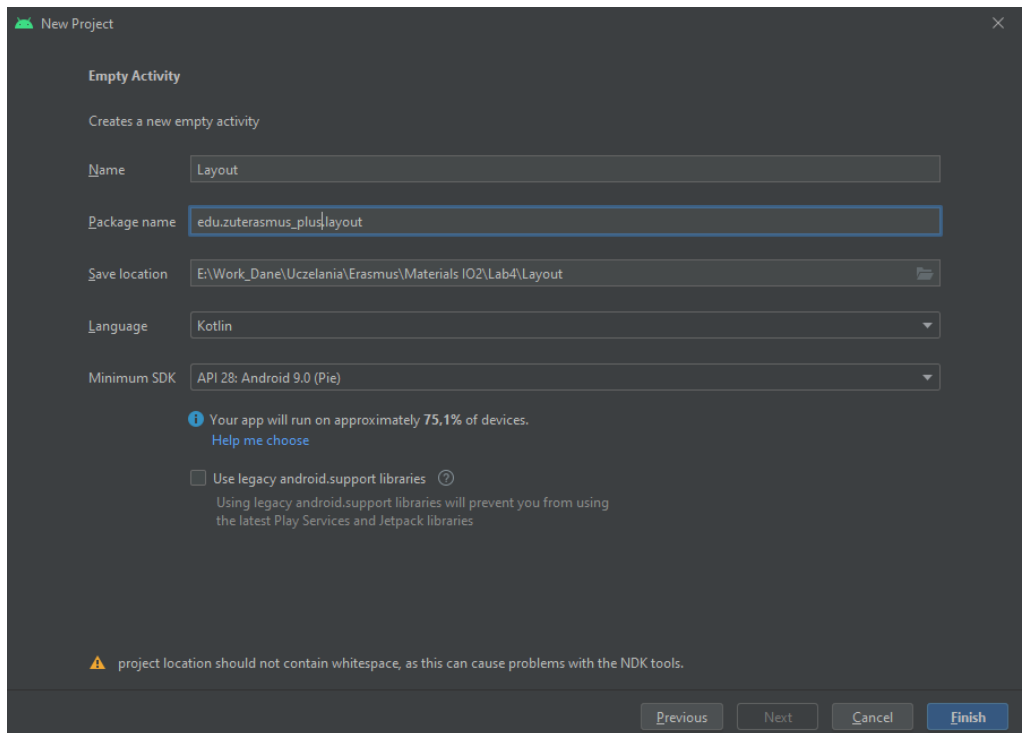
This lab is designed to show you the basics of creating interfaces, along with an introduction to using tools to help you create interfaces. We first create a typical Hello World application and then a Calculator application.

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

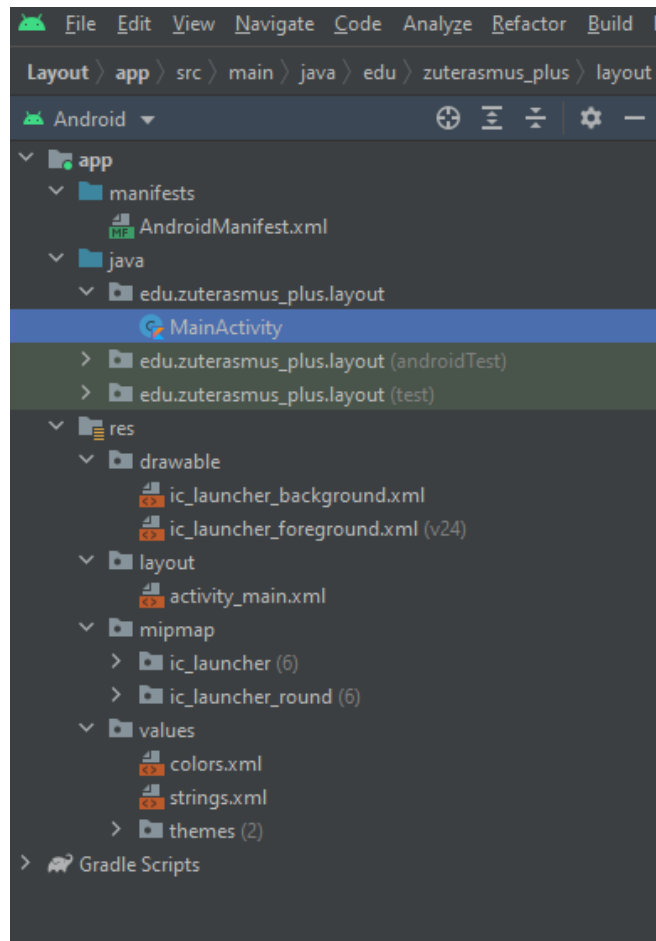


4. Press Finish and wait for the project skeleton to be created.

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 the **AndroidManifest.xml** file containing important information for the compiler, including definitions of application components or permissions.

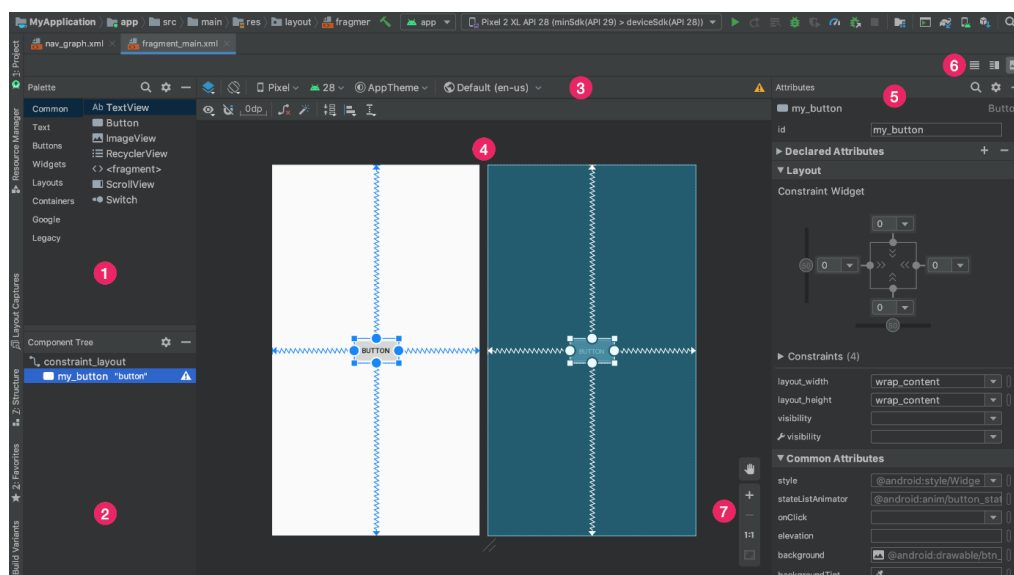
The **res** directory is used to store information about the application resources, including the **layouts** directory, where we define the appearances of the application in an **XML** file.



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>) :

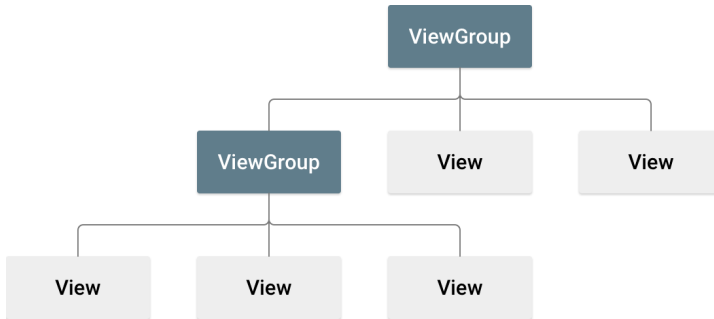


- 1) **Palette:** Contains various views and view groups that you can drag into your layout.
- 2) **Component Tree:** Shows the hierarchy of components in your layout.
- 3) **Toolbar:** Click these buttons to configure your layout appearance in the editor and change layout attributes.
- 4) **Design editor:** 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>

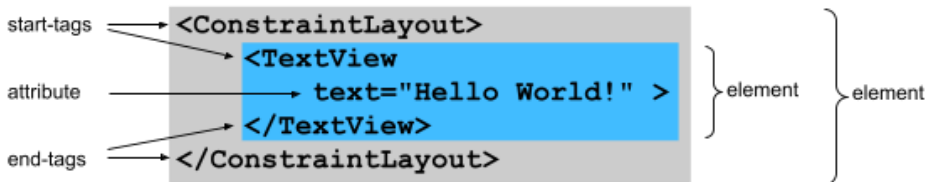
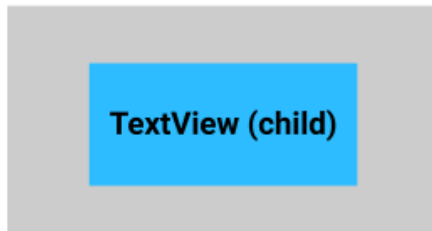
The user interface is built hierarchically using **ViewGroup** (Layout) and **View** (widget) objects



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

ConstraintLayout (parent)



The xml form is as follows

```
<?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`) in the XML file correspond to the names of the classes in which these objects are defined.

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**".

Exercise 1

- Replace "Hello World" with your name.
- Add a new `TextView` object with the name of the city you are from.
- 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>



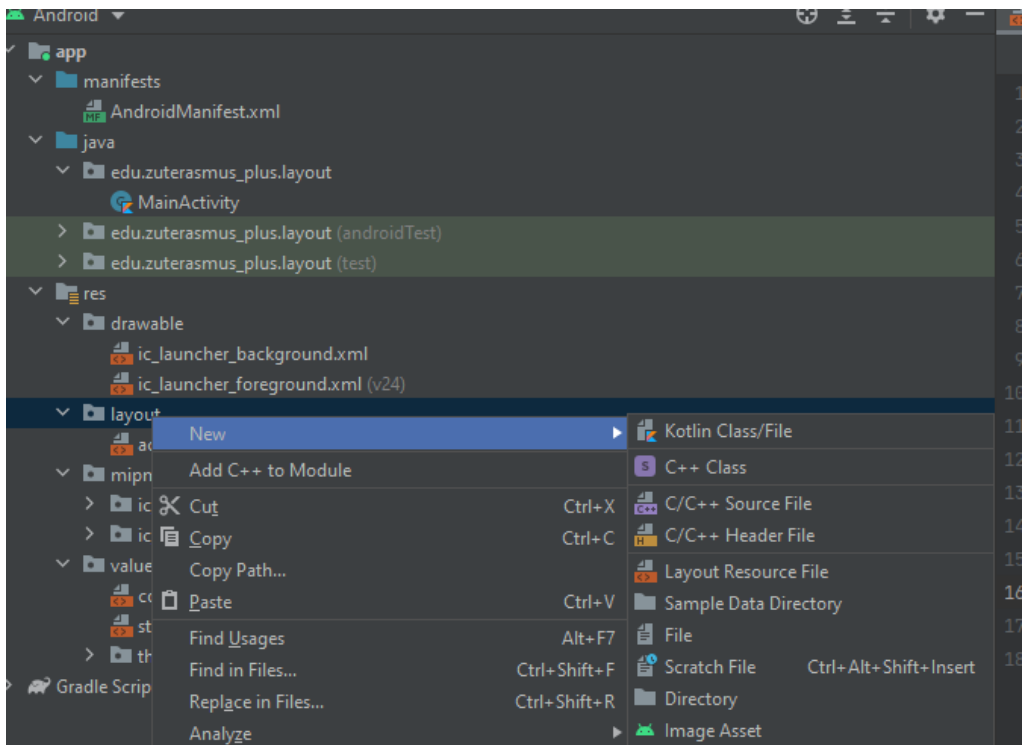
Project Calculator

1. Create a new layout or download new project from https://github.com/matam/Erasmus_Lab2

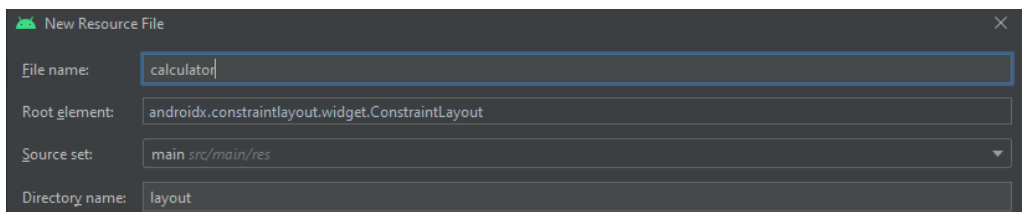
If you download the project from the repository, go to no. 2

Right-click on the folder

layout->New->Layout Resource File



Create a new layout file “**calculator.xml**”





Copy content from file:

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

Create a new styles.xml file in the values directory and copy the content from the following source

https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/styles.xml

Replace the colors.xml file in the values directory and copy the content from the following source

https://raw.githubusercontent.com/matam/Erasmus_Lab1/Calculator/app/src/main/res/values/colors.xml

Download the 24 px Backspace Icon and upload to the drawable directory

https://github.com/rmaciaszczyk/SummerSchool_Lab1/blob/main/app/src/main/res/drawable/ic_baseline_backspace_24.xml

You can import the icon using VectorAsset Studio

<https://developer.android.com/studio/write/vector-asset-studio#svg>

2. Look inside calculator.xml. Analyse its structure

Exercise 3

- What and how many **ViewGroup** type objects were used
- What and how many **View** type objects were used

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

Layout assignment

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

```
setContentview(R.layout.activity_main)
```

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

- 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
```

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.

```
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
```

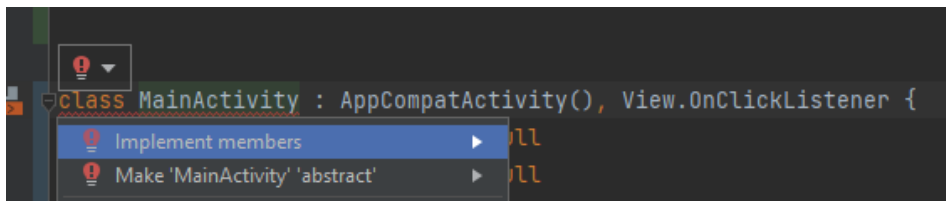
6. Add **OnClick()** event to button
 - a. Add interface **View.OnClickListener** to the definition of class

```
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**

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()** below **setContentView()**)

```
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



```
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()`,

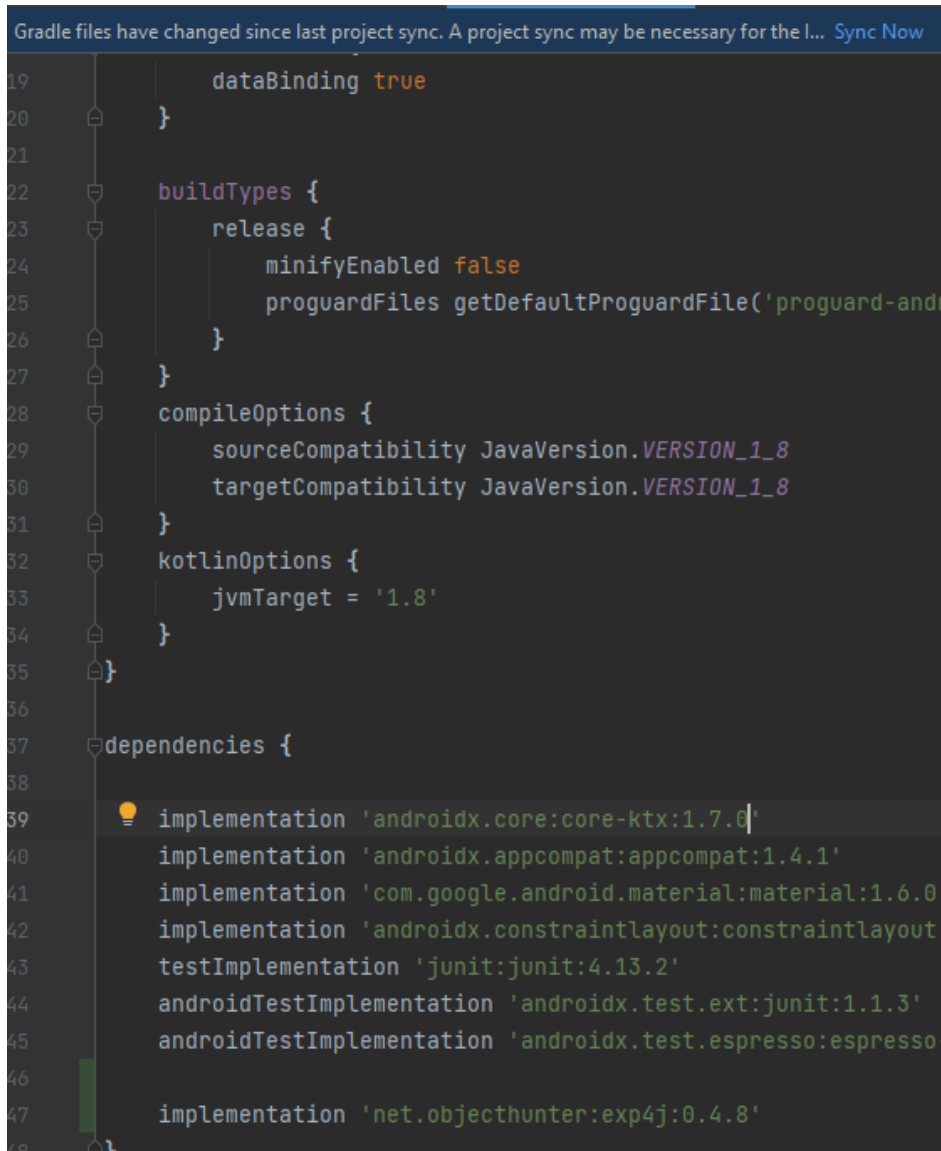


a) Adding a library to your application

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
19     dataBinding true
20 }
21
22 buildTypes {
23     release {
24         minifyEnabled false
25         proguardFiles getDefaultProguardFile('proguard-android-optimize.txt')
26     }
27 }
28 compileOptions {
29     sourceCompatibility JavaVersion.VERSION_1_8
30     targetCompatibility JavaVersion.VERSION_1_8
31 }
32 kotlinOptions {
33     jvmTarget = '1.8'
34 }
35 }
36
37 dependencies {
38
39     implementation 'androidx.core:core-ktx:1.7.0'
40     implementation 'androidx.appcompat:appcompat:1.4.1'
41     implementation 'com.google.android.material:material:1.6.0'
42     implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
43     testImplementation 'junit:junit:4.13.2'
44     androidTestImplementation 'androidx.test.ext:junit:1.1.3'
45     androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
46
47     implementation 'net.objecthunter:exp4j:0.4.8'
48 }
```

b) Add import

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

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

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

```
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 Cleanup

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

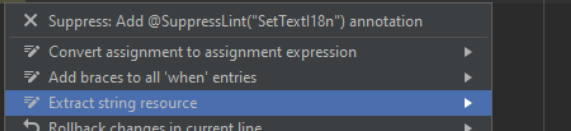
(<https://developer.android.com/training/basics/supporting-devices/languages>)

a) Creating constants using, AndroidStudio prompts:

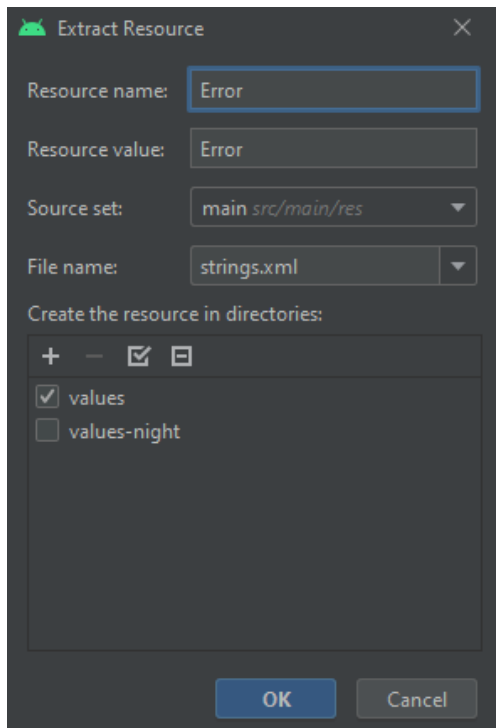
In the evaluateExpression method we have hardcoded the String "Error".

Hover over this object and use **ALT+Enter** to select **Extract string resource**,

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



Then fill in the name of the resource



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.

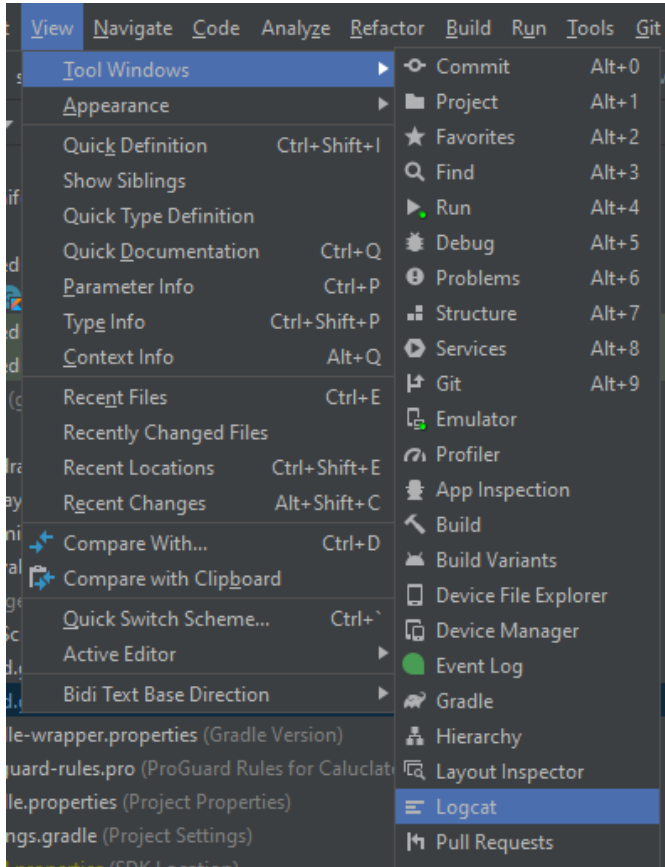
Exercise 4

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

- Press "2"
- Press "5"
- Press "/"
- Press "="
- Press "="



To see what is causing the error use the Logcat tool



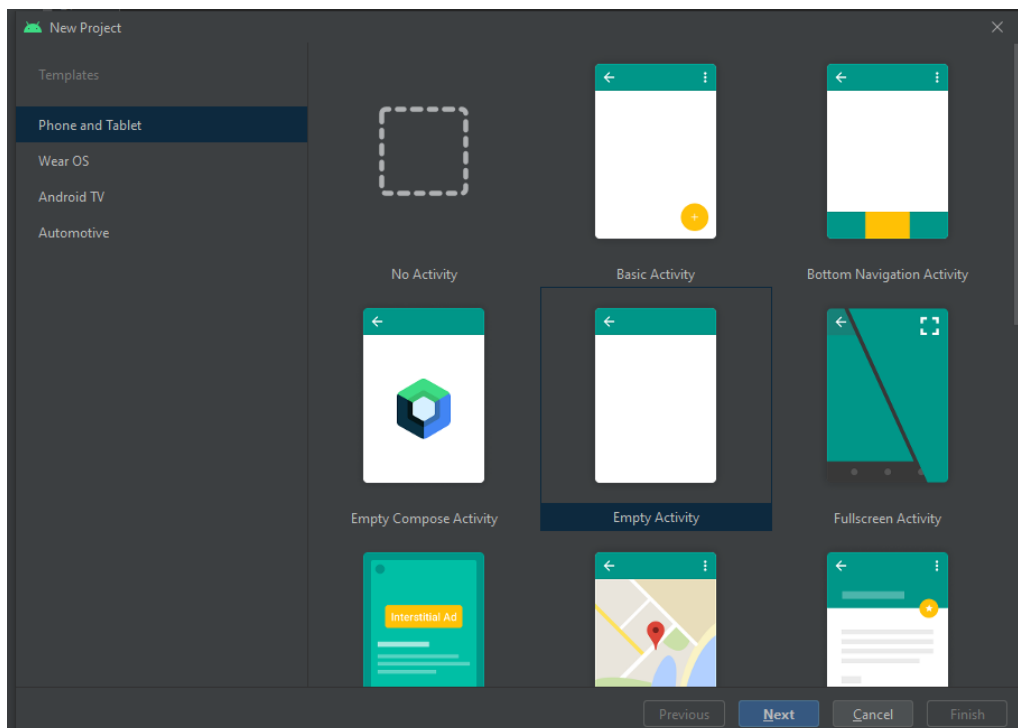
The resulting application code can be found at https://github.com/matam/Erasmus_Lab2.

Sensors

Most mobile devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy, and are useful if you want to monitor three-dimensional device movement or positioning, or you want to monitor changes in the ambient environment near a device. For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

During these labs, a sensor reading application will be created and then rebuilt to use the MVVM design pattern.

1. Create a New project -> Empty Activity





2. Define App Name (**Lab5**) and Package (**edu.zut.erasmus_plus.sensors**), choose minimum API (**API28**)
3. Explore the code
 1. Locate MainActivity.kt, activity_main.xml, AndroidManifest.xml
4. Go to build.gradle -> Upgrade all dependencies and libraries for Project and Module (we can skip)

5. Run App

6. Create a colours schema (not obligatory)

Before defining layout, please use [Color Tool - Material Design](https://material.io/resources/color/#/!/?view.left=0&view.right=0&primary.color=283593&secondary.color=1E88E5) and define your app colours; then, use this colour when defining elements.

Example colour:

<https://material.io/resources/color/#/!/?view.left=0&view.right=0&primary.color=283593&secondary.color=1E88E5>

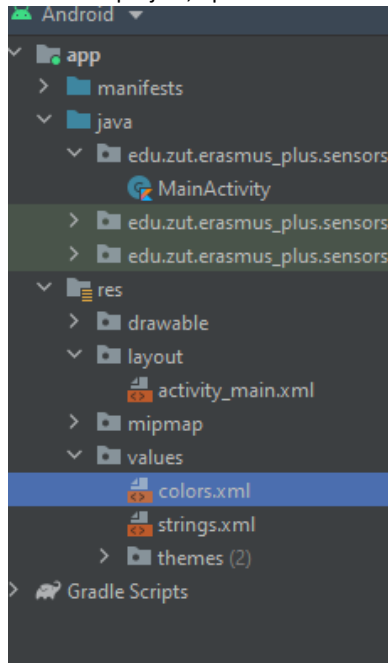
More information about colour: <https://material.io/design/color/the-color-system.html#color-theme-creation>

Once you have selected the correct colour scheme, go to the ACCESSIBILITY tab and check the warnings.



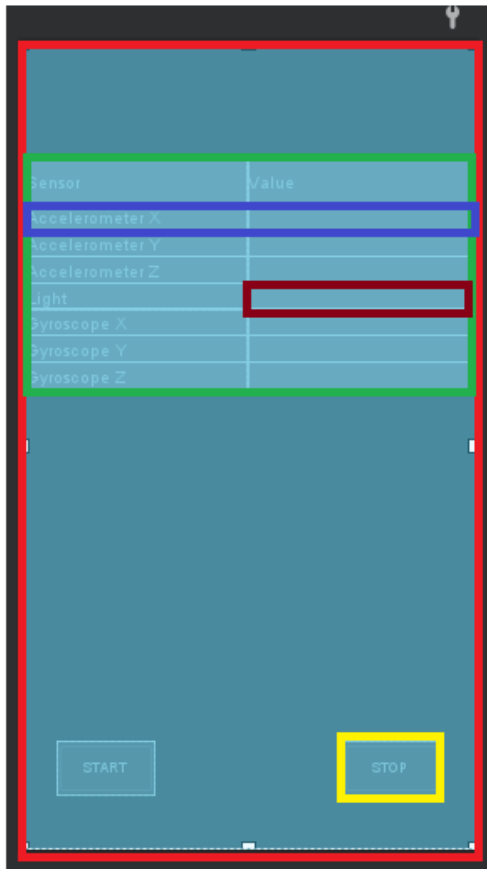
The next step is to export the configuration. At the top of the screen, select the button EXPORT and save colors.xml.

Under the project, open colors.xml and insert the value from the downloaded file.



7. Design Layout

Please design layout like this:



Constraint Layout

Table Layout

Table Row

TextView

Button

This layout was built using two types of layouts, ConstraintLayout and TableLayout

Skeleton

```
<?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="wrap_content"
android:layout_height="match_parent"
android:background="@color/design_default_color_background"
tools:context=".MainActivity">

    <TableLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="100dp"
        android:padding="5dp"
        android:stretchColumns="2"
        app:layout_constraintTop_toTopOf="parent"
        tools:context=".MainActivity"
        tools:layout_editor_absoluteX="16dp">
```



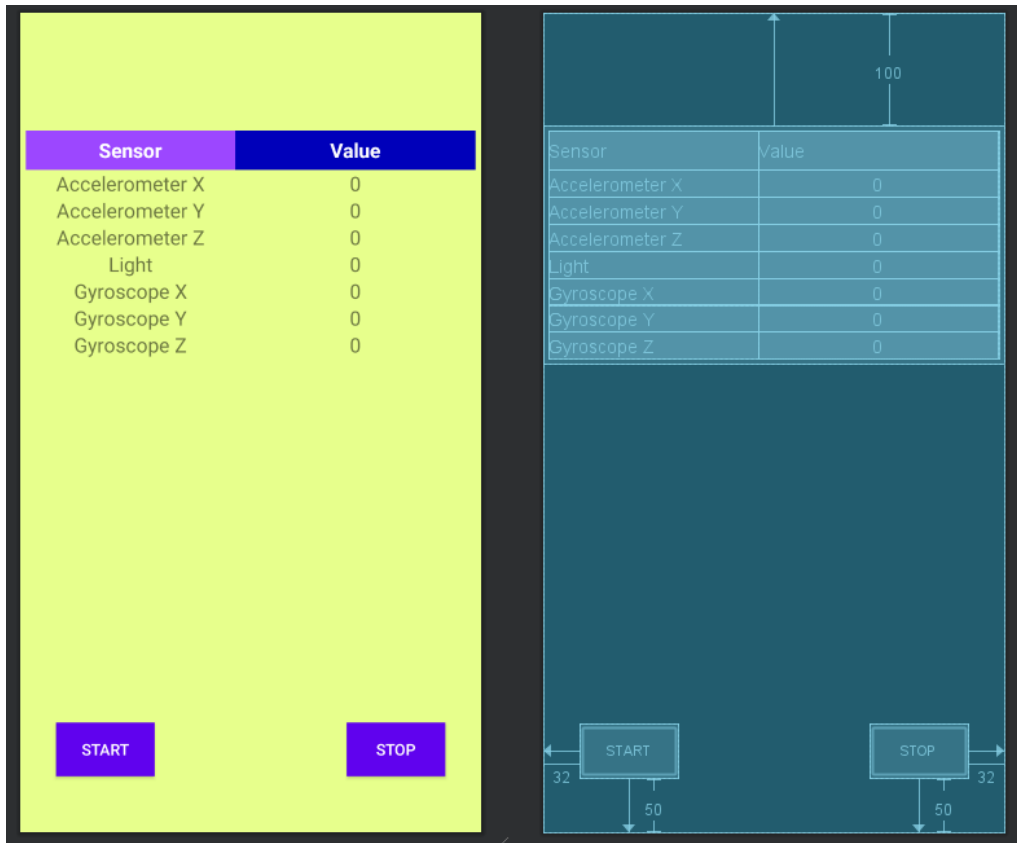
```
<TableRow
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="187dp"
        android:layout_height="match_parent"
        android:layout_column="1"
        android:background="@color/primaryLightColor"
        android:gravity="center"
        android:text="@string/sensor_name"
        android:textColor="@color/primaryTextColor"
        android:textSize="18sp"
        android:textStyle="bold" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="35dp"
        android:layout_column="2"
        android:background="@color/primaryDarkColor"
        android:gravity="center"
        android:text="@string/value_sensor"
        android:textColor="@color/primaryTextColor"
        android:textSize="18sp"
        android:textStyle="bold" />
</TableRow>
```

Please finish, this is only the beginning





8. Go to the MainActivity.kt and inside onCreate() add SensorManager and Sensor Object

```
mSensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
mAccelerometer = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)  
mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)  
mGyroscope = mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
```

Previously define objects as global for the class e.g.

```
private lateinit var mSensorManager : SensorManager  
private var mAccelerometer : Sensor ?= null  
...
```

9. Add interface to MainActivity class

```
class MainActivity : AppCompatActivity(), SensorEventListener {
```

10. Add callback methods **onAccuracyChanged**, **onSensorChanged**





```
override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) {
    print("accuracy changed")
}

override fun onSensorChanged(event: SensorEvent?) {
    if (event != null && resume) {
        if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
            findViewById<TextView>(R.id.acc_x).text = event.values[0].toString()
            findViewById<TextView>(R.id.acc_y).text = event.values[1].toString()
            findViewById<TextView>(R.id.acc_z).text = event.values[2].toString()
        }

        if (event.sensor.type == Sensor.TYPE_LIGHT) {
            findViewById<TextView>(R.id.light).text = event.values[0].toString()
        }

        if (event.sensor.type == Sensor.TYPE_GYROSCOPE) {
            findViewById<TextView>(R.id.gyro_x).text = event.values[0].toString()
            findViewById<TextView>(R.id.gyro_y).text = event.values[1].toString()
            findViewById<TextView>(R.id.gyro_z).text = event.values[2].toString()
        }
    }
}
```

Make sure that the object names in the code (R.id.XXXX) were consistent with the ID names in the layout.

11. Define auxiliary methods

```
private fun registerListener()
{
    this.resume = true

    mSensorManager.registerListener(this, mAccelerometer,
    SensorManager.SENSOR_DELAY_NORMAL)
    mSensorManager.registerListener(this, mLight,
    SensorManager.SENSOR_DELAY_NORMAL)
    mSensorManager.registerListener(this, mGyroscope,
    SensorManager.SENSOR_DELAY_NORMAL)

    changeButtonStatus()
}
private fun unregisterListener()
{
    this.resume = false

    mSensorManager.unregisterListener(this)
    changeButtonStatus()
}
private fun changeButtonStatus()
{
    findViewById<Button>(R.id.start_button).isEnabled = !resume
    findViewById<Button>(R.id.stop_button).isEnabled = resume
}
```





and also add variable **resume**

```
private var resume = false
```

12. Use previous methods to register and unregister SensorsListener.

```
override fun onResume() {  
    super.onResume()  
  
    registerListener()  
}  
  
override fun onPause() {  
    super.onPause()  
  
    unregisterListener()  
}
```

Why do we register and unregister with these methods?

13. Add onClick() methods (define also at layout)

```
fun resumeReading(view: View) {  
    registerListener()  
}  
  
fun pauseReading(view: View) {  
    unregisterListener()  
}
```

14. Run the app

Additional task

15. Definition of different colours depending on button status

Create background_button.xml and add code

```
<?xml version="1.0" encoding="utf-8"?>  
<selector xmlns:android="http://schemas.android.com/apk/res/android">  
    <item android:drawable="@color/primaryColor" android:state_enabled="true" />  
    <item android:drawable="@color/secondaryColor" android:state_enabled="false" />  
/>  
    <!-- default state -->  
    <item android:drawable="@color/primaryColor" />  
</selector>
```

16. Change definition background for each button

```
android:background="@drawable/button_background"
```

17. Run app



MVVM, LiveData

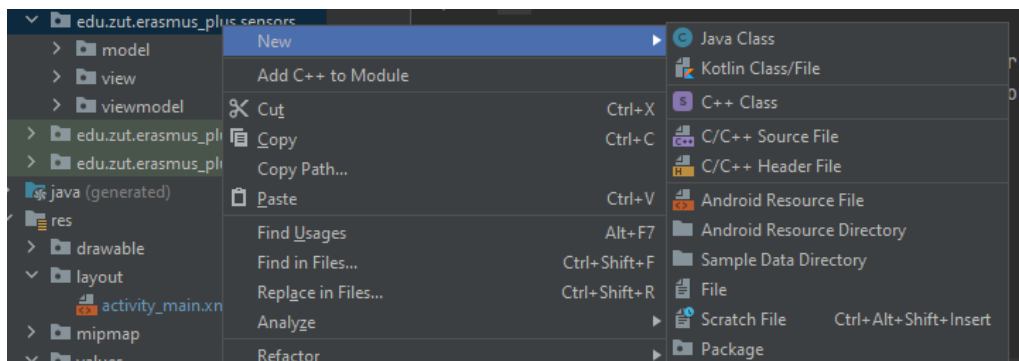
The above application shows the basic approach to programming in Android. It is now recommended that Android applications be developed using the MVVM (Model - View - ViewModel) design pattern.

The following is a solution using the Architecture Components introduced in the library AndroidX

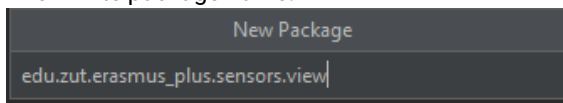
We are going to change the application we have just built into an application that follows the MVVM design pattern

1. Prepare packages
Create packages
 - **edu.zut.erasmus_plus.sensors.viewmodel**
 - **edu.zut.erasmus_plus.sensors.view**
 - **edu.zut.erasmus_plus.sensors.model**

Right click on **edu.zut.erasmus_plus.sensors** -> new -> Package



Then write package name.

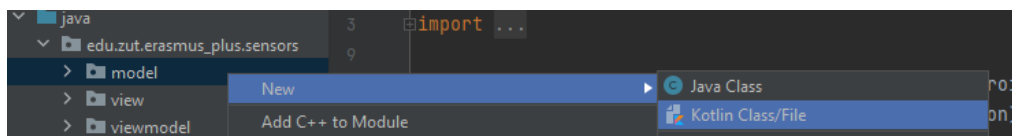


2. Move MainActivity.kt to **edu.zut.erasmus_plus.sensors.view**
All activities and fragments are classified as a view in the MVVM model. Then drag and drop MainActivity.kt to the proper package name.
3. Create Data Class
<https://kotlinlang.org/docs/data-classes.html>

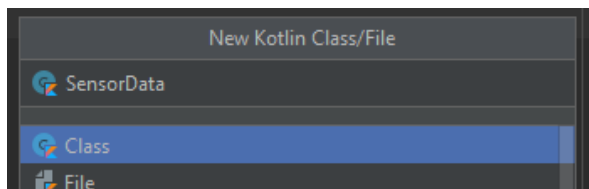
A data class is a class that contains only fields and crude methods for accessing them (getters and setters). These are simply containers for data used by other classes. These classes don't contain any additional functionality and can't independently operate on the data that they own.

- a) Create file SensorData.kt inside **edu.zut.erasmus_plus.sensors.model**

Right click on model->New-> Kotlin Class/File



Give name



b) Define class

Inside SensorData.kt define all variable

```
package edu.zut.erasmus_plus.sensors.model

data class SensorData(
    var accX: Float,
    var accY: Float,
    var accZ: Float,
    var gyroX: Float,
    var gyroY: Float,
    var gyroZ: Float,
    var light: Float
)
```

4. Data Binding

Detailed information: <https://developer.android.com/topic/libraries/data-binding>

Once the interface has been created, it is necessary to link existing objects to the code. The currently recommended approach is to use Data Binding. In addition to automatically creating the code, it also allows you to update the assembled view when the data changes using LiveData. The Data Binding Library was built with observability in mind, a pattern that has become quite popular in mobile application development. Observability is a complement to data binding, whose basic concept only considers the view and data objects. However, it is through this pattern that data can automatically propagate its changes to the view. This eliminates the need to manually update views every time new data is available, which simplifies the code base and reduces the amount of template code.

Our application will use the previously created date class to broadcast information about sensor changes.



5. Enabling Data Binding

You'll now enable data binding in the project. Open the app's build.gradle file, and add this line inside android tag.

```
android {  
    compileSdk 31  
  
    dataBinding {  
        enabled true  
    }  
}
```

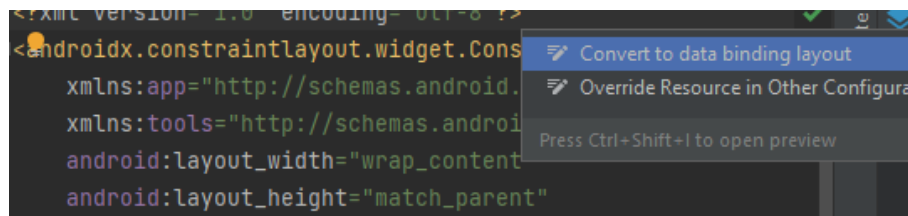
Now convert the layout to a Data Binding layout.

To convert a regular layout to Data Binding layout:

1. Wrap your layout with a <layout> tag
2. Add layout variables (optional)
3. Add layout expressions (optional)

Or

Android Studio offers a handy way to do this automatically: Right-click the root element, select Show Context Actions, then Convert to data binding layout:





```
<?xml version="1.0" encoding="utf-8"?>
<layout 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">

    <data>

    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:background="@color/cardview_shadow_start_color"
        tools:context=".view.MainActivity">
```

The **<data>** tag will contain layout variables. We will add values there later

Layout variables are used to write **layout expressions**. Layout expressions are placed in the value of element attributes, and they use the *@{expression}* format. For example:

```
android:text="@{String.valueOf(index + 1)}"
android:visibility="@{age < 13 ? View.GONE : View.VISIBLE}"
android:transitionName="@{"image_" + id}"

// Bind the name property of the viewmodel to the text attribute
android:text="@{viewmodel.name}"
// Bind the nameVisible property of the viewmodel to the visibility attribute
android:visibility="@{viewmodel.nameVisible}"
// Call the onLike() method on the viewmodel when the View is clicked.
android:onClick="@{() -> viewmodel.onLike()}">
```

More info about layout expression https://developer.android.com/topic/libraries/data-binding/expressions#expression_language

6. Add dependencies inside build.gradle(app)

```
dependencies {
    //ViewModel
    implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.1'
    implementation 'androidx.activity:activity-ktx:1.4.0'
    //Lifecycle
    implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.4.1"
```

7. Create class to reading sensors value and move all the code responsible for collecting values for the sensors to a separate class
 - a) Create file **SensorDataLiveData.kt** inside **edu.zut.erasmus_plus.sensors.model**
 - b) Move code from **Activity_Main.kt** (all function except **onCreate()**)
 - c) Remove unnecessary objects from the class and the code responsible for creating Sensor Manager instances and sensor objects.

After this, MainActivity.kt looks like this:





```
package edu.zut.erasmus_plus.sensors

import ...

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Practically all existing code has been removed.

SensorDataLiveData class looks like this



```
class SensorDataLiveData (
    context: Context,
    private val sensorDelay: Int = SensorManager.SENSOR_DELAY_UI
) : LiveData<SensorData>(), SensorEventListener {

    private val mSensorManager: SensorManager =
        context.getSystemService(Context.SENSOR_SERVICE) as SensorManager
    private val accelerometer: Sensor =
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    private val gyroscope: Sensor =
        mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
    private val light: Sensor =
        mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)

    private val mAccelerometerReading = FloatArray(3)
    private val mGyroscopeReading = FloatArray(3)
    private val mLightReading = FloatArray(1)

    override fun onActive() {
        super.onActive()
        registerListeners()
    }
    override fun onInactive() {
        super.onInactive()
        unregisterListeners()
    }
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {}

    override fun onSensorChanged(event: SensorEvent) {
        if (event.sensor == accelerometer) {
            System.arraycopy(event.values, 0, mAccelerometerReading, 0,
                mAccelerometerReading.size)
        } else if (event.sensor == gyroscope) {
            System.arraycopy(event.values, 0, mGyroscopeReading, 0,
                mGyroscopeReading.size)
        } else if (event.sensor == light) {
            System.arraycopy(event.values, 0, mLightReading, 0,
                mLightReading.size)
        }

        value = SensorData(
            mAccelerometerReading[0], mAccelerometerReading[1],
            mAccelerometerReading[2],
            mGyroscopeReading[0], mGyroscopeReading[1],
            mGyroscopeReading[2],
            mLightReading[0]
        )
    }

    fun unregisterListeners() {
        mSensorManager.unregisterListener(this)
    }

    fun registerListeners() {
        mSensorManager.registerListener(
            this,
            accelerometer,
            SensorManager.SENSOR_DELAY_NORMAL,
            sensorDelay
        )
        mSensorManager.registerListener(
```





Now, this class is responsible for handling the sensors. Our goal is to have every change be communicated using observers. We will use the **LiveData** class for this purpose. Changes will observe changes on the previously created **SensorData** model. The class definition takes the following form.

```
class SensorDataLiveData(context: Context) : LiveData<SensorData>(),  
SensorEventListener
```

With the methods from the LiveData class (**onActive()** and **onInactive()**) it can start or stop collecting events from the sensors. The **onActive()** method is run when the first observer joins our class and the second when the last one disconnects.

The rest of the code is consistent with what was previously implemented in **MainActivity()**, except that in the **onSensorChanged()** method, the **value** object returns the result. In our case, it is an object of **SensorData** class with the last values from sensor readings.

8. Create ViewModel

The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.

Create file `SensorViewModel.kt` inside `edu.zut.erasmus_plus.sensors.viewmodel`

Inside file adds two private variables and their get methods.

```
class SensorViewModel(application: Application): AndroidViewModel(application) {  
    private val _sensor = SensorDataLiveData(application)  
    private var _pauseReading = MutableLiveData<Boolean>()  
  
    val sensor: LiveData<SensorData>  
        get() = _sensor  
  
    fun getPauseReading(): MutableLiveData<Boolean> {  
        return _pauseReading  
    }  
}
```

The first variable is used to read sensor values, from the previously created `SensorDataLiveData` class, where it is necessary to pass the application context. Hence, we modify the class definition as above. Please note that our class inherits from `AndroidViewModel`.





The **_pauseReading** variable determines if we stop/start reading the sensors. It needs to be initialised, so we add the following code to the SensorViewModel:

```
init {  
    _pauseReading = MutableLiveData(false)  
}
```

The last method that needs to be added is the correct response to stop/start sensor reading. Add the following function to the SensorViewModel:

```
fun changeButtonStatus()  
{  
    if(_pauseReading.value==true) _sensor.registerListeners()  
    else _sensor.unregisterListeners()  
    _pauseReading.value?.let {  
        _pauseReading.value = !it  
    }  
}
```

9. Change layout

After creating VM, we must change layout and add a link to the VM

- a) Add information about variable

Open activity_mail.xml

Inside properties **<data>** **</data>** insert new variable

```
<data>  
    <variable  
        name="sensorViewModel"  
        type="edu.zut.erasmus_plus.sensors.viewmodel.SensorViewModel" />  
</data>
```

Adding variable (**sensorViewModel**) will allow you to use objects from the **SensorViewModel** class

- b) Change properties inside activity_main.xml

Find the **TextView** object responsible for displaying the **acc_X** value (around line 64) and change the **android:text** property.

```
<TextView  
    android:id="@+id/acc_X"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_column="2"  
    android:text="@{String.valueOf(sensorViewModel.sensor.accX)}"  
    android:textAlignment="center"  
    android:textSize="18sp" />
```





- c) Change the **android:text** property for all objects representing the reading value from the sensors (7 times)
- d) Change onClick and enable properties for Button

Change as follow.

```
<Button
    android:id="@+id/start_button"
    . . .
    android:enabled="@{sensorViewModel.pauseReading}"
    android:onClick="@{() ->sensorViewModel.changeButtonStatus()}"
    . . .
/>

<Button
    android:id="@+id/stop_button"
    . . .
    android:enabled="@{!sensorViewModel.pauseReading}"
    android:onClick="@{() ->sensorViewModel.changeButtonStatus()}"
    . . .
/>
```

Only the changed parts are shown above.

10. Change view – MainActivity.kt

- a) Open MainActivity.kt
- b) Inside onCreate, change code like this:

```
class MainActivity : AppCompatActivity() {
    private var resume = false

    private lateinit var binding: ActivityMainBinding
    private val sensorViewModel: SensorViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        requestWindowFeature(Window.FEATURE_NO_TITLE)
        requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT

        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.sensorViewModel = sensorViewModel
        binding.lifecycleOwner = this
    }
}
```

Notice that we added two variables, binding and **sensorViewModel**. The binding variable uses DataBinding and provides us with automatic access to variables in the layout without using findViewById().

The sensorViewModel variable binds the created ViewModel to the view.





11. Run Application

The above steps allowed us to use the now recommended model of creating applications using the Android Components defined in the androidX library.

Výsledný kód aplikace najdete na adrese https://github.com/matam/Erasmus_Lab3-4.



Using Data Base and Recycler View

Data storage is one of the most commonly used functionalities in an application. If the data has structure, then relational databases are used to store the data. In the Android environment, the database is implemented using the SQLite database, but its direct use is quite complicated. The AndroidX library introduced the Room framework, which made the use of the database much easier.

Dynamic View objects display data whose number of elements is variable, one of the most commonly used being RecyclerView.

This document presents a database application for storing information about books.

These labs include the following elements:

- Build new app and configure build.grade
- Create data model
- Create Database instance
- Create RecyclerView
- Bind data from database to RecyclerView

1. Create New project -> Empty Activity

2. Define App Name (**Lab4**) and Package (**edu.zut.wi.erasmus.lab4**), chose minimum API (**API28**)

3. Add dependencies to the **build.grade**

```
def room_version = "2.3.0"
def lifecycle_version = "2.3.1"
implementation("androidx.room:room-runtime:$room_version")
annotationProcessor "androidx.room:room-compiler:$room_version"
// To use Kotlin annotation processing tool (kapt)
kapt("androidx.room:room-compiler:$room_version")
// To use Kotlin Symbolic Processing (KSP)
//ksp("androidx.room:room-compiler:$room_version")
// optional - Kotlin Extensions and Coroutines support for Room
implementation "androidx.lifecycle:lifecycle-livedata-
ktx:$lifecycle_version"
implementation("androidx.room:room-ktx:$room_version")
// optional - RxJava2 support for Room
implementation "androidx.room:room-rxjava2:$room_version"
// optional - RxJava3 support for Room
implementation "androidx.room:room-rxjava3:$room_version"
// optional - Guava support for Room, including Optional and
ListenableFuture
implementation "androidx.room:room-guava:$room_version"
// optional - Test helpers
testImplementation("androidx.room:room-testing:$room_version")
// optional - Paging 3 Integration
implementation("androidx.room:room-paging:2.4.0-alpha04")
and plugin
```

```
id "org.jetbrains.kotlin.kapt"
```



4. Define new package: database(**edu.zut.wi.erasmus.lab4.database**)
5. Add data class inside new package **Book.kt**

```
package edu.zut.wi.erasmus.lab4.database

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity
data class Book(
    @PrimaryKey val uid: Int,
    @ColumnInfo(name = "title") val title: String?,
    @ColumnInfo(name = "subtitle") val subtitle: String?,
    @ColumnInfo val publisher: String?
)
```

6. Create Dao –interface – **BookDao.kt**

```
@Dao
interface BookDao {
    @Query("SELECT * FROM book ORDER BY title ASC")
    fun getAlphabetizedBooks(): Flow<List<Book>>

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(book: Book)

    @Query("DELETE FROM book")
    suspend fun deleteAll()
}
```

7. Implement the Room database – **BookRoomDatabase.kt**

```
@Database(entities = arrayOf(Book::class), version = 1, exportSchema = false)
public abstract class BookRoomDatabase: RoomDatabase() {

    abstract fun bookDao(): BookDao

    companion object {
        // Singleton prevents multiple instances of database opening at the
        // same time.
        @Volatile
        private var INSTANCE: BookRoomDatabase? = null

        fun getDatabase(
            context: Context,
            scope: CoroutineScope
        ): BookRoomDatabase {
            // if the INSTANCE is not null, then return it,
            // if it is, then create the database
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    BookRoomDatabase::class.java,
                    "book_database"
                ).addCallback(BookDatabaseCallback(scope))
                    .build()
                INSTANCE = instance
            }
        }
    }
}
```

```
        // return instance
        instance
    }
}
}
suspend fun populateDatabase(bookDao: BookDao) {
    // Delete all content here.
    bookDao.deleteAll()

    // Add sample words.
    var book = Book(title = "new title 1",publisher = "publisher
1",subtitle = "subtitle 1")
    bookDao.insert(book)
    book = Book(title = "new title 2",publisher = "publisher
2",subtitle = "subtitle 2")
    bookDao.insert(book)
    book = Book(title = "new title 3",publisher = "publisher
3",subtitle = "subtitle 3")
    bookDao.insert(book)
    book = Book(title = "new title 4",publisher = "publisher
4",subtitle = "subtitle 4")
    bookDao.insert(book)
    book = Book(title = "new title 5",publisher = "publisher
5",subtitle = "subtitle 5")
    bookDao.insert(book)

}

private class BookDatabaseCallback(
    private val scope: CoroutineScope
) : RoomDatabase.Callback() {

    override fun onCreate(db: SupportSQLiteDatabase) {
        super.onCreate(db)
        INSTANCE?.let { database ->
            scope.launch {
                database.populateDatabase(database.bookDao())
            }
        }
    }
}
}
```

8. Implementing the Repository – BookRepository.kt

```
class BookRepository(private val bookDao: BookDao) {
    val allBooks: Flow<List<Book>> = bookDao.getAlphabetizedBooks()

    @WorkerThread
    suspend fun insert(book: Book) {
        bookDao.insert(book)
    }
}
```

9. Create new package : viewmodel(**edu.zut.wi.erasmus.lab4.viewmodel**)

10. Implement the ViewModel and ViewModelFactory – BookViewModel.kt

```
class BookViewModel(private val repository: BookRepository) : ViewModel()
{
    val allWords: LiveData<List<Book>> = repository.allBooks.asLiveData()
```



```
        fun insert(book: Book) = viewModelScope.Launch {
            repository.insert(book)
        }
    }

    class BookViewModelFactory(private val repository: BookRepository) :
        ViewModelProvider.Factory {
        override fun <T : ViewModel> create(modelClass: Class<T>): T {
            if (modelClass.isAssignableFrom(BookViewModel::class.java)) {
                @Suppress("UNCHECKED_CAST")
                return BookViewModel(repository) as T
            }
            throw IllegalArgumentException("Unknown ViewModel class")
        }
    }
}
```

Part II . Implement RecyclerView

11. Create layout for recycler item (recyclerview_item.xml)

```
<?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="wrap_content"
    android:background="@color/design_default_color_background">

    <TextView
        android:id="@+id/idBook"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:minWidth="100dp"
        style="@style/book_id"
        android:text="TextView" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="11"
        android:orientation="vertical">

        <TextView
            android:id="@+id/title"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/book_title"
            android:text="TextView" />

        <TextView
            android:id="@+id/subtitle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/book_subtitle"
            android:text="TextView" />

        <TextView
            android:id="@+id/publisher"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/book_publisher"
            android:text="TextView" />
    </LinearLayout>
</LinearLayout>
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/book_publisher"
        android:text="TextView" />
    </LinearLayout>
</LinearLayout>
```

12. Add to the `activity_main.xml` code for RecyclerView and FAB

```
<?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">

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerview"
        android:layout_width="0dp"
        android:layout_height="0dp"
        tools:listitem="@Layout/recyclerview_item"

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

    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/fab"
        android:src="@drawable/ic_baseline_add_24"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="16dp"
        android:contentDescription="@string/add_book"/>

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

13. Create new package : adapter(**`edu.zut.wi.erasmus.lab4.adapter`**) and create new class `BookListAdapter.kt`

```
class BookListAdapter : ListAdapter<Book,
BookListAdapter.BookViewHolder>(BooksComparator()) {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
BookViewHolder {
        return BookViewHolder.create(parent)
    }

    override fun onBindViewHolder(holder: BookViewHolder, position: Int) {
        val current = getItem(position)

        holder.bind(current.uid.toString(),current.title,current.subtitle,current.p
ublisher)
    }
}
```

```
    }

    class BookViewHolder(itemView: View) :
RecyclerView.ViewHolder(itemView) {
        private val titleBook: TextView = itemView.findViewById(R.id.title)
        private val subtitleBook: TextView =
itemView.findViewById(R.id.subtitle)
        private val idBook: TextView = itemView.findViewById(R.id.idBook)
        private val publisherBook: TextView =
itemView.findViewById(R.id.publisher)

        fun bind(id: String?,title: String?,subtitle: String?,publisher:
String?) {
            idBook.text = id
            titleBook.text = title
            subtitleBook.text = subtitle
            publisherBook.text = publisher
        }

        companion object {
            fun create(parent: ViewGroup): BookViewHolder {
                val view: View = LayoutInflater.from(parent.context)
                    .inflate(R.layout.recyclerview_item, parent, false)
                return BookViewHolder(view)
            }
        }
    }

    class BooksComparator : DiffUtil.ItemCallback<Book>() {
        override fun areItemsTheSame(oldItem: Book, newItem: Book): Boolean
{
            return oldItem === newItem
        }

        override fun areContentsTheSame(oldItem: Book, newItem: Book):
Boolean {
            return oldItem.title == newItem.title
        }
    }
}
```

14. Call adapter inside MainActivity.kt
Put below **setContentView**

```
val recyclerView = findViewById<RecyclerView>(R.id.recyclerview)
val adapter = BookListAdapter()
recyclerView.adapter = adapter
recyclerView.layoutManager = LinearLayoutManager(this)
```

15. Create Application Instants and create instants the repository and the database, create class **BookApplication.kt**

```
class BooksApplication: Application() {
    val applicationScope = CoroutineScope(SupervisorJob())
    // Using by lazy so the database and the repository are only created
when they're needed
    // rather than when the application starts
    val database by lazy {
```

```
BookRoomDatabase.getDatabase(this,applicationScope) }  
    val repository by Lazy { BookRepository(database.bookDao())}  
}
```

16. Change AndroidManifest.xml add line inside <application ...

```
android:name=".BooksApplication"
```

17. Run App

18. Implement Add New Book Activity – NewBookActivity.kt – create using wizard

```
class NewBookActivity : AppCompatActivity() {  
    private val bookViewModel: BookViewModel by viewModels {  
        BookViewModelFactory((application as BooksApplication).repository)  
    }  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_new_book)  
        val editTitle = findViewById<EditText>(R.id.edit_title)  
        val editSubTitle = findViewById<EditText>(R.id.edit_subtitle)  
        val editPublisher = findViewById<EditText>(R.id.edit_publisher)  
  
        val button = findViewById<Button>(R.id.button_save)  
        button.setOnClickListener {  
            val replayIntent = Intent()  
            if(TextUtils.isEmpty(editTitle.text)){  
                setResult(Activity.RESULT_CANCELED,replayIntent)  
            } else {  
                bookViewModel.insert(Book(title =  
editTitle.text.toString(),publisher =  
editPublisher.text.toString(),subtitle = editSubTitle.text.toString()))  
            }  
            finish()  
        }  
    }  
}
```

19. Improve layout – activity_new_book.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout 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:orientation="vertical"  
    tools:context=".NewBookActivity">  
  
    <EditText  
        android:id="@+id/edit_title"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:minHeight="@dimen/min_height"  
        android:fontFamily="sans-serif-light"  
        android:hint="@string/hint_title"  
        android:inputType="textAutoComplete"  
        android:layout_margin="@dimen/big_padding"  
        android:textSize="18sp" />  
  
    <EditText
```

```
        android:id="@+id/edit_subtitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="@dimen/min_height"
        android:fontFamily="sans-serif-light"
        android:hint="@string/hint_subtitle"
        android:inputType="textAutoComplete"
        android:layout_margin="@dimen/big_padding"
        android:textSize="18sp" />

        <EditText
            android:id="@+id/edit_publisher"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="@dimen/big_padding"
            android:fontFamily="sans-serif-light"
            android:hint="@string/hint_publisher"
            android:inputType="textAutoComplete"
            android:minHeight="@dimen/min_height"
            android:textSize="18sp"
            android:autofillHints="@string/hint_publisher" />

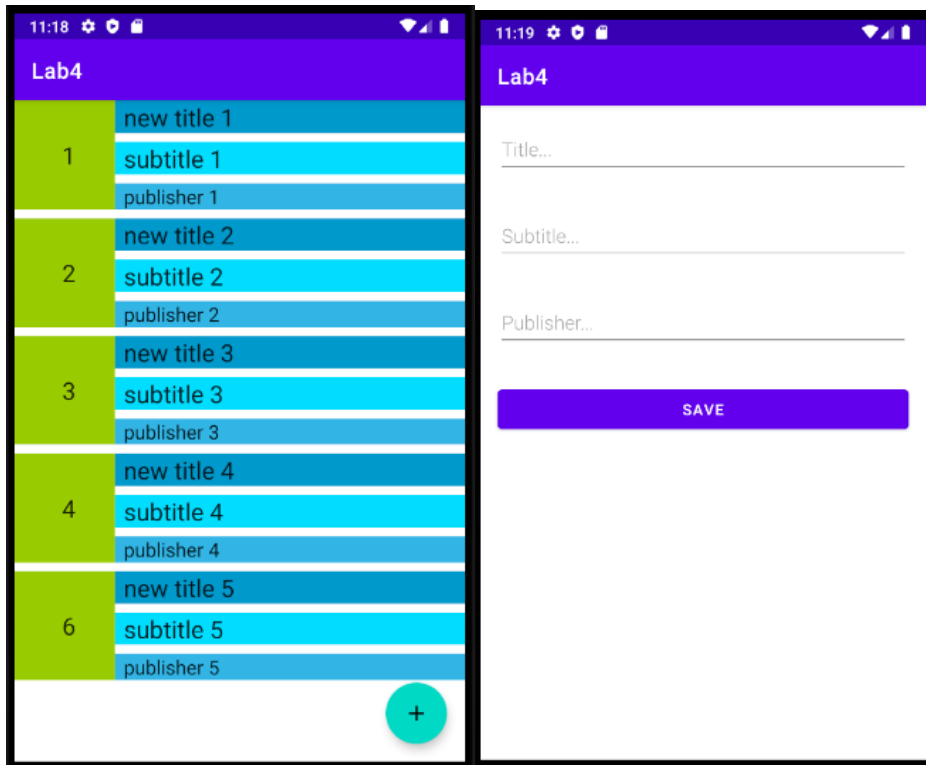
        <Button
            android:id="@+id/button_save"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/button_save"
            android:layout_margin="@dimen/big_padding" />
    </LinearLayout>
```

20. Add code to FloatingActionButton – MainActivity.kt

```
val fab = findViewById<FloatingActionButton>(R.id.fab)
fab.setOnClickListener {
    val intent = Intent(this@MainActivity, NewBookActivity::class.java)
    startActivity(intent)
}
```

21. Run Application

Final Application



Main Activity

New Book Activity

Additional information: <https://developer.android.com/codelabs/android-room-with-a-view-kotlin>

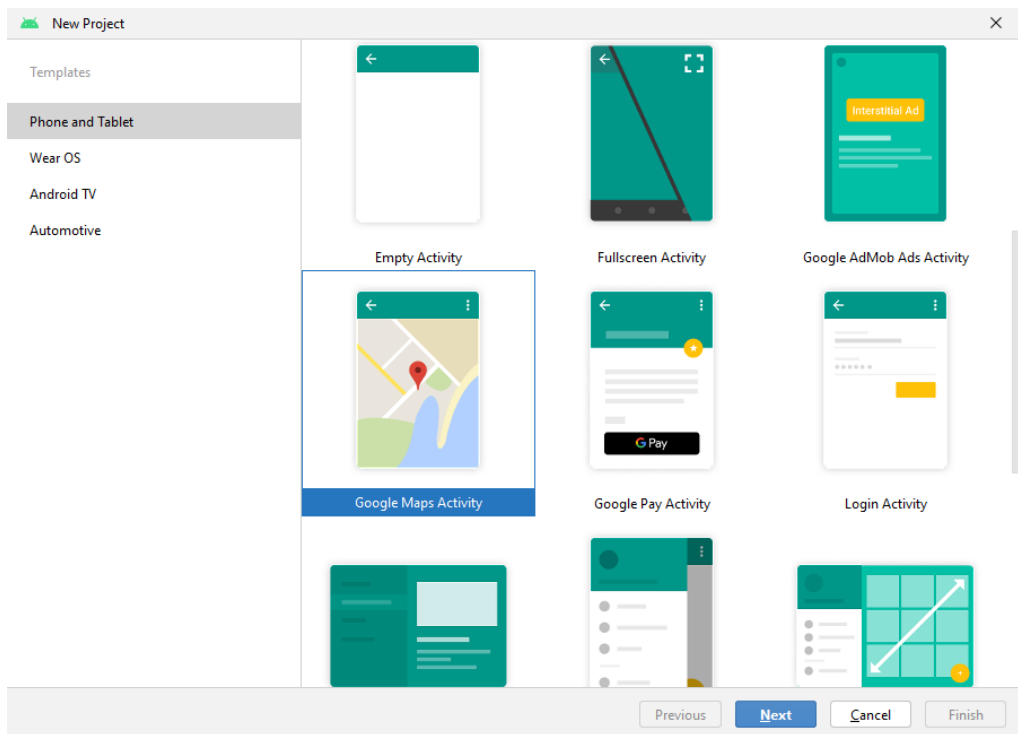
The resulting application code can be found at https://github.com/matam/Erasmus_Lab5.

Localization

Localisation provides the opportunity to create an application that takes into account the context of the user's location. It allows the creation of content that will dynamically change depending on the position of the device. Location also allows developers to collect statistics, facilitates profiling, but there is some privacy risk. Access to location requires the user to grant the app permissions.

This document describes an application that displays a user's position on a continuous or one-off basis. In addition, the process of granting and verifying permissions will be described. The application uses an off-the-shelf design that displays a map.

1. Create project from wizard



Select a Google Maps activity.

2. Familiarise yourself with the code. Notice what interfaces are implemented, analyse the methods created
3. Run the application.
When launched, a map with a marker in Sydney should show up. However, it does not show up due to the lack of a valid API key.

In Run we have information about the absence of a valid key



```
Run: app
D/Inp...
D/InputMethodManager: startInputInner - Id : 0
D/InsetsController: onStateChanged: InsetsState: {mDisplayFrame=Rect(0, 0 - 1600, 2560), mDisplayCutou
E/Google Maps Android API: Authorization failure. Please see https://developers.google.com/maps/docum
E/Google Maps Android API: In the Google Developer Console (https://console.developers.google.com)
  Ensure that the "Google Maps Android API v2" is enabled.
  Ensure that the following Android Key exists:
  API Key: YOUR_API_KEY
```

4. Adding the right key to the project: Exact information is at the following address

<https://developers.google.com/maps/documentation/android-sdk/get-api-key>

- a. Insert your API_KEY into AndroidManifest.xml
 - b. Configure API_KEY – add Android Maps
5. We will now move on to adding a device position display to the app.
 6. We will use Google's recommended use of Google Play Services for the location

How add Google Play services - <https://developers.google.com/android/guides/setup>

Add dependencies to build.gradle (choose the latest version)

apply plugin: 'com.android.application'

...

```
dependencies {
    implementation 'com.google.android.gms:play-services-location:21.0.1'
}
```

7. We will change the **activity_main.xml** graphic design to add two buttons to it.





```
<?xml version="1.0" encoding="utf-8"?>
<android.widget.LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center">

    <androidx.fragment.app.FragmentContainerView
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/map"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        tools:context=".MapsActivity"
    />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:orientation="horizontal">

        <Button
            android:id="@+id/btGetLastPosition"
            android:onClick="getLastPos"
            android:text="@string/get_position"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:layout_margin="10dp"/>

        <Button
            android:id="@+id/btContinousPosition"
            android:onClick="startStopRequestLocation"
            android:text="@string/start_loop"
            android:layout_height="match_parent"
            android:layout_width="match_parent"
            android:layout_weight="1"
            android:layout_margin="10dp"/>

    </LinearLayout>
</android.widget.LinearLayout>
```

Correct errors by adding relevant definitions to the file **strings.xml** (get_position-"Get Position", start_loop -> „Start Loop”), also add a new definition stop_loop ->”Stop_Loop”

8. Get last known location - <https://developer.android.com/training/location/retrieve-current>

The first step will be to add some code to determine the last known position. We will hook this function up to the "Get Position" button, the handling method will be called **getLastPos()**.

We will add this method to the main activity file.





```
fun getLastPos(view: View)
{
    //checkPermission()
    fusedLocationClient.lastLocation
        .addOnSuccessListener { location : Location? ->
            val myPosition = location?.let {
                LatLng(it.latitude, it.longitude)
            }
            myPosition?.let {
                mMap.addMarker(MarkerOptions().position(myPosition).title("My position"))
                mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))
            }
        }
}
```

and add the object definition in class **MapsActivity**

```
private lateinit var fusedLocationClient: FusedLocationProviderClient
```

The object should be initialised. Add the following code to the **onCreate()** method.

```
fusedLocationClient = LocationServices.getFusedLocationProviderClient(this)
```

The code above does not include a permission check. According to the rules, a permission check is required before each use of functionality that must have permission from the user.

9. Specify app permission

The first step is to add information to the manifest file about what permissions are necessary for the application to run. In our case, we will add both coarse and fine-grained permissions. Note in which section of the manifest the permissions are added.

```
<manifest ... >
<!-- Always include this permission -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<!-- Include only if your app benefits from precise location access. -->
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

10. Now you can move on to creating a function that will verify the entitlements.





```
private fun checkPermission()
{
    if (ContextCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_FINE_LOCATION
    ) != PackageManager.PERMISSION_GRANTED)

requestPermissionLauncher.launch (Manifest.permission.ACCESS_FINE_LOCATION)
}
private val requestPermissionLauncher =
    registerForActivityResult (
        ActivityResultContracts.RequestPermission ()
    ) {
        isGranted: Boolean ->
        if (isGranted) {
            Log.i ("Permission: ", "Granted")
        } else {
            Log.i ("Permission: ", "Denied")
        }
    }
}
```

The method given above is the simplest and only checks permissions once, in case the user rejects permanently do not handle this situation. As an additional task, implement a full permission query cycle according to:

<https://developer.android.com/guide/topics/permissions/overview#workflow>

11. It is now possible to edit the **getLastPos()** method and comment out the permission check.
12. Run the application
13. We will now add support for continuous item updates. Also we will be using Google Play services. The task of this feature will be to continuously update the position marker on the map. For this purpose, a return method will be created, whose task will be to update the position when a new position is received. We need to define the following objects in **MapsActivity()**, they are to be available to all methods in the class.

```
private var isRequestLoacation: Boolean = false
private lateinit var locationRequest: LocationRequest
private lateinit var locationCallback: LocationCallback
```

14. In the **onCreate()** method, we will now initialise the methods for the location.
The first object, is used to specify the location parameters, while the second defines the return methods. Note the update frequency.





```
locationRequest = LocationRequest.Builder(Priority.PRIORITY_HIGH_ACCURACY,
    500)
    .build()

locationCallback = object : LocationCallback() {
    override fun onLocationResult(locationResult: LocationResult) {
        if (locationResult != null) {
            super.onLocationResult(locationResult)
            locationResult.lastLocation?.let {
                val myPosition = it?.let {
                    LatLng(it.latitude, it.longitude)
                }
                myPosition?.let {
                    mMap.addMarker(MarkerOptions().position(myPosition).title("My position"))
                    mMap.moveCamera(CameraUpdateFactory.newLatLng(myPosition))
                }
            }
        }
    }
}
```

15. Now we can call the location request, we will do this in the **startStopRequestLocation()** method that was created to handle the second button

First we check the permissions, then based on the status we will check whether to start the position update task or stop it. In our case, we are just adding a point to the map. The following code can be optimised by creating common add methods





```
fun startStopRequestLocation(view: View)
{
    checkPermission()
    if (!isRequestLoacation)
    {
        binding.btContinousPosition.text=getString(R.string.stop_loop)
        val addTask=
        fusedLocationClient.requestLocationUpdates(locationRequest,
        locationCallback, Looper.myLooper())
        addTask.addOnCompleteListener {task->
            if (task.isSuccessful) {
                Log.d("startStopRequestLocation", "Start loop Location
                Callback.")
            } else {
                Log.d("startStopRequestLocation", "Failed start Location
                Callback.")
            }
        }
    }
    else
    {
        binding.btContinousPosition.text=getString(R.string.start_loop)
        val removeTask =
        fusedLocationClient.removeLocationUpdates(locationCallback)
        removeTask.addOnCompleteListener { task ->
            if (task.isSuccessful) {
                Log.d("startStopRequestLocation", "Location Callback
                removed.")
            } else {
                Log.d("startStopRequestLocation", "Failed to remove
                Location Callback.")
            }
        }
    }
    isRequestLoacation=!isRequestLoacation
}
```

16. Start the application. If you are executing the application on an emulator, the position can be changed in the virtual machine settings.
17. The initial project as well as the final project is placed in the repository https://github.com/matam/Erasmus_Lab6

Task:

1. Carry out the attribution process according to the recommended workflow
2. Show only the last 5 markers on the map.





Networking

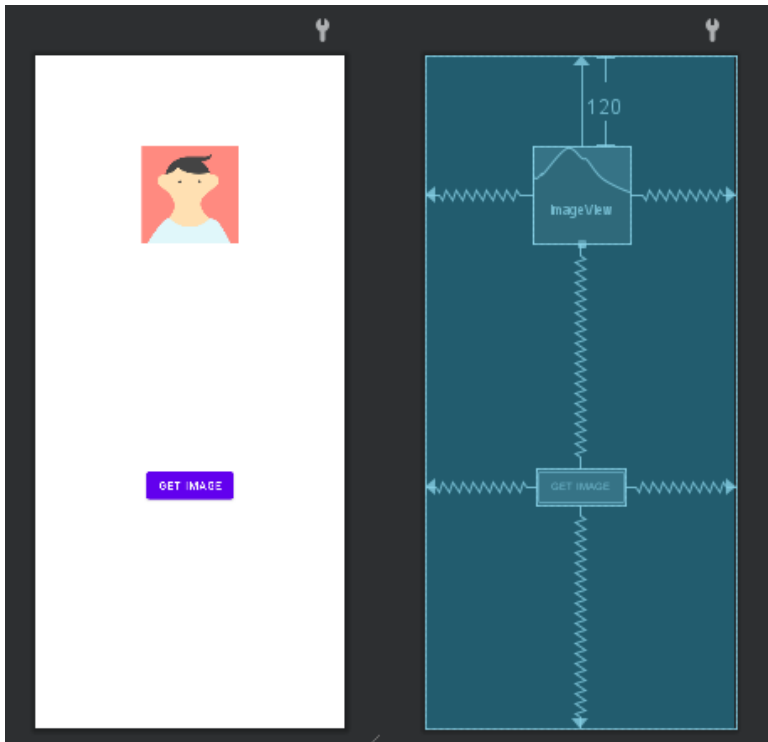
When developing applications, there is often a need to use network connections. Network requests are used to download or update data. The use of the network can generate considerable costs, as well as being a particular threat to the user's privacy, which is why permission to use the network is necessary, and the perform network operations and read network status in your application, your manifest must include the permissions. Performing network operations requires them to be completed in a separate thread not to burden the main thread. Before connecting, it is good practice to check that the device is connected to the Internet. Network checks must be performed before any download operation.

Currently, most applications use a given service's REST API to retrieve data. Sometimes, however, it is necessary to retrieve a file from the server in the traditional way. The HTTP/HTTPS protocol is then preferred.

In this laboratory, we develop a program that downloads images using API NASA (Astronomy Picture of the day) - <https://api.nasa.gov/#browseAPI> .

1. To realize this laboratory, we need Generate API KEY - <https://api.nasa.gov/#signUp> . Please signup and store the received **API KEY**.
2. Create a New project -> Empty Activity
3. Define App Name (**Lab7**) and Package (**edu.zut.erasmus_plus.networking**), choose minimum API (**API28**)
4. Explore the code
 1. Locate MainActivity.kt, activity_main.xml, AndroidManifest.xml
5. Go to build.gradle -> Upgrade all dependencies and libraries for Project and Module (we can skip)
6. Design layout like this:





The application screen will contain two elements, a button to download data, and a photo downloaded from the internet.

7. Run App

8. Define permissions at AndroidManifest.xml

To use the Internet, you need to define the appropriate permission; in addition, to check the status of the network you also need to define the permission.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="edu.zut.erasmus_plus.networking" >
...
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>

    <application>
...
    </application>
</manifest>
```

9. Create a method to check the network connection



The following code will be useful to check if there is an internet connection. Please use this method before getting file from internet

```
private fun isNetworkConnected(): Boolean {
    val connectivityManager =
getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
    val activeNetwork = connectivityManager.activeNetwork
    val networkCapabilities =
connectivityManager.getNetworkCapabilities(activeNetwork)
    return networkCapabilities != null &&
networkCapabilities.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERN
ET)
}
```

10. Retrofit

Retrofit is an Android and Java library that excels at retrieving and uploading structured data, such as JSON and XML. This library makes HTTP requests using **OkHttp**, another library from Square. Using this library we define service, data class and repository.

The use of the library consists of defining the data (data class), the service defining the queries (interface) and the calling method. We will first define the dependencies.

11. Define dependencies

```
//Retrofit
implementation 'com.squareup.retrofit2:retrofit:2.9.0'

// JSON Converter
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
```

12. Define data class

The data from the API will be structured, and returned as JSON. The creation of a Data Class will facilitate the use of this data. A description of the individual fields is provided under

<https://github.com/nasa/apod-api>





```
data class AstronomyPictureDayEntity(  
    @SerializedName("copyright")  
    val copyright: String?,  
    @SerializedName("date")  
    val date: String?,  
    @SerializedName("explanation")  
    val explanation: String?,  
    @SerializedName("hdurl")  
    val hdurl: String?,  
    @SerializedName("media_type")  
    val mediaType: String?,  
    @SerializedName("service_version")  
    val serviceVersion: String?,  
    @SerializedName("title")  
    val title: String?,  
    @SerializedName("url")  
    val url: String?  
)
```

13. Define service

There will be a two-stage process for downloading the image, first an object describing the image of the day will be downloaded along with the address from where the image can be downloaded. The second stage involves downloading the image. Therefore, the service contains two download methods defined.

```
//Nasa Astrology picture of the day  
interface ApodService {  
    companion object {  
        const val BASE_URL_APOD_ITEM = "https://api.nasa.gov/"  
        const val BASE_URL_APOD_IMAGE = "https://apod.nasa.gov/"  
        const val API_KEY = "YIm2xWGBYbtM12tptMjEGJZqxEIyONsd2hC6h21B"  
        fun getApodItem(): ApodService {  
            val retrofit = Retrofit.Builder()  
                .addConverterFactory(GsonConverterFactory.create())  
                .baseUrl(BASE_URL_APOD_ITEM)  
                .build()  
            return retrofit.create();  
        }  
        fun getApodImage(): ApodService {  
            val retrofit = Retrofit.Builder()  
                .baseUrl(BASE_URL_APOD_IMAGE)  
                .build()  
            return retrofit.create();  
        }  
    }  
    @GET  
    fun downloadImageUrl(@Url fileUrl: String): Call<ResponseBody>  
  
    @GET("planetary/apod")  
    fun getApod(@Query("api_key") api_key: String, @Query("date") date:  
String ): Call<AstronomyPictureDayEntity>  
}
```





14. Using Retrofit

```
private fun getApodItem() {
    val service = ApodService.getApodItem()
    val serviceRequest = service.getApod(ApodService.API_KEY, formattedDate)
    serviceRequest.enqueue(object : retrofit2.Callback<AstronomyPicture-
DayEntity> {
        override fun onResponse(
            call: retrofit2.Call<AstronomyPictureDayEntity>,
            response: retrofit2.Response<AstronomyPictureDayEntity>
        ) {
            val apod = response.body()
            apod?.url?.let {
                Log.i(MainActivity::class.simpleName, "URL: " + apod.url)
                getApodImage(it)
            }
        }
        override fun onFailure(call: retrofit2.Call<AstronomyPicture-
DayEntity>, t: Throwable) {
            Log.i(MainActivity::class.simpleName, "on FAILURE!!!!")
        }
    })
}

private fun getApodImage(url: String) {
    val service = ApodService.getApodImage()
    val serviceRequest = service.downloadImageUrl(url)
    serviceRequest.enqueue(object : retrofit2.Callback<ResponseBody> {
        override fun onResponse(
            call: retrofit2.Call<ResponseBody>,
            response: retrofit2.Response<ResponseBody>
        ) {
            response.body()?.let { readStream(it.byteStream())
                current = LocalDateTime.now().minusDays(clickCounter++)
                formattedDate = current.format(formatter)
                button.setText("Get Image " + formattedDate)
            }
        }
        override fun onFailure(call: retrofit2.Call<ResponseBody>, t: Thro-
wable) {
            Log.i(MainActivity::class.simpleName, "on FAILURE!!!!")
        }
    })
}
```

Using the created service comes down to defining your own service instance and referring to the defined method in this case **getApodItem**

This is followed by a call to the **enqueue()** method, thus sending a network query. The result of the query is received in callbacks **onResponse()** or **onFailure()** methods, depending on the result. In the above code, there are two methods, one retrieves the photo metadata and the other retrieves the photo.

15. Let's add code to handle the button and to display the image (**onCreate()**)





```
button = findViewById(R.id.button)
button.setText("Get Image " + formattedDate)

imageView = findViewById(R.id.imageView)

button.setOnClickListener {
    if (isNetworkConnected()) {
        getApodItem()
    }
}
```

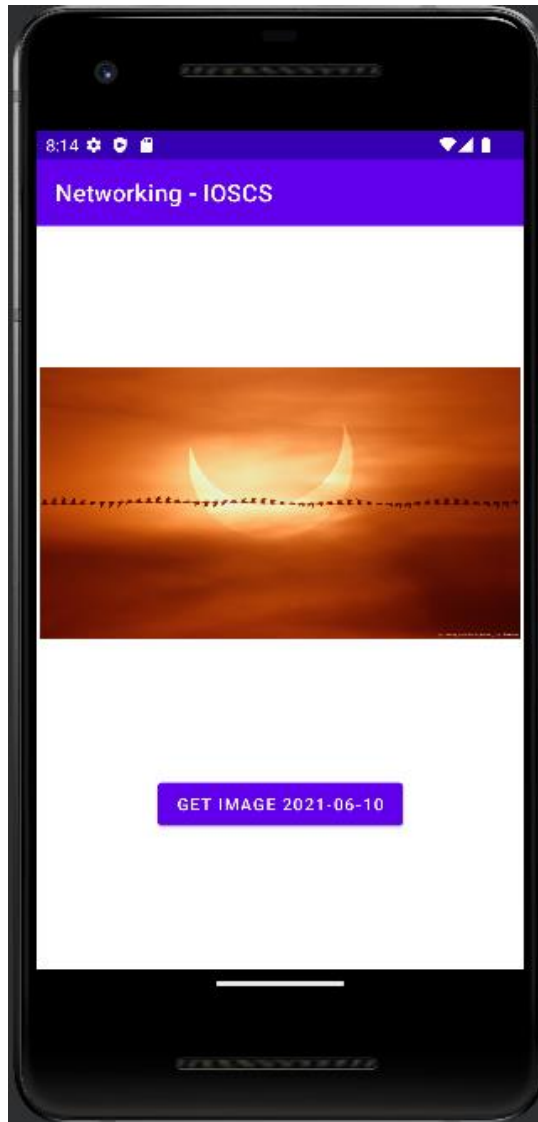
16. And a method that converts the resulting bitstream into an image

```
private fun readStream(inputStream: InputStream) {
    val bitmapImage = BitmapFactory.decodeStream(inputStream)

    CoroutineScope(Dispatchers.Main).launch() {
        imageView.setImageBitmap(bitmapImage)
    }
}
```

17. Run Apps

As a result, the application may look like the following:



Question

1. Is it necessary to ask the user for permission to use the internet?
2. Are the defined permissions of the type "dangerous"?

The resulting application code can be found at https://github.com/matam/Erasmus_Lab7.