Laboratory 1
Introduction

1 Useful Information

1.1 Attendance

At most **two** absences are allowed during the semester. The missed laboratory assignments will have to be completed at home.

1.2 Laboratory assignment presentation

- Mandatory, after submitting the assignment to moodle.
- The laboratory assignments will be submitted during the semester. At most **two** laboratory assignments can be submitted during the exam session and **two more** can be submitted during the reexamination period.

1.3 Grading

- The laboratory grade accounts for **30%** of the final grade.
- In order to be admitted to the exam, the laboratory grade must be at least **5**.
- The laboratory grade can be increased by **1 point** with a quiz that will be taken in the last week of the semester.
- In order to obtain **the maximum grade** on an assignment, it has to be submitted until the next laboratory. The highest score that can be

obtained is **100 points**. For each week of delay, the maximum score that can be obtained decreases by **25 points**. For a submission delayed by 4 weeks or more, the highest score that can be received is **10 points**.

- **Plagiarized submissions** will be graded with **0 points** and they will have to be resubmitted.

2 Development environment

We will use **AndroidStudio** to implement and test the applications which will be presented throughout the semester.

3 Android Application Elements

3.1 Components

Android applications have four types of components. Each component type provides an entry point which can be used by the users or by the system to access the application. The four types of components are the following:

- **Activities**
  - An **activity** provides a single screen which contains the user interface. Usually an activity fills the display and it generally implements one window of the application. An Android application often has multiple activities one of which is the **Main Activity**. The main activity is the one the user sees when the application is being launched.
  - An email application could have an activity to list all of the user's emails, another activity to read the emails and another one to compose new emails.
- **Services**
  - A **service** is a component which does not provide a user interface. It is used to keep an application running in the background as it performs long running tasks, while a different application could run in the foreground.

  - A service could play music in the background while the user is interacting with a different application.

- **Broadcast Receivers**
  - The **broadcast receiver** is a component that allows the application to receive events delivered by the system or by another application. It's not necessary for an application to run in order for the broadcast receiver to capture the transmitted event. Broadcast receivers don't have a user interface but they can create a notification shown in the status bar.
  - An application could schedule an alarm to remind the user of an upcoming event. When the alarm is about to go off, an event will be delivered to one of the application's broadcast receivers, so that the app does not have to remain running in the meantime.
- **Content Providers**
  - **Content providers** manage the shared set of application data. They provide methods which can be used to access data stored in the file system, in a SQLite database, on the web or on other persistent storages that the app can access. Through the content providers, other applications which have the necessary permissions, can query or access the data.
  - A content provider manages the device's contact information. Any application that has the proper permissions can query the content provider in order to read or to write information about the contacts.

3.2 The manifest file

The application's manifest is an XML file, **AndroidManifest.xml**, which can be found in the project's `app/manifest` subfolder. The manifest contains various information about the application, such as:

- The declaration of all of the application's components
- The permissions required by the application

- The minimum API level needed so that the application can run
- The declaration of the hardware and software features (camera, Bluetooth services, etc.) that the application uses or requires

3.3 Resources

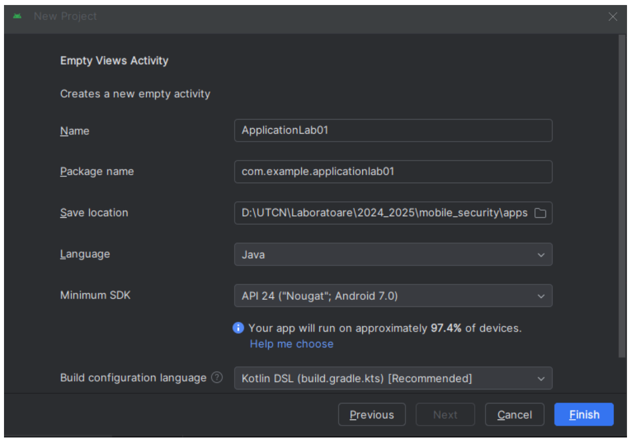
Android applications require that the resources are separated from the source code. This allows applications to provide alternative resources for different device configurations. The resources can be found in the project's `app/res` subfolder and they consist of images, menus, colors, values, layouts of the application's activities.

Each resource is identified by a unique integer ID, which can then be used to reference the resource from the application's code or from another resource.

4 Creating a new project in Android Studio

- From the *Welcome to Android Studio* window, click **New Project** or, from the *top left menu*, click **File** → **New** → **New Project**.
- From the *Templates* tab on the left, choose **Phone and Tablet** and on the right, choose **Empty Views Activity**.
- Fill in the information required in the *New Project* window:
  - **Name**: The name of the application that we are going to develop. This is the name that will show up in the phone's list of applications.
  - **Package name**: The complete name of the project. This has to be unique for all of the packages installed on one Android device.
  - **Save location**: The folder in which all of the project's files will be stored.
  - **Language**: The programming language which will be used to develop the application.
  - **Minimum SDK**: The minimum Android API level which is required on the device so that the application can run.

- **Build configuration language**: A tool called *Gradle* is used to build Android applications. The build configuration language specifies what programming language the build scripts will be written in. The build scripts that we will need throughout this laboratory will automatically be generated by Android Studio.



Empty Views Activity

Creates a new empty activity

Name: ApplicationLab01

Package name: com.example.applicationlab01

Save location: D:\UTCN\Laboratoare\2024_2025\mobile_security\apps

Language: Java

Minimum SDK: API 24 ("Nougat"; Android 7.0)

Your app will run on approximately **97.4%** of devices.
Help me choose

Build configuration language: Kotlin DSL (build.gradle.kts) [Recommended]

Figure 1: New project information

5 Running an Android application

In order to run an application inside an emulator, an **Android Virtual Device (AVD)** will have to be set up. The AVD is a set of configurations used to describe the type of Android device on which we want to run our application.

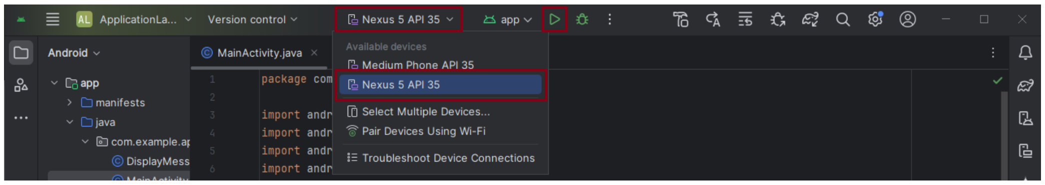
5.1 Creating an Android Virtual Device

- In the top left menu click **View** → **Tool Windows** → **Device Manager**
- In the tab that opened up, click on the **Plus sign** → **Add a new device** → **Create a Virtual Device**
- In the newly opened window, search for and choose a **Nexus 5** phone

- Choose the recommended API level
- Check the device's settings and click **Finish**

5.2 Running the application inside the AVD

- Select the previously created AVD and run the application



```
package com

import andr
import andr
import andr
import andr
```

Figure 2: Select the AVD and run the application

- The *Runnign Devices* tab will automatically open on the right side of the screen and the application will be run in the emulator. The emulator doesn't have to be closed for changes to become visible. It can be left opened and the application can be re-run by clicking the run button.

6 The application's user interface

An Android's application user interface is composed of a hierarchy of **View** and **ViewGroup** objects.

Views are the basic building blocks for user interface components. A View is a visible element, such as a button, that the user can see and interact with. The View is responsible for drawing the element and for handling the events related to it.

A **ViewGroup** is an invisible container that defines the layout structure for Views and other ViewGroup objects.

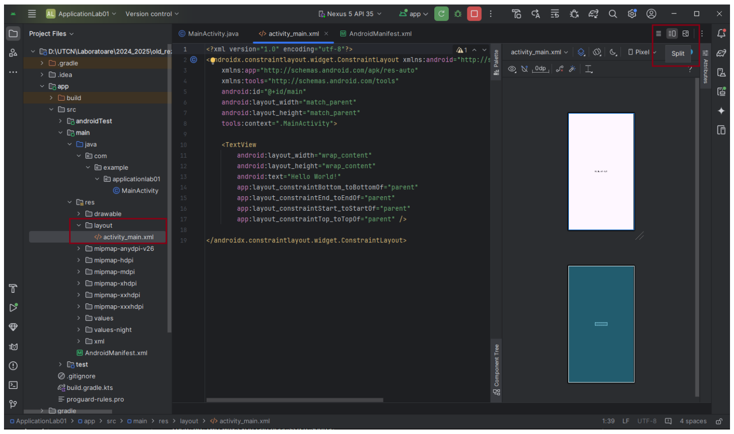
The **layout** defines the structure for a user interface inside the application, such as an activity. A layout consists of a hierarchy of View and ViewGroup objects. There are multiple types of layouts, among which: *AbsoluteLayout*, *FrameLayout*, *LinearLayout*, *RelativeLayout*, *ConstraintLayout* and *GridLayout*.

7 The test application

The test application that we have just created already contains a *ConstraintLayout* with a *TextView* subelement. The main activity's layout can be seen and edited inside the `res/layout/activity_main.xml` file.

The layout resource can be referenced in the project's code through the **R class**, using the resource's ID, which in this case will be *activity_main*: *R.layout.activity_main*.

In order to see the layout resource's XML code, as well as the way the layout's design should look when the application is run, click on the `activity_main.xml` file and choose the *Split view* from the top right corner of the screen as shown in Figure 3.



```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://s
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    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_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

Figure 3: Layout file split view

7.1 Creating a LinearLayout

- Right click on the `app/res/layout` subfolder → **New** → **Layout Resource File**
- Give the file a suggestive name (`linear_layout.xml`) and choose *LinearLayout* as the *Root element*
- Inside the `linear_layout.xml` file, change the **android:orientation** attribute to *horizontal*

- The **android:layout_width** and **android:layout_height** attributes are mandatory for each GUI element. Setting the value of these attributes to **match_parent** ensures that the element will have the same width and height as its parent. In our particular case, the *LinearLayout* element is the root element so setting the value of *android:layout_width* and *android:layout_height* to *match_parent* means that the element will have the same size as the device's screen

- In order for our application's main activity to use the layout that we have just defined, we have to pass the new layout's ID to the **setContentView** function call

  - In the `app/java/<package_name>/MainActivity.java` file, update the parameter with which the **setContentView** function is called from **R.layout.activity_main** to **R.layout.linear_layout**

7.2 Adding a text input field (*EditText*)

- In the *LinearLayout* that we have defined above, add and element of type **EditText** and set the **android:layout_width** and **android:layout_height** attributes to **wrap_content**. A value of *wrap_content* means that the element's width and height will be as big as necessary to fit the element's contents

- Set the value of the **android:id** attribute to **@+id/edit_message**. The *android:id* attribute specifies a unique identifier for the element which can later be used from the application's code to read and set the value of the element. The @ character is necessary when we are refering to a resource object from within a XML file. It is followed by the resource's type, in our case *id* then a slash character (/) and then the name of the resource (*edit_message*). The + symbol that precedes the resource's type is only needed when we are defining a resource's ID for the first time

- Set the **android:hint** attribute to **@string/edit_message**. The *android:hint* attribute contains the default string that will be displayed in the *EditText* element. Instead of hard-coding this value, we will define it in a string type resource. The creation of the string type resource will be described in the following subsection

7.3 Adding a string type resource

We will add two new string resources inside the `app/res/values/string.xml` file

```
<string name="edit_message">Enter a message</string>
<string name="button_send">Send</string>
```

Listing 1: String resources

7.4 Adding a button

In the `app/res/layout/linear_layout.xml` file, after the *EditText* element, we will also add a button:

```
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/button_send"
/>
```

Listing 2: Adding a button

7.5 Updating the width of the EditText element

As we have set the width of the *EditText* element to *wrap_content*, the text field will be as wide as it is needed for the placeholder text to fit in it. However, a longer input text could be inserted in the *EditText* element. We will change the width of the field so that it will occupy all of the free screen space. In order to achieve this, we will use the **android:layout_weight** attribute.

The weight of an element is a number which specifies how much of the free screen space should be occupied by the element. For example, if we have two elements, the first one with a weight of 2 and the second one with a weight of 1, then the first element will occupy 2/3 of the screen's width while the second one will take up only 1/3 of the screen's width.

The default weight of an element is 0. If only one element has a weight that's greater than 0, then that element will take up all of the free space that's left after all of the other elements have been placed on the screen.

In order to change the weight of the *EditText* element, the following attributes have to be updated:

- Add the **android:layout_weight="1"** attribute to the element
- Change the value of **android:layout_width** attribute to **0px**

8 Adding a new activity

Up until this point, the application that we have created only has one activity. It contains a text input field and a button. In this section, we will add code to the `MainActivity.java` file so that a new activity will be started every time the use presses the *Send* button.

8.1 Intents

An **Intent** is an object which describes an operation to be performed. Intents facilitate communication between components and they can be thought of as the application's intention to perform a given action. Intents have three main use cases:

- Starting an activity
- Starting a service
- Delivering a broadcast

There are two types of intents:

- **Explicit intents**
  - They are usually used to start a component in the same application as they require the full class name of the activity or service that needs to be started.
- **Implicit intents**
  - They do not require a specific component name. Instead, they declare an action that needs to be performed and this action can then be handled by a component from a different application.

  - For example, an implicit intent could be used to request from another qualified application, to open a web page or to show a location on the map.

8.2 Adding an *onClick* listener to the Send button

- In the `app/res/layout/linear_layout.xml` file, add the attribute **android:onClick=”sendMessage”** to the **Button** element. *sendMessage* is the name of the method that will be called every time the *Send* button is going to be pressed

- In the `app/java/<package_name>/MainActivity.java` file, add the **sendMessage** method. In order for the compiler to associate this method with the method specified in the XML file, the method has to be *public*, it has to return *void* and it has to receive a single parameter of type *View*. The *View* element which the function receives as a parameter is the element which was pressed, in our case the *Send* button.

```
public void sendMessage(View view) {
  // Read the content of the EditText field
  // Start a new activity
}
```

Listing 3: Send button onClick listener

8.3 Defining the *sendMessage* method

- Create an *Intent* object which will be used to start the *DisplayMessage* activity

```
Intent intent = new Intent(this, DisplayMessageActivity
  .class);
```

Listing 4: Defining an Intent

The *Intent* class constructor has two parameters:

  - A **context**, for which we will pass the value **this** as the *Activity* class is a subclass of *Context*

  - The component's class to which the system will send the *Intent*. In our case, this will be the *DisplayMessageActivity* class.

- Extract the text from the *EditText* element

```
EditText editTextComponent = findViewById(R.id.
  edit_message);
String message = editTextComponent.getText().toString()
  ;
```

Listing 5: Extracting the text from the input box

- Send the extracted text to the *DisplayMessageActivity* component through the *Intent*. In order to do this, the **putExtra** method of the *Intent* will be used. *Intents* can carry extra information as key-value pairs. The *putExtra* method expects the name of the key as the first parameter and the value as the second parameter.

```
intent.putExtra(EXTRA_MESSAGE, message);
```

Listing 6: Sending the message through the Intent

- Define the *EXTRA_MESSAGE* public constant so that the *DisplayMessageActivity* component will be able to access it and use it to interrogate the Intent.

```
public static final String EXTRA_MESSAGE = "lab1.
  my_first_app.MESSAGE";
```

Listing 7: Define the EXTRA_MESSAGE constant

- Call **startActivity** in order to start the activity which was specified when the *Intent* was defined.

```
startActivity(intent);
```

Listing 8: Start the new activity

8.4 Creating the ***DisplayMessageActivity***

- In the *Project Files* view on the left side of the screen, right click `app/java/<package_name>` → **New** → **Activity** → **Empty Views Activity**

- Fill in the details about the new activity and press **Finish**

  Any subclass of the *Activity* class has to implement the **onCreate** method. This is the method in which the newly created activity will receive the message sent by the activity which created it. This is also the place where the activity's layout has to be defined by calling the **setContentView** method. Android Studio automatically adds the *onCreate* method to newly created activities. Together with the newly created **DisplayMessageActivity.java** file, the activity's layout resource was also created and it can be found in the `res/layout/activity_display_message.xml` file.

- Inside the `res/layout/activity_display_message.xml` file, add a **TextView** component, which will be used to display the message received by the activity

```
<TextView
  android:id="@+id/show_message"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  app:layout_constraintBottom_toBottomOf="parent"
  app:layout_constraintLeft_toLeftOf="parent"
  app:layout_constraintRight_toRightOf="parent"
  app:layout_constraintTop_toTopOf="parent"
/>
```

Listing 9: TextView component

The *TextView* was relatively position using the **app:layout_constraintBottom_toBottomOf**, **app:layout_constraintLeft_toLeftOf**, **app:layout_constraintRight_toRightOf** and **app:layout_constraintTop_toTopOf** constraints. At least one horizontal and one vertical constraint should be specified. Setting the values of the constraints to **parent** references the parent container, which in this case will be the *ConstraintLayout*, and positions the *TextView* element in the center of the screen.

- **app:layout_constraintBottom_toBottomOf=”parent”**
  - The system will try to position the *TextView* element and the parent element so that their bottom edges share the same location.
- **app:layout_constraintLeft_toLeftOf=”parent”**
  - The system will try to position the *TextView* element and the parent element so that their left edges share the same location.

8.5 Receiving the Intent

Each activity is invoked by an *Intent*. In order to obtain a reference to the intent that requested the current activity, the **getIntent** method should be called. To obtain the content of the *EditText* field, which was sent by the *MainActivity* through the intent, the **getStringExtra** method should be called.

```
Intent intent = getIntent();
String message = intent.getStringExtra(MainActivity.
 EXTRA_MESSAGE);
```

Listing 10: Obtain the Intent and retrieve the text

8.6 Displaying the message inside the *TextView*

In order to display the received message inside the *TextView* component, we first need to get a reference to the component by calling the **findViewById** function. The content of the *TextView* element can then be set by calling the **setText** method.

```
TextView textView = findViewById(R.id.show_message);
textView.setText(message);
```

Listing 11: Obtain component reference and set the text

9 Assignment

Increase the font size of the text displayed inside the *TextView* component and change its color to green.

10 References

- `https://developer.android.com/guide/components/fundamentals`
- `https://developer.android.com/reference/android/view/View`
- `https://developer.android.com/reference/android/view/ViewGroup`
- `https://developer.android.com/develop/ui/views/layout/declaring-layout`
- `https://developer.android.com/guide/components/intents-filters`
- `https://developer.android.com/studio/run/managing-avds`
- `https://developer.android.com/guide/topics/resources/providing-resources`
- `https://developer.android.com/guide/components/activities/intro-activities`
- `https://developer.android.com/reference/android/content/Intent`
- `https://developer.android.com/reference/androidx/constraintlayout/widget/ConstraintLayout`