Mobile Computing

Practice # 1

First Android Application

1. Create an Android project in Android Studio (AS) for a first application

(when AS is opened, if you have a previous project displayed, close it using File->Close Project)

a) The form (Select a Project Template) for this **Create New Project** or **File->New->New Project** should appear.

Choose the Phone and Tablet **tab** at the top, and then 'No Activity' (this creates a project without components). Click on **Next** at the bottom.

b) Fill the 'Configure your project' form in this way:

Name is the name of your application. The project directory, and the name visible in Android Studio are composed from this name, removing the spaces. If it is composed by several words, capitalize them. For instance: My First App

Package Name is the unique identification of your application and usually contains also your organization and personal **id** inside it. It should be specified as a domain name in reverse: for instance, starting with ***org*** or ***com*** (non-profit or commercial organizations), organization name, developer or team name, and finally the app name. In my case, for instance, it could be: ***org.feup.apm.myfirstapp***. Choose one appropriate for you.

Each Android app should have a **unique** identity (when submitted to Google the app store), expressed as this Java package, grouping the classes you write.

Save Location is the directory where all the project files are stored. You should use a directory with the Application Name, inside some other directory with easy access from the PC file system.

Language specifies the programming language. It can be **Java** or **Kotlin**. For now, choose Java.

Minimum SDK is the minimum android API version where your application will run. That means it will not install on devices with an API level older than the one specified. Also, you **cannot** call in your code any method defined in newer versions of the API (unless you use an external static Google support library with it: now called the **JetPack** or **androidx** libraries). So, this setting should be a balance between the features we want and can use, and the base existing devices that we want to support. For now, specifying as minimum API, the level 21 (Lollipop 5.0) guaranties support to 94% of the world devices …

Leave the 'Use legacy android.support libraries' (the old Google support libraries) unchecked and click the **Finnish** button.

b) After Android Studio creates your project, wait for the first build to finish. Then examine the files in the project directories (app folder in the Android logical structure), specially the AndroidManifest.xml, the Java source, and the resource files (inside the subdirs of the **res** directory).

You can see in Android Studio, under the app/java project directory, three namespaces: one is the chosen project namespace, and the other two are for testing (using TDD) with two different frameworks (android test (espresso) and unit tests). For now, you can delete those two projects (delete first the files inside and then the namespaces). Also you can delete the corresponding subdirs in the file system.

Also, going to the File → Project Structure → Dependencies → app module (or the corresponding button on Android Studio toolbar) dialog box, you should delete the espresso-core, and the 2 junit external dependencies (libraries). This will lower the app size.

c) Now we will **add** the code for **the first activity** in our application. From the Android Studio menu select File → New → Activity → Gallery … . A 'New Android Activity' wizard appears. Select 'Empty Activity' (the simplest

one) and click the **Next** button at the bottom.

The activity that is added is empty. But It has already an icon, a style and a layout. For our app we will need to delete or modify the generated layout (it is generated with only a ConstraintLayout on it).

In the form that is now presented, we need to specify the activity name (the class name, which is also added to the manifest), generate a layout file (maintain this selected) and its name, and if it is our launcher activity (the one that is executed when the user starts the app). **Select this option** and maintain the next ones. For the class name choose **FirstActivity**.

We will use the 'Up Navigation' functionality that was automatically available after API level 16. As we are supporting Android versions with API 21 or higher, we don´t need to use a support library for that.

d) Try to run the application in several emulators and real devices (connected through USB to the development computer). You should see a screen with only an Action Bar (title) and a blank screen. The style of the app was already configured by Android Studio following the recommended design guidelines and using a derived Activity class (***AppCompatActivity***) from a support library, guaranteeing uniformity across API levels.

2. Modify the interface and behavior

Modify the generated project in the following way:

i. Open the res/layout/activity_first.xml file in design mode. You can see that it only contains a ***ConstraintLayout***. These layouts allow complex positioning of their children through conditions.

ii. In the Component Tree select the ConstraintLayout and in the context menu (mouse right button) use Convert View … to convert it to the standard and simple LinearLayout. Adjust the properties to obtain the following (see the file in Text mode).

```
<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="horizontal"
   tools:context=".FirstActivity">
</LinearLayout>
```

iii. You can now delete (because it takes space in the APK), in the Project Structure Dependencies tab, the constraint layout support library, as we do not need it anymore.

Try now to use the visual editor (design tab) in the res/layout/activity_first.xml file **to**:

a. Add a text field (PlainText, implemented as EditText view with the id edit_message) empty but with a hint ("Enter a message") inside the Linear Layout. All the interface strings should be first defined in the strings.xml resource file (res\values), and referenced with the syntax @string/<string name>.
b. Add a button, also inside the Linear Layout but following the text field, with a label "Send" (put it also in the strings resource) and an id of button_send. Also remove the layout-weight attribute value.
c. Try that the combination of text field and button occupy the screen width, like:

For that, the property layout_width of the EditText can be defined as zero: 0dp, but with the property layout_weight as 1.
Android tries to allocate space to interface elements proportional to their weights, but using the minimum to be visible. When a weight is not specified it is assumed to be 0.

d. The strings and layout XML files should now contain the following (or similar):

Strings:

```
<resources>
    <string name="app_name">My First App</string>
    <string name="edit_hint">Enter a message</string>
    <string name="button_send">Send</string>
</resources>
```

Layout:

```
<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="horizontal"
    tools:context=".FirstActivity">

  <EditText
      android:id="@+id/edit_message"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:ems="10"
      android:hint="@string/edit_hint"
      android:inputType="text"/>

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

e. Run the application and confirm the layout, verifying the EditText behavior and functionality.

3. Create a second activity

a) Using the Android Studio File/New menu entry, create another activity. Follow File → New → Activity → Empty Activity.

b) In the Configure Activity form fill it this way:

Activity Name: SecondActivity
Deselect all the checkboxes in the form (don't generate a layout and this is not a launcher activity). Click ***Finnish***.

c) Verify that the activity is already in the AndroidManifest and add the following property to this activity element: android:parentActivityName=".FirstActivity". This will create a way to navigate from the Second to the First activity.

4. Send and receive a message while navigating to the Second activity

This second activity should display the message sent with the intent that activates the activity (this intent will be built and sent by the FirstActivity activity).

So we need to get the activation intent, extract the message, and show it in this activity screen. We could

specify a layout in an XML layout file, but instead, and because this one is very simple, we'll do it in code. It is always possible to build layouts in code, but it is not very practical nor recommended, because it can promote some confusion between interface and business logic.

Edit the onCreate() method as follows:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);

   // Get the message from the intent
   Intent intent = getIntent();
   String message = intent.getStringExtra(FirstActivity.EXTRA_MESSAGE);

   // Create the text view
   TextView textView = new TextView(this);
   textView.setTextSize(30f);
   textView.setText(message);

   // Set the text view as the activity layout
   setContentView(textView);
}
```

The EXTRA_MESSAGE constant string will be defined in the first activity.

5. Respond to the Send button (FirstActivity) and activate the SecondActivity activity.

a) Define a new method in the FirstActivity activity as:

```
/** Called when the user clicks the Send button */
public void btSendOnClick(View view) {
   // Do something in response to button
}
```

(use Alt+Enter to update the import statements in the .java files)

b) Add to the onCreate() method the connection between the Send button and the previous method:

```
// find the button in the layout and set the listener for the click event
findViewById(R.id.button_send).setOnClickListener(this::btSendOnClick);
```

c) In the listener build an explicit intent pointing to the second activity, extract the text from the EditText box and attach it to the intent. After that, we can start the second activity.

```
/** Called when the user clicks the Send button */
public void btSendOnClick (View view) {
   Intent intent = new Intent(this, SecondActivity.class);
   EditText editText = findViewById(R.id.edit_message);
   String message = editText.getText().toString();
   intent.putExtra(EXTRA_MESSAGE, message);
   startActivity(intent);
}
```

d) Extra information attached to intents have a name (pair name/value). One last thing is to define that name. Let's do it as a constant static string in the first activity. Add the field EXTRA_MESSAGE as:

```
public class FirstActivity extends AppCompatActivity {
```

```
    public final static String EXTRA_MESSAGE = "<package>.MESSAGE";     //To be unique. Replace <package>
    ...
}
```

6. Run and use your first two activities application

Try some emulators and real devices.