Mobile Computing

Practice # 1

First Android Application

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

a) A first form for this **New Project** should appear. Fill it in this way:

Application 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

Company Domain is the identification of your organization or enterprise and usually contains also your personal **id** in the organization. It should be specified as a domain name: for instance, for me: apm.feup.org.

Each Android app should have a **unique** identity (when submitted to the app store), expressed as the Java package grouping the classes you write. Android Studio forms automatically this unique package name from the Company Domain (reversed) plus the Application Name.

Project Location is the directory where all the project files are stored. It should be the Application Name inside some other directory.

In the second form you should choose the type(s) of the target device. Choose **Phone and Tablet**. Also specify the minimum level of the Android version supported. A good choice is **API 16** (the first Jelly Bean), because it already includes most of the modern functionality expected on Android apps, and a big slice of all devices run already on that version or posterior.

The third form allows us to add the first activity to the project. This first activity is also the launching activity, and Android Studio marks it as so in the application manifest. Usually we start to choose an **Empty Activity** (the simplest one).

In spite of the name the activity that is added is not empty. It has already an icon, a style and a layout (with a TextView). For our app we need to delete or modify those resources.

Finally in the fourth form we can specify the activity name (the class name, which is also added to the manifest), generate a layout file (maintain this selected) and if it should derive from AppCompatActivity (deselect this option). Choose **FirstActivity** for the name.

b) After Android Studio creates your project, examine the files in the project directories, specially the AndroidManifest.xml, the Java source file, and the resource files (inside the subdirs of the res directory).

We will use the 'Up Navigation' functionality that was automatically available on API 16. As we are supporting Android versions with API 16 or higher, we don´t need to use the support libraries.

c) Try to run the application in several emulators and real devices (connected through USB to the development computer). You should see a screen with an Action Bar (title and icon) and a greeting message, doing nothing. The style of the app was already configured by Android Studio as one available in the minimum API supported (in this case API 16).

2. Modify the interface and behavior.

Modify the generated project in the following way:

i. Delete now everything from the res/layout/activity_first.xml file, but not the first line or the file itself.
Add now the following Linear Layout (Android Studio does not allow to drag and drop visually the root element of a layout):

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
</LinearLayout>
```

ii. Also delete the res/values/dimens.xml files (both), as our layout does not use them and also the res/values/colors.xml file.

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

a. Add a text field (EditText 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).
c. Try that the combination of text field and button occupy the screen width:

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:

Strings:

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

Layout:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal">
  <EditText android:id="@+id/edit_message"
    android:layout_weight="1"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />

  <Button
    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 using the Android Studio File/New menu entry.

a) Select an EmptyActivity and fill the form in this way:

Activity Name: SecondActivity
Deselect all of the checkboxes in the form.

b) Verify that the activity is already in the AndroidManifest and add the following property to this activity element: android:parentActivityName=".FirstActivity".

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 a 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(40f);
   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 and activate the SecondActivity activity.

a) Add an onClick property to the Button on the first activity layout. Call it "sendMessage". This will be the handler of the onClick event.

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

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

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

c) 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 sendMessage(View view) {
   Intent intent = new Intent(this, SecondActivity.class);
   EditText 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 First extends Activity {
   public final static String EXTRA_MESSAGE = "<package>.MESSAGE";    //To be unique. Replace <package>
   ...
}
```

6. Run and test your first two activities application.