Lab 2 - Activities, Intents, SharedPreferences

In this assignment you will create your own app from the scratch. You will learn how to start an Activity from another Activity via Intents, and how to pass data between activities (via Intents and via SharedPreferences). You will also learn how to start a camera photo capture, and will learn about different layouts and UI widgets.

Your app is going to consist of the following activities:

- **RegistrationActivity** - This activity lets the user enter his/her name, which will be saved to SharedPreferences. From this activity the user is forwarded to:
- **ProfileActivity** - This activity shows the name entered earlier and lets the usrer take a picture. The picture will be shown on the same page. The user is also given an option to enter a message.
- **DisplayMessageActivity** - Displays the message entered above in a single screen.

Let's start programming the above app.

First, create an empty project, without any classes. Create a `RegistrationActivity.java` in the main **src** folder, make it extend `AppCompatActivity` class. **RegistrationActivity** needs a layout. Create a new folder called layout in the res folder and add profile_layout.xml to the folder. Go back to `RegistrationActivity.java` and override `onCreate` method. Add `setContentView(R.layout.`*`registration_layout`*`)`; This tells the OS, that this Activity will show profile_layout screen. The R file links to all the resources. Open it to see for yourself! Ctrl+B to see declaration, Ctrl+Enter+B to see more.

Now go to `profile_layout.xml`. Add a Large text widget to your layout by dragging it to the screen. Design and Text tabs allow you to switch between the xml and its rendering. Add one more `TextView` below it. This one will hold our consent terms. Now add an `EditText` element below. This is where the user will sign their name. Then, add a button at the bottom. Finally, open the `TextView` of the layout and add a horizontal line between the `TextView` with our consent terms, and the EditText where a user's signature will be typed. You can add such a line by putting the following XML snippet in between the two elements:

```
<View
  android:layout_width="fill_parent"
  android:layout_height="1dip"
  android:background="#000000" />
```

Now we want to add the right text to the fields. We are not going to hardcode values within the XML, but to refer the XML to another file strings.xml where the values will be stored. For this we are going to create entries in the values/strings.xml file:

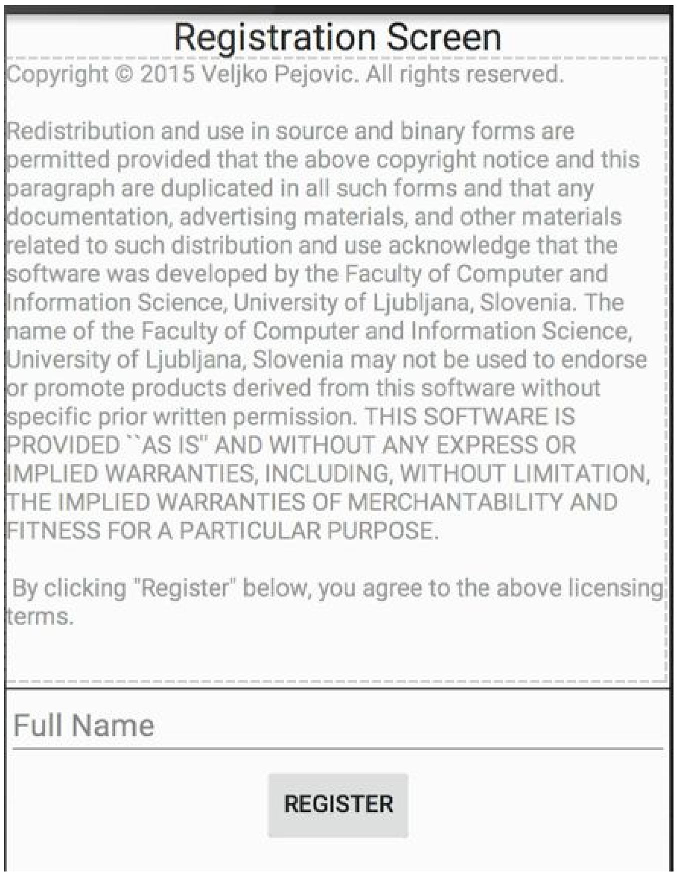
- `reg_title_text` with value "Registration Screen"
- `reg_full_name_text` with value "Full Name" and
- `reg_consent_text` with value: "`Copyright (c) 2015 <copyright holder>.`
  `All rights reserved.`

  `Redistribution and use in source and binary forms are permitted`
  `provided that the above copyright notice and this paragraph are`

```
duplicated in all such forms and that any documentation,
advertising materials, and other materials related to such
distribution and use acknowledge that the software was developed
by the Faculty of Computer and Information Science, University of
Ljubljana, Slovenia. The name of the the Faculty of Computer and
Information Science, University of Ljubljana, Slovenia may not be used
to endorse or promote products derived
from this software without specific prior written permission.
THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
By clicking "Register" below, you agree to the above licensing terms.”
and reg_button_text with value “Register””
```

You should now refer to these values in the layout file. Open the Text tab, and replace android:text with appropriate references, e.g. `“@string/reg_consent_text”`. In the `EditText` element, add `android:hint=”@strong/reg_full_name_text”`.

You’ll get something like this:



Registration Screen

Copyright © 2015 Veljko Pejovic. All rights reserved.

Redistribution and use in source and binary forms are permitted provided that the above copyright notice and this paragraph are duplicated in all such forms and that any documentation, advertising materials, and other materials related to such distribution and use acknowledge that the software was developed by the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. The name of the Faculty of Computer and Information Science, University of Ljubljana, Slovenia may not be used to endorse or promote products derived from this software without specific prior written permission. THIS SOFTWARE IS PROVIDED ``AS IS'' AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

By clicking "Register" below, you agree to the above licensing terms.

Full Name

REGISTER

To link our button with an action, we have to capture `onClick` event. There are multiple ways of accomplishing this. For example, we can add the following property to the `Button` element in the XML layout file: `android:onClick="registerUser"`. This means that registerUser function will be called. We now go back to the **RegistrationActivity** to implement the function. The function has to take a View object as an argument, i.e. public void registerUser(View view). Here we want to check if a user filled out the username and if not, show a warning. First, we need to access the profile_layout’s `EditText` element. To do so, we can use `findViewById(int)` function of the Activity class. We can drill deeper into the returned View to get the value of the entered text and check if it’s length is larger than zero. If not, we can show a warning via the View's `setError(Strong)` method. Remember that you should not hardcode any text that is going to be shown to the user.

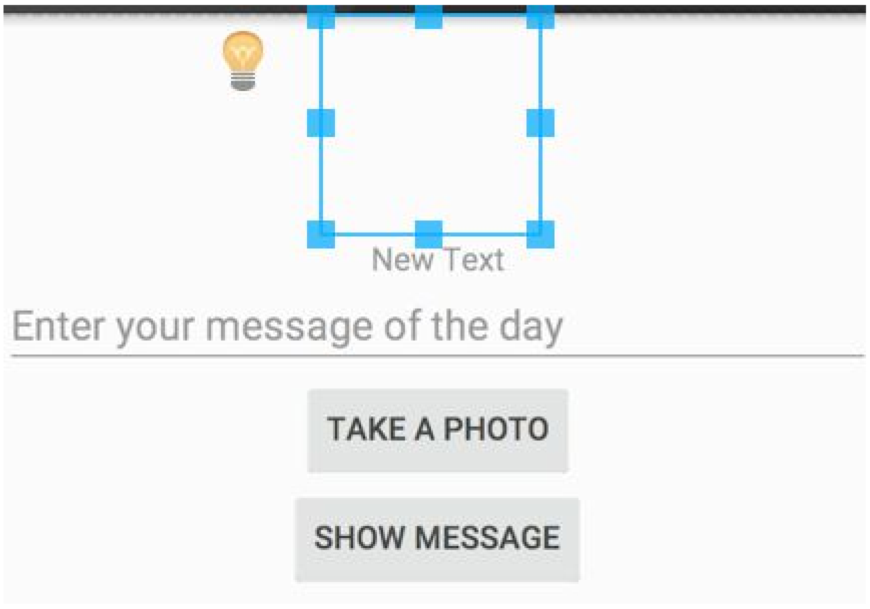
Once the user fills out the requested info, and clicks the button, the entered information should be saved and the user should be forwarded to another activity - **ProfileActivity**. First, to save the information we use `SharedPreferences`. `SharedPreferences` are one of the ways of achieving data persistence in Android. They represent a set of key-value pairs, that is kept saved as long as the application is installed on a phone. The following code saves a String value under the key "full_name":

```
     SharedPreferences settings = getApplicationContext()
             .getSharedPreferences("preferences", MODE_PRIVATE);
      SharedPreferences.Editor editor = settings.edit();
      editor.putString("full_name", value);
      editor.commit();
```

To start an Activity from another Activity we use Intents. An intent takes the name of the class it needs to launch. Like this:

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

To complete **ProfileActivity**, we need to draw its interface. We create profile_layout in the layout directory of the resources. Now, put an ImageView object on it, as well as a `TextView` object below. Underneath that add an `EditText` object. Finally add two buttons. We will use one of the buttons to start a camera, and we will show the image on the ImageView object. The TextView object will show a user's name, while the `EditText` button will let the user input their "message of the day". Clicking on the second button will start a new activity that will show the message up close.
You should mimic the process from the first activity, and your end result should look something like this:



In the first activity we specified the function to be called when a button is clicked on directly in the layout (via GUI, but also possible via XML). An alternative is to define what happens when you click on a button in the Activity itself, more specifically in its `onCreate` method.
In `onCreate` method of **ProfileActivity** add:

```
final Button cameraButton = (Button) findViewById(R.id.prof_button_camera);
     cameraButton.setOnClickListener(new View.OnClickListener() {
                 public void onClick(View v) {
                     // Perform action on click
                 }
             });
```

To launch an activity which will show us the camera output, we use:

```
Intent takePictureIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
if (takePictureIntent.resolveActivity(getPackageManager()) != null) {
   startActivityForResult(takePictureIntent, REQUEST_IMAGE_CAPTURE);
}
```

Where `REQUEST_IMAGE_CAPTURE` is our constant code for this specific intent. We should define it in the **ProfileActivity** class.

Spend some time over this code until you understand it, then add it in the `onClick` method.
This calls an Activity, which will take care of photographing, and requires a result back. We get the result back by overriding the onActivityResult method. We should now check `requestCode` equals our sent code and if the resultCode equals the built-in code for a successful return from an activity: `Activity.`*`RESULT_OK.`* Then, we can get the binary data from a Bundle that was returned by the camera activity, and set the value into our image holder (`ImageView`).

```
Bundle extras = data.getExtras();
Bitmap imageBitmap = (Bitmap) extras.get("data");
ImageView profileView = (ImageView) findViewById(R.id.prof_image);
profileView.setImageBitmap(imageBitmap);
```

Another functionality of **ProfileActivity** is to show the message of the day. Similarly as with the camera button, we will add an onClickListener for the message button in the onCreate method of **ProfileActivity**. In the `onClick` method we will get the text of the message, and then start a new activity called **DisplayMessageActivity** using an Intent. We will bundle a string message to the intent, like this:

```
Intent intent = new Intent(ProfileActivity.this,
DisplayMessageActivity.class);
EditText editText = (EditText) findViewById(R.id.prof_msg_text);
String message = editText.getText().toString();
intent.putExtra(EXTRA_MESSAGE, message);
startActivity(intent);
```

The `putExtra` method takes a name of the extra resource (created a String constant EXTRA_MESSAGE), and the value that needs to be sent to another activity.

Finally, before moving to **DisplayMessageActivity**, ensure that you have populated the `TextView` with the user's full name. We saved the full name already in SharedPreferences. Now we will retrieve it with:

```
SharedPreferences settings = getApplicationContext()
```

```
        .getSharedPreferences("preferences", MODE_PRIVATE);
String fullName = settings.getString("full_name","Default Name");
```

DisplayMessageActivity is going to be fairly simple. We won't even create a layout for it. Rather, we will get the data from the intent that started it, and show everything in a `TextView`. The `onCreate` method should look like this:

```
super.onCreate(savedInstanceState, persistentState);
// Get the message from the intent
Intent intent = getIntent();
String message = intent.getStringExtra(ProfileActivity.EXTRA_MESSAGE);

// Create the text view
TextView textView = new TextView(this);
textView.setTextSize(40);
textView.setText(message);

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

That's it! We're done with programming. We just have to ensure that the app knows what all of it's parts are. We dig into Manifest now. Each activity should be listed there. The main one is **RegistrationActivity**, it's the first that gets launched:

```
<activity
   android:name=".RegistrationActivity"
   android:label="@string/title_registration_activity" >
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

Other Activities will be done analogously.

Our app uses the camera. We will note that in the manifest, so that Play store knows that we're using it, and can suggest camera-less users not to install the app:

```
<uses-feature android:name="android.hardware.camera"
             android:required="true" />
```

Finally, we want a non-default icon for our app. First, have an image that you'd like to use ready. Now, click on drawable folder in the resources with the right mouse button and select `new->image asset`. Follow the instructions and Android Studio will format the icon for you. Easy!

Now, if you start the application, you will notice that on smaller screens you cannot get pass the registration screen. The problem is that you can't scroll down to click "Register". To fix this, we will add `ScrollView` around the `LinearLayout` in the `registration_layout.xml`.

```
<ScrollView
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="si.uni_lj.lrss.lab2.RegistrationActivity"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="16dp"
    android:id="@+id/scrollView">
```

That’s it! Test your app. If you see any problems use the debugging tools. Android Device Monitor contains a set of tools including Logcat, Traceview and UI hierarchy viewer. You can start the monitor by clicking on the icon of a lone green robot in the toolbar.