**Lab 4: Fragments and Menus**

In case you found something to improve, please tell us!
https://forms.gle/Nf27cXFf7AwaBL55A

This class teaches you how to use fragments to handle portions of the main activity in our sport tracker. Additionally, you will learn how to use action bar menu item.

# 1 Android Studio Tricks

Here are very useful **Android Studio** tricks you should always use (check *Section 5* of **Lab1b** for more detailed explanation on how to use **Android Studio** tools):

1. Use **Alt+Enter** (**Option+Enter** for Mac users) when you have an error in your code: put the cursor on the error and click **Alt+Enter**. You can also use it to update the **`gradle`** dependencies to the latest version.
2. Use **Ctrl+Space** to check the documentation of a **`View`**, method or attribute: put the cursor on the object and do **Ctrl+Space**. You can also use it to complete the typing of these objects. Otherwise **Android Studio** always gives a list of suggestions where you can choose the object you need.
3. **ALWAYS CHECK THE COMPILATION ERRORS!** They are usually quite self-explanatory.
4. **ALWAYS DEBUG AND CHECK THE ERRORS IN LOGCAT!** Read the usually self-explanatory errors and click on the underlined blue line to go in the position of the code where the error is.

For more **useful keyboard shortcuts**, please check this LINK[1]

# 2 Introduction to Fragments

**`Fragment`**s are behaviours or portions of user interface in an **`Activity`**. Multiple fragments can be combined in a single activity to build a multi-pane UI and they can be reused in multiple activities. A **`Fragment`** has its own lifecycle, but it is directly affected by the lifecycle of the **`Activity`**, as shown in Figure 1. For example, when the **`Activity`** is paused, so are the **`Fragment`**s in it, and when the **`Activity`** is destroyed, so are the **`Fragment`**s.

A **`Fragment`** has its own layout and it lives in a **`ViewGroup`** inside the **`Activity`**'s view hierarchy. There are two ways to add a fragment to a layout:

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

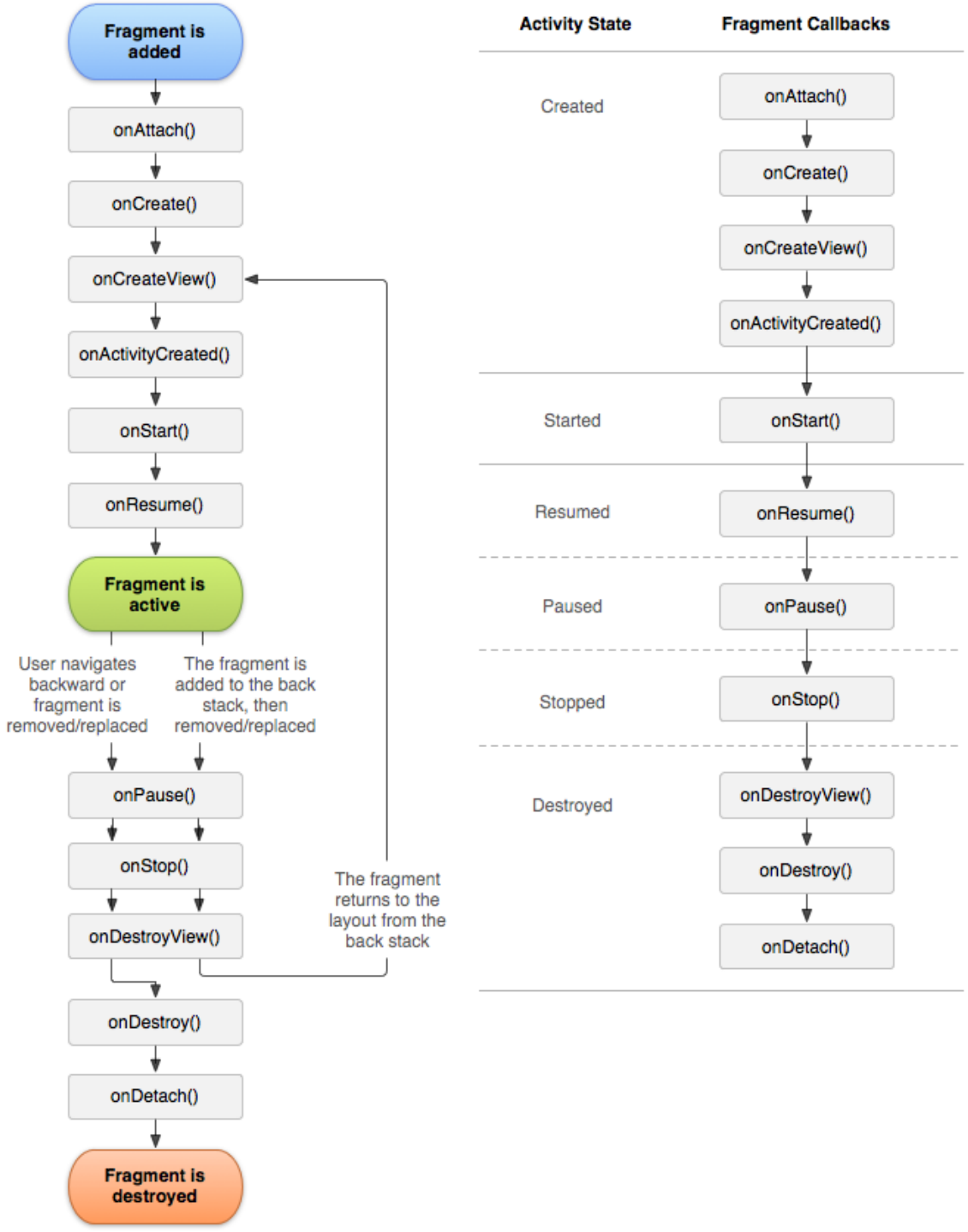


Figure 1: **`Fragment`** and **`Activity`** lifecycle

1. Declaring it inside the activity's layout file, as a fragment element, specifying the properties as if it was a view. The `android:name` specifies the **`Fragment`** class to instantiate.
2. Programmatically, adding it through the **`FragmentManager`**, which manages fragments, such as adding or removing them from the activity.

However, a fragment can also be an invisible worker for the activity, that is, without any layout.

# 3 Lab 4 Structure

We want to continue our sport tracker app by adding **`Fragment`**s as three tabs to show multiple pages, as shown in Figure 2:

1. the user profile page.
2. a page for recording a new activity,
3. a page with the list of past recordings.

For this, we will use the **`FragmentManager`** by implementing a **`ViewPager`** to have a fragment for each page to swipe and a **`FragmentStatePagerAdapter`** which handles the fragments on a **`ViewPager`**.

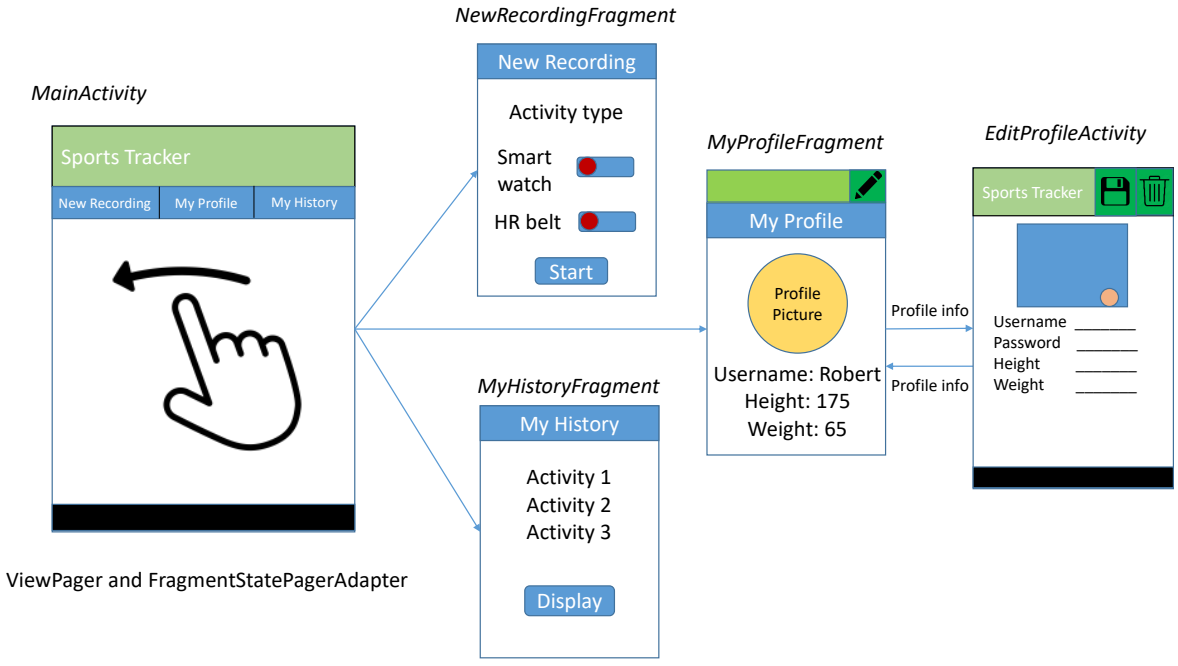


Figure 2: The structure of the fragment in our sport tracker app

# 4 Fragments and ViewPager

## 4.1 Adding Fragments

First thing to do is to create three **Fragment** classes in the same package of **MainActivity** (right-click on the package name within the mobile module and **New → Fragment (Blank)**). We can call the **Fragment**s and their corresponding layouts as follows:

- **MyProfileFragment** and **fragment_my_profile.xml**
- **NewRecordingFragment** and **fragment_new_recording.xml**
- **MyHistoryFragment** and **fragment_my_history.xml**

The generated fragments have some already implemented methods, although for now we can only focus on the **onCreateView(...)**, which is the method called to inflate the **Fragment**'s' layout. Finally, the **Activity** that contains the fragments should implement the interface called **OnFragmentInteractionListener** for the communication between the **Activity** and its **Fragment**s. In order to do this, add to the class definition (after **extends AppCompatActivity**) the java command **implements** followed by the interface for each **Fragment** (for instance, **NewRecordingFragment.OnFragmentInteractionListener**). Use *Android Studio*'s code inspector to automatically import each interface and implement the required method (defined by the *interface*). In our case, the code inspector will ask us to create the **onFragmentInteraction(...)** method, which we will leave empty.

## 4.2 SectionStatePagerAdapter class

The next thing to add to the project is a class that extends a **FragmentStatePagerAdapter**, which is an implementation of a **PagerAdapter** that uses a **Fragment** to manage each page. This class is good to manage an arbitrary number of fragments, and handles them like a **ListView** handles its child items. The class requires to implements two methods: **getItem(...)** and **getCount()**. It also requires to implement the constructor that takes as input a **FragmentManager**. To handle the fragments we need a **List<Fragment>** (i.e. a **List** that contains elements of type **Fragment**), which we will fill with the three fragments created.

First, create a class called **SectionStatePagerAdapter** in the same package with a superclass **FragmentStatePagerAdapter** (right-click on the package name and **New → Java Class**). Once created you will have some errors, which you can solve by following the suggestions of *Android Studio*. This is for overriding the already mentioned methods whose content we need to change. Before this, we need to insert two fields in the class for the list of fragments and a list of fragment titles to be able to identify the fragments.

```
private final List<Fragment> mFragmentList = new ArrayList<>();
private final List<String> mFragmentTitleList = new ArrayList<>();
```

Then, we insert a method to add elements in the list of fragments.

```
public void addFragment(Fragment fragment, String title) {
    mFragmentList.add(fragment);
    mFragmentTitleList.add(title);
}
```

Then, we can add a method for getting the position of a specific fragment in the list created by giving as input the title we assigned to it.

```
public int getPositionByTitle(String title) {
    return mFragmentTitleList.indexOf(title);
}
```

Now that we have the fields and the method to add fragments, we can override the two methods **`getItem(...)`**, which returns a **`Fragment`** given the index in the list, and **`getCount()`**, which returns the size of the list of fragments.

```
@Override
public Fragment getItem(int i) {
    return mFragmentList.get(i);
}

@Override
public int getCount() {
    return mFragmentList.size();
}
```

Finally, we override the last method called **`getPageTitle(...)`** to get the title of the fragments, which will be set in the tabs strip that we will put under the action bar to swipe through fragments.

```
@Nullable
@Override
public CharSequence getPageTitle(int position) {
    return mFragmentTitleList.get(position);
}
```

## 4.3 Moving MainActivity features to MyProfileFragment

The **MainActivity** will be the activity which contains our three fragments. For this reason, we move the functionality and the layout we added in *Lab 3* to the **MyProfileFragment**.

First, we need to copy-paste the layout of the main activity (including the outer **ConstraintLayout** container) into **fragment_my_profile.xml**, in the **FrameLayout** that is already there (you can remove the default **TextView** put by *Android Studio*).

Then, we need to copy the intent functionality we added in *Lab 3* to **MainActivity onCreate** method and paste it in the method **onCreateView(...)** of the **MyProfileFragment**, specifically receiving the user profile from the **LoginActivity**. The method will look like the code below.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    fragmentView =  inflater.inflate(R.layout.fragment_my_profile,
            container, false);

    Intent intent = getActivity().getIntent();
    userProfile = (Profile) intent.getSerializableExtra("userProfileWelcome");
    setUserImageAndWelcomeMessage();
    sendProfileToWatch();

    return fragmentView;
}
```

Notice that we are saving the view of the fragment in **fragmentView** to be able to call the components of the fragments layout (such as **findViewById(...)**). We do this because the layout is not linked anymore with the **Activity** but with each **View** of the fragment. Moreover, the fragment cannot implement some activity-related method such as **getIntent()** or **startService(...)**, therefore we need to use **getActivity()** to call activity-related methods through the activity linked to the fragment.. Try to change the methods **setUserImageAndWelcomeMessage()** and **sendProfileToWatch()** accordingly.

## 4.4 MainActivity as a ViewPager

Now, we need to set up first the **MainActivity** layout then its functionality. We have an empty layout for the **MainActivity**, which we will fill in. The activity that contains the

fragments handled by a **PagerAdapter** needs to include a **ViewPager** layout, as shown in the code snippet below.

```
<androidx.viewpager.widget.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mainViewPager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <androidx.viewpager.widget.PagerTabStrip
        android:id="@+id/pagerTabStrip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#20B2AA"
        android:textColor = "#fff"
        android:paddingTop="15dp"
        android:paddingBottom="15dp" />

</androidx.viewpager.widget.ViewPager>
```

We also add an element called **PagerTabStrip** which adds the title tabs under the action bar and we will be able to swipe through these tabs.

## 4.5 Setup of ViewPager and Fragments in MainActivity

The last step to have a working app with swiping tabs after login is to add the fragments as **private** objects in the **MainActivity** class definition. Then we add this code in the **onCreate(...)** method to set up our **ViewPager** and initialize the fragments.

```
mSectionStatePagerAdapter = new SectionStatePagerAdapter
        (getSupportFragmentManager());

myProfileFragment = new MyProfileFragment();
newRecFragment = new NewRecordingFragment();
myHistoryFragment = new MyHistoryFragment();

ViewPager mViewPager = findViewById(R.id.mainViewPager);
setUpViewPager(mViewPager);
```

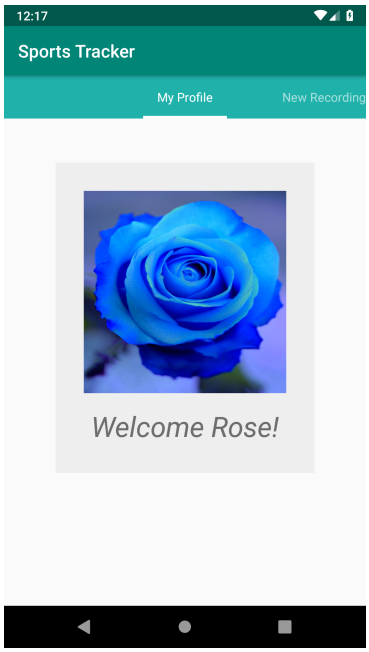


Figure 3: Fragments as swiping tabs

```
// Set NewRecordingFragment as default tab once started the activity
mViewPager.setCurrentItem(mSectionStatePagerAdapter
        .getPositionByTitle("New Recording"));
```

The missing method **setUpViewPager(...)** is simply filling in the **FragmentStatePagerAdapter** created previously:

```
private void setUpViewPager(ViewPager mViewPager) {
    mSectionStatePagerAdapter.addFragment(newRecFragment, "New Recording");
    mSectionStatePagerAdapter.addFragment(myProfileFragment, "My Profile");
    mSectionStatePagerAdapter.addFragment(myHistoryFragment,"My History");
    mViewPager.setAdapter(mSectionStatePagerAdapter);
}
```

In the code above, we first need to create an instance of the **SectionStatePagerAdapter**, which takes in input the **FragmentManager** to be able to add the three new fragments as tabs in the **ViewPager**. Then, we save the **ViewPager** in a field in the **MainActivity**. This class has a method **setAdapter(...)** that takes as input a **PagerAdapter**. Then, we implement a private method called **setUpViewPager(...)**, which will add the fragments in the list of fragments of the adapter, so that they will be our tabs in the welcome page after login.

Finally, we need to override the method **onFragmentInteraction()** in **MainActivity()**

(for now we can leave it empty), and we need to implement the **OnFragmentInteracti onListener** of the three fragment classes. The **MainActivity** code should resemble the following:

```
public class MainActivity extends AppCompatActivity implements NewRecordingFragment
    .OnFragmentInteractionListener, MyProfileFragment.OnFragmentInteractionListener,
    MyHistoryFragment.OnFragmentInteractionListener {
    ...
    @Override
    public void onFragmentInteraction(Uri uri) { }
}
```

The output of adding the fragments is shown in Figure 3.

In the next sections will see how to add some more features to our fragments, specifically menu items in the action bar.

# 5 Layouts and functionalities of the Fragments

Now that we have taken care of creating the three fragment that constitute the **MainAct ivity**, let's add some meaningful content to their layouts!

If you are running behind in time, you *can* skip parts 5.1 and 5.1.1 as they are related to layouts and **onClick** button responses, because you have already learned to do them previously. However, if you do so, don't forget to come back to them when you finish the rest of the lab.

## 5.1 NewRecordingFragment

We start by adding some elements to the **fragment_new_recording.xml**. In this fragment, we want to select the type of sport activity we want to track. Additionally, we want to select the external devices we wish to connect to in order to collect sensor data, which can be the smartwatch and Bluetooth heart-rate sensor.

If you open the **fragment_new_recording.xml**, there should be some automatically generated code by *Android Studio*. Delete this content and add a **ConstraintLayout** container. In this container, start by adding an **ImageView** (at the top center), and a **TextView** under it. Use the **running.png** image (to be downloaded from Moodle) for the **ImageView**. Remember that you can either drop it directly in the **res/drawable** folder, or alternatively add it as an image asset using the *Android Studio* asset generator, in which case the image will be available in the **res/mipmap** folder. Don't forget to give meaningful ids to the

widgets. Because these widget will display the chosen sport activity, you can name them **imageActivity** and **nameActivity**.

Now, add a **LinearLayout** under the **TextView**, set its id to **selectActivity**. In this **LinearLayout** we will place four **ImageButton**s. These buttons will be used to select which sport activity the user wishes to monitor. Use as source images the four png files available on Moodle (**running.png**, **cycling.png**, **skiing.png** and **climbing.png**). The Button should have the following ids respectively: **runningButton**, **cyclingButton**, **skiingButton** and **climbingButton**. Below is an example of the parameters used for the first **ImageButton**, use similar parameters for the other three **ImageButtons**:

```
<ImageButton
    android:id="@+id/runningButton"
    android:layout_width="0dp"
    android:layout_height="match_parent"
    android:layout_margin="4dp"
    android:layout_weight="1"
    android:adjustViewBounds="true"
    android:background="@null"
    android:padding="0dp"
    android:scaleType="fitCenter"
    android:src="@mipmap/running" />
```

Finally, add two **Switch**es one below the other, to control the remote sensors we consider: the smartwatch and a Bluetooth heart-rate sensor. Again here make sure to give meaningful identifiers to all the widgets. Figure 4 shows how the layout should approximately look like.

### 5.1.1 Setting onClick responses from the ImageButtons

Skip this part if you have already skipped part 5.1, but don't forget to come back to it at the end of the lab!

Since we have added a few **ImageButton**s in the **fragment_new_recording.xml**, we should implement their respective **onClick** methods. You have learned in the previous labs that this can be done either in the Java code or from the xml file. We propose here to use the Java code to set the **onClick** responses of the **ImageButtons**, but you are free to do it from the XML file if you prefer, you should be able to!

First let us explain what we want to happen when we click a certain **ImageButton**: the **ImageView** with the id **imageActivity** should take the image corresponding to the clicked activity, and the **TextView** with the id **nameActivity** should be set to its name. So let's

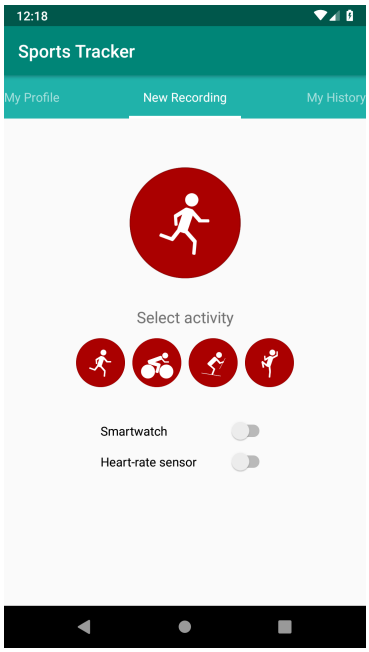


Figure 4: new recording fragment example layout

first create a generic function that does this, and takes as an input the ID of the activity. To do so, we will declare the following enumeration in the file **NewRecordingFragment.java**:

```
enum SPORT {RUNNING, CYCLING, SKIING, CLIMBING}
```

Below is the definition of the function, which goes in the same class:

```
public void setActivityNameAndImage(SPORT activity) {
    Drawable image = getResources().getDrawable(R.drawable.ic_logo);
    String name = "Select Activity";

    ImageView activityImage = fragmentView.findViewById(R.id.imageActivity);
    TextView activityName = fragmentView.findViewById(R.id.nameActivity);

    switch (activity) {
        case RUNNING:
            image = getResources().getDrawable(R.mipmap.running);
            name = "Running";
            break;
        case CYCLING:
            image = getResources().getDrawable(R.mipmap.cycling);
            name = "Cycling";
```

```
            break;
        case SKIING:
            image = getResources().getDrawable(R.mipmap.skiing);
            name = "Skiing";
            break;
        case CLIMBING:
            image = getResources().getDrawable(R.mipmap.climbing);
            name = "Climbing";
            break;
    }

    activityName.setText(name);
    activityImage.setImageDrawable(image);
}
```

Once again, the method **findViewById(...)** should be called from the current fragment using the **View** we saved as **fragmentView**, not the one from the parent activity!

Now let's implement the **onClick(...)** methods for our **ImageButton**s, we will do that inside the **onCreateView(...)** function of the fragment as follows in between inflating the layout in the fragment view, and returning it:

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                         Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    fragmentView = inflater.inflate(R.layout.fragment_new_recording,
            container, false);

    ImageButton buttonRunning = fragmentView.findViewById(R.id
            .runningButton);
    buttonRunning.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            setActivityNameAndImage(SPORT.RUNNING);
        }
    });

    ImageButton buttonCycling = fragmentView.findViewById(R.id
            .cyclingButton);
    buttonCycling.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View view) {
            setActivityNameAndImage(SPORT.CYCLING);
        }
    });

    ImageButton buttonSkiing = fragmentView.findViewById(R.id.skiingButton);
    buttonSkiing.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            setActivityNameAndImage(SPORT.SKIING);
        }
    });

    ImageButton buttonClimbing = fragmentView.findViewById(R.id
            .climbingButton);
    buttonClimbing.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            setActivityNameAndImage(SPORT.CLIMBING);
        }
    });

    return fragmentView;
}
```

Now each time you tap the button of a specific activity, the image and text on top of the layout should change accordingly. Test it out!

## 5.2 MyProfileFragment

The **MyProfileFragment** should contain all the information of the user, including his/her profile picture. So let's remove all the content that we previously copy-pasted from the **MainActivity**. Instead, we will copy the content of **activity_edit_profile.xml** and modify it slightly.

First, let's remove the two **FloatingActionButton** elements. Then change all the **EditText** elements to **TextView** elements. Use the following ids and values for the different **TextViews**:

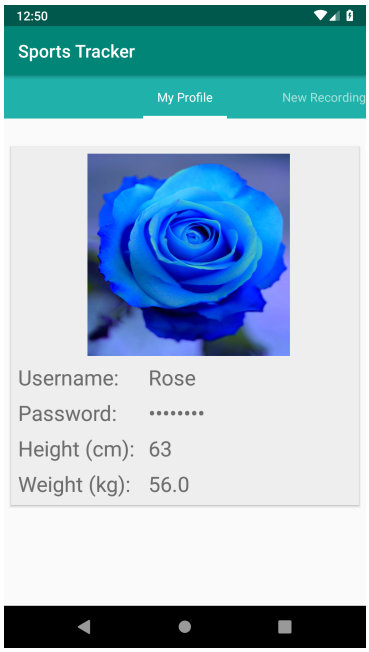


Figure 5: MyProfile fragment layout

| *id* | *text* |
|---|---|
| **usernameText** | ”Username:” |
| **passwordText** | ”Password:” |
| **heightText** | ”Height (cm):” |
| **weightText** | ”Weight (kg):” |

Next to Each of these widgets we will place another **`TextView`** which will contain the value of the all the fields. Use the following ids, values and input type (note that the password field should be hidden, hence its input type should be password):

| *id* | *text* | *inputType* |
|---|---|---|
| **usernameValue** | ”Username” | text |
| **passwordValue** | ”” | textPassword |
| **heightValue** | ”0” | text |
| **weightValue** | ”0” | text |

Figure 5 shows how the layout should look like.

As we have added additional widgets to this layout, we need to modify the code to properly fill them when opening the parent activity. You can refactor the previously existing **`setUserProfileInfo()`** private method accordingly.

```
private void setUserImageAndProfileInfo() {
```

```
    final InputStream imageStream;
    try {
        imageStream = new FileInputStream(userProfile.photoPath);
        final Bitmap selectedImage = BitmapFactory.decodeStream
                (imageStream);
        ImageView imageView = fragmentView.findViewById(R.id.loggedInUserImage);
        imageView.setImageBitmap(selectedImage);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    TextView Username = fragmentView.findViewById(R.id.usernameValue);
    Username.setText(userProfile.username);

    TextView passwordTextView = fragmentView.findViewById(R.id.passwordValue);
    passwordTextView.setText(userProfile.password);

    TextView UserWeight = fragmentView.findViewById(R.id.weightValue);
    UserWeight.setText(String.valueOf(userProfile.weight_kg));

    TextView UserHeight = fragmentView.findViewById(R.id.heightValue);
    UserHeight.setText(String.valueOf(userProfile.height_cm));
}
```

# 6 Adding an action bar Menu

Now we would like to add an **ActionBarMenu** first to **MyProfileFragment** and, then to **MyHistoryFragment** and **EditProfileActivity**. The app bar allows to add buttons for user actions. This feature lets you put the most important actions for the current context right at the top of the app. So let's do that.

## 6.1 ActionBarMenu for MyProfileFragment

Let's start by adding a **menu_my_profile.xml** in the **res/menu** folder of the mobile app. If this folder does not exist simply create it by right-clicking on the **res** folder then selection **New → Android Resource Directory**. Then create the XML menu file by right-clicking on the **res/menu** folder then selecting **New → Menu Resource File**. The created XML file will contain the elements of the menu. In this case we want to have one action button to

edit the user information and/or profile picture:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_edit"
        android:icon="@android:drawable/ic_menu_edit"
        android:title="@string/edit_data"
        app:showAsAction="ifRoom" />
</menu>
```

The option **app:showAsAction="ifRoom"** allows to always show the menu item as a button in the app action bar.

Now in the **MyProfileFragment.java** add the following line in the **onCreate(...)** method to enable the menu in the fragment:

```
setHasOptionsMenu(true);
```

Now you need to create the option menu, do it by adding the **onCreateOptionsMenu(...)** function as follows:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.menu_my_profile, menu);
}
```

## 6.2 ActionBarMenu for MyHistoryFragment

Following the same steps as above, add the action bar menu in **MyHistoryFragment.java**. The menu items will be defined in **menu_my_history.xml**. For now we will have one action button to clear the history, we will implement the actual action in upcoming labs. Below is the XML code for it:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
```

```
        android:id="@+id/action_delete"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="@string/delete_history"
        app:showAsAction="ifRoom" />
</menu>
```

## 6.3 ActionBarMenu for EditProfileActivity

Now that we know about menus, we will remove the validation button in the **EditProfileActivity** to set it in the action bar. The menu items will be defined in **menu_edit_profile.xml**. We will have two action buttons to clear the profile, and to validate it:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_validate"
        android:icon="@android:drawable/ic_menu_save"
        android:title="@string/validate_data"
        app:showAsAction="ifRoom" />

    <item
        android:id="@+id/action_clear"
        android:icon="@android:drawable/ic_menu_delete"
        android:title="@string/clear_data"
        app:showAsAction="ifRoom" />
</menu>
```

Now if you run the app, you should see the action buttons in the app bar like in Figure 6. In **EditProfileActivity** you should only override the method **onCreateOptionsMenu(...)**, as shown in the code below.

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_edit_profile, menu);
    return super.onCreateOptionsMenu(menu);
}
```

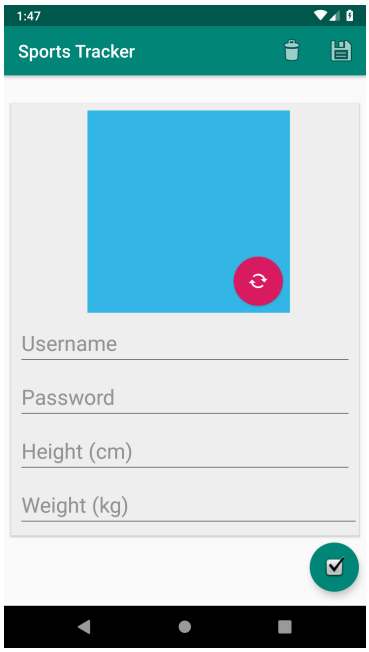


Figure 6: EditProfile Activity layout with ActionBarMenu

# 7 Reacting to menu interactions

Finally, let's add the responses to the menu item selection!

We will do so for **`MyProfileFragment`** and **`EditProfileActivity`** since the actions we wish to implement are well-defined. For now, the **`MyHistoryFragment`** will be left aside.

## 7.1 `MyProfileFragment` menu item selection

We want upon clicking on the edit action bar menu item of this fragment, to start the **`EditProfileActivity`** for result (like we do in **`LoginActivity`**). Let's implement that by overriding the **`onOptionsItemSelected(...)`** method as follows:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_edit:
            Intent intentEditProfile = new
            Intent(getActivity(),EditProfileActivity.class);
            intentEditProfile.putExtra("userProfile",userProfile);
            startActivityForResult(intentEditProfile,EDIT_PROFILE_INFO);
            break;
```

```
    }
    return super.onOptionsItemSelected(item);
}
```

Remember to save **EDIT_PROFILE_INFO** as private field in the **MyProfileFragment** class. Since we are starting an activity for result, we need to handle the response of the intent, like we did for **LoginActivity**:

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == EDIT_PROFILE_INFO && resultCode == AppCompatActivity.RESULT_OK)
        userProfile = (Profile) data.getSerializableExtra("userProfile");
        if (userProfile != null) {
            setUserImageAndProfileInfo();
        }
    }
}
```

Note here that **RESULT_OK** value is public static constant exported by the **AppCompatAct**⌋**ivity** class!

## 7.2 EditProfileActivity menu item selection

This **Activity** contains two action bar menu buttons, the first one is to clear the profile, and the second one to validate it (it will actually replace the validate button, which we ask you to remove from the XML layout, as well as any reference to it in the java code if there is any). We will implement the action of the menu items as follows by overriding the **onOptionsItemSelected()** function:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_clear:
            clearUser();
            break;
        case R.id.action_validate:
            editUser();
            break;
    }
```

```
    return super.onOptionsItemSelected(item);
}
```

And we define the method **clearUser()** as follows:

```
public void clearUser() {
    ImageView userImage = findViewById(R.id.userImage);
    TextView username = findViewById(R.id.editUsername);
    TextView password = findViewById(R.id.editPassword);
    TextView height = findViewById(R.id.editHeight);
    TextView weight = findViewById(R.id.editWeight);

    userImage.setImageDrawable(null);
    username.setText("");
    password.setText("");
    height.setText("");
    weight.setText("");
}
```

You are all set for this lab... or not just yet! We have a bonus question that we encourage you to try out, and believe you have learned all the necessary tools and methods to do it.

# 8 Bonus question

As a bonus, we would like to always pass the **userProfile** from and to activities and fragments that use it. For example, when opening the **EditProfileActivity**, the fields could be already filled by previous data if the user profile was already set before.

We are already passing the **userProfile** from the **LoginActivity** to the **MyProfileFragment**, you can do the same for the other activities and/or fragments!