

Lab 3 - Fragments, Orientation Changes

Based on <http://developer.android.com/guide/components/fragments.html>

In this assignment you will create an app that combines Activities and Fragments. A fragment is a self-contained component with its own UI and lifecycle; it can be reused in different parts of an application's user interface depending on the desired UI flow for a particular device or screen. We will work with simple Fragments containing only textual information, but, as you will see in your projects, these Fragments can contain much more complex elements.

According to the Android developer's website: "A Fragment represents a behavior or a portion of user interface in an Activity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities)." The easiest way to start understanding this is to see Fragments in action. In this lab, we will use Fragments to adapt the user interface according to the orientation of the screen. We will create an app for tourists in Ljubljana, and we will show the main menu of the app 1) on a separate screen from the content of the selected item, in case the device is set in portrait mode, or 2) on the same screen as the content of the selected item, in case the device is set in the landscape mode. See the figures below:

Orientation
Language
Zmajski Most (Dragon Bridge)
Kinoteka
Architecture Museum of Ljubljana
National Museum of Slovenia
Hills around the city

In the portrait mode, the user will see one screen with the options (see screenshot on the left), and after clicking any of the options will be shown a screen with the relevant information only (see screenshot on the right). To select another option, the user has to hit the "BACK" button, which reverts back to the options screen.

Zmajski Most (Dragon Bridge). Completed in 1901, designed by Croatian Jurij Zainovich. It is guarded by four detailed dragon statues from the city's coat-of-arms. Look out for the dragon motif throughout the city. Be careful around the Dragon Bridge area, as it is on a major busy road just outside the pedestrian zone and near misses (and worse) between inattentive tourists and traffic are common. The dragon bridge is located at the end of the Ljubljana Open Market, just a block or two down the river (north-east direction) from the Triple bridge.

In landscape mode, the user will see only one screen at all times. The right side of that screen, however, will change according to the option selected on the left.

Orientation	Zmajski Most (Dragon Bridge). Completed in 1901, designed by Croatian Jurij Zainovich. It is guarded by four detailed dragon statues from the city's coat-of-arms. Look out for the dragon motif throughout the city. Be careful around the Dragon Bridge area, as it is on a major busy road just outside the pedestrian zone and near misses (and worse) between inattentive tourists and traffic are common.
Language	The dragon bridge is located at the end of the Ljubljana Open Market, just a block or two down the river (north-east direction) from the Triple bridge.
Zmajski Most (Dragon Bridge)	
Kinoteka	
Architecture Museum of Ljubljana	
National Museum of Slovenia	

Let's code this!

First, create a project with an Empty Activity `MainActivity.java`, it should extend `AppCompatActivity` class. **MainActivity** needs two layouts - one for portrait, the other for landscape mode. By default Android will create a portrait layouts in `res/layout` folder, while for landscape layouts the XML files should be in `res/layout-land`. Create the missing folder (note: you might have to change your view mode - upper left - to Project in order to see the new folder) and add a brand new `activity_main.xml` to it. At runtime, the app will show one or another depending on the orientation.

Let's populate `activity_main.xml` in `layout` folder. Modify the XML so that `FrameLayout` is placed instead of the default layout, and within it add `fragment`. The fragment should have `android:name` property set to `"PACKAGE_NAME.TitlesFragment"`, where `PACKAGE_NAME` is the name of your app's package. This means the layout will show a frame (like a picture frame), within which a Fragment will be shown. That Fragment will be of type **TitleFragment**. We haven't created that class yet. Also, populate `android:id` fields of both the `FrameLayout` and the fragment with `"@+id/container"` and `"@+id/titles"` respectively.

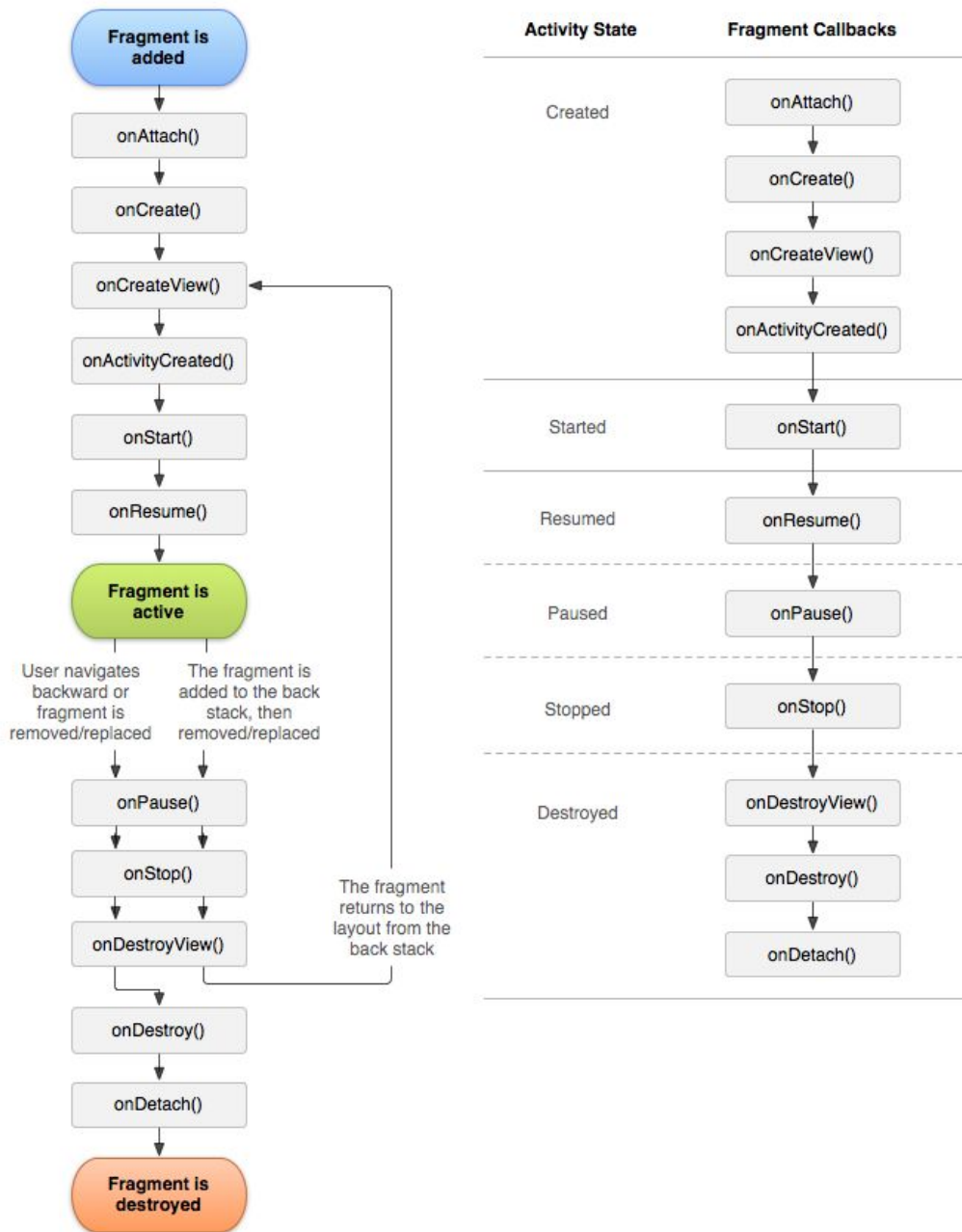
Now, let's populate the other `activity_main.xml`, the one that is located in `res/layout-land`. In this one add `LinearLayoutCompat`, instead of the default layout, and set its `android:orientation` to "horizontal". As its children add a `fragment` and a `FrameLayout`. The Fragment tag should have `android:name` property set to `"PACKAGE_NAME.TitlesFragment"`. For the fragment set the `android:id` field to `"@+id/titles"`. For the `FrameLayout` element, ensure that it has its `android:id` set to `"@+id/details"`. This means that the layout will show two elements one by one, horizontally, and that these elements are a **TitlesFragment**, and a frame (that we are yet to populate). To balance the width of the left and the right side of the layout, set `android:layout_width="0dp"` in both the fragment and the `FrameLayout`. Additionally, in each element set the `android:layout_weight` parameter to a value between 0 and 1 depending on the portion of space you want a particular part of the layout to take.

At this point we have our layouts defined, we should write some Java code to ensure that the layouts are drawn as needed, and populated with the tourist info.

First, let's go to **MainActivity**. Check whether `onCreate` sets the content view to `R.layout.activity_main`. The good thing here is that we do not have to specify which exact layout will be shown. Instead, Android will decide which one to show at the time the activity is created, depending on the phone's orientation.

Second, we need to provide a list of options. Add the content of `strings.xml` from `ucilnica` to your `res/values/strings.xml`. These are a few excerpts from the `Ljubljana Wikitravel` page (<http://wikitravel.org/en/Ljubljana>). Then, create **TitlesFragment** class (don't use the template!) and make it extend **ListFragment**. There are two **ListFragment** classes that you can extend from, use the Androidx one, as it supports a wider range of devices. **ListFragment** is just like a regular Fragment class but with a built in support for list sources. We will use a list of entries to populate the fragment, so extending from **ListFragment** makes sense.

Now is a good time to examine the Fragment lifecycle. Carefully analyse the figure below. Fragment lifecycle is similar to Activity lifecycle. However, a Fragment lifecycle kicks in only if it is attached to an Activity. A Fragment is visible (resumed) only when it is connected to a resumed Activity.



Create two private fields in the **TitlesFragment** class:

```
boolean mDualPane;
int mCurCheckPosition = 0;
```

mDualPane will help us track whether we can show two panes one by one. It is set to true only if R.id.details is shown. The other variable mCurCheckPosition keeps track of the currently checked list entry.

The fragment is populated when the corresponding activity is created, thus, in the **TitlesFragment** class we override onActivityCreated. Add the following code:

```
super.onActivityCreated(savedInstanceState);

// Populate list with our static array of titles.
setListAdapter(new ArrayAdapter<String>(getActivity(),
```

```

        R.layout.support_simple_spinner_dropdown_item,
        getResources().getStringArray(R.array.titles)));

// Check to see if we have a frame in which to embed the details
// fragment directly in the containing UI.
View detailsFrame = getActivity().findViewById(R.id.details);
mDualPane = detailsFrame != null && detailsFrame.getVisibility() ==
                                                    View.VISIBLE;

if (mDualPane) {
    // In dual-pane mode, list view highlights selected item.
    getListView().setChoiceMode(ListView.CHOICE_MODE_SINGLE);
    // Make sure our UI is in the correct state.
    showDetails(mCurCheckPosition);
}

```

When a user clicks on an entry in the landscape mode we want to show the corresponding text in “details” frame. We will override `onListItemClick` and call `showDetails` function. `onListItemClick` gives us an index of the clicked element, so we will just pass that index to `showDetails`.

Let's write `showDetails`:

```

void showDetails(int index) {
    mCurCheckPosition = index;

    if (mDualPane) {
        // We can display everything in-place with fragments.
        // Have the list highlight this item and show the data.
        getListView().setItemChecked(index, true);

        //Make new fragment to show this selection.
        DetailsFragment details = DetailsFragment.newInstance(index);

        // TODO: here you should replace the existing fragment in
        // “details” FrameLayout with the new fragment.
        // hint: use FragmentManager, don't forget to commit!

    } else {
        // Otherwise we need to launch a new activity to display
        // the details fragment with selected text.
        Intent intent = new Intent();
        intent.setClass(getActivity(), DetailsActivity.class);
        intent.putExtra("index", index);
        startActivity(intent);
    }
}

```

What does the above function do? First, it checks `mDualPane` to see if it needs to operate in the landscape mode or not. If so, it will find a spot for a **DetailsFragment** in `R.id.details` (a frame, like

a picture frame), and then it will either keep the original content, or create a new **DetailsFragment** based on the selected list index. This new instance will then replace the old details. **FragmentManager** is used for switching fragments in and out. Don't forget to call `commit` when you want to finalise the changes. If the app does not operate in the landscape mode, we don't show the selection fragment on the same screen as a details fragment, so we can use separate Activities. Thus, in this case the app launches a new activity (**DetailsActivity**) and passes the index in the extras.

Now, we need that **DetailsFragment**. Create a new class and make it extend `Fragment` (from the `AndroidX` library). We will create this fragment via **TitlesFragment**, which passes the index of the entry that the user has selected. To capture this, we will use `newInstance(int)` as a factory function for creating **DetailsFragment**, like this:

```
public static DetailsFragment newInstance(int index) {
    DetailsFragment f = new DetailsFragment();

    // Supply index input as an argument.
    Bundle args = new Bundle();
    args.putInt("index", index);
    f.setArguments(args);

    return f;
}
```

This creates a new **DetailsFragment** and saves the index in `Fragment`'s arguments. Each fragment has the ability to save arguments, and these arguments are preserved in case, for example, the orientation changes. We can, therefore, think of our **DetailsFragments** as cards with different tourist info entries for Ljubljana that we will dynamically show to the user as she selects titles from **TitlesFragment**. OK, so let's proceed with showing the data. Add the following two methods to **DetailsFragment**:

```
public int getShownIndex() {
    return getArguments().getInt("index", 0);
}

@Override
public View onCreateView(LayoutInflater inflater,
                        ViewGroup container, Bundle savedInstanceState) {
    if (container == null) {
        // Currently in a layout without a container, so no
        // reason to create our view.
        return null;
    }

    ScrollView scroller = new ScrollView(getActivity());
    TextView text = new TextView(getActivity());
    int padding = (int) TypedValue.applyDimension(
        TypedValue.COMPLEX_UNIT_DIP,
        4, getActivity().getResources().getDisplayMetrics());
    text.setPadding(padding, padding, padding, padding);
    scroller.addView(text);
    text.setText(
```

```
getResources().getStringArray(R.array.descriptions)[getShownIndex()]);
    return scroller;
}
```

This completes the landscape portion of our app. The next step is to create **DetailsActivity**, extending **AppCompatActivity**, which will show the right content to the user in case of the portrait mode. Override `onCreate` and ensure that the Activity changes fragments as needed, by putting the following code somewhere in the method:

```
DetailsFragment details = new DetailsFragment();
details.setArguments(getIntent().getExtras());
getSupportFragmentManager().beginTransaction().add(android.R.id.content,
details).commit();
```

Also, make sure that the Activity is destroyed if we go to the landscape mode (we don't need it any more in that case, as we can show everything via the MainActivity and fragments). Add this to `onCreate`:

```
if (getResources().getConfiguration().orientation
    == Configuration.ORIENTATION_LANDSCAPE) {
    // If the screen is now in landscape mode, we can show the
    // details in-line so we don't need this activity.
    finish();
    return;
}
```

Now test your app and try to improve it. Here is one suggestion: save a user's choice in the `TitlesFragment` so it is not lost when the `Fragment` is destroyed.

Happy coding!