Lab 2: ListView with Fragments

Design and Development of Mobile Applications
E. De Coninck, S. Leroux, P. Simoens
2015-2016

Goal

In this session we will create an app that shows a (static) list of superheroes. The details of a superhero are shown when the user touches the corresponding item. Your app will have a different appearance in landscape and portrait. In portrait mode the list is shown in the main activity and the details in a separate activity. In landscape mode there is a single activity containing the list on the left and the details on the right. Based on these requirements we need two fragments (for the list and for the details) and two activities to be added to the APK of our app.

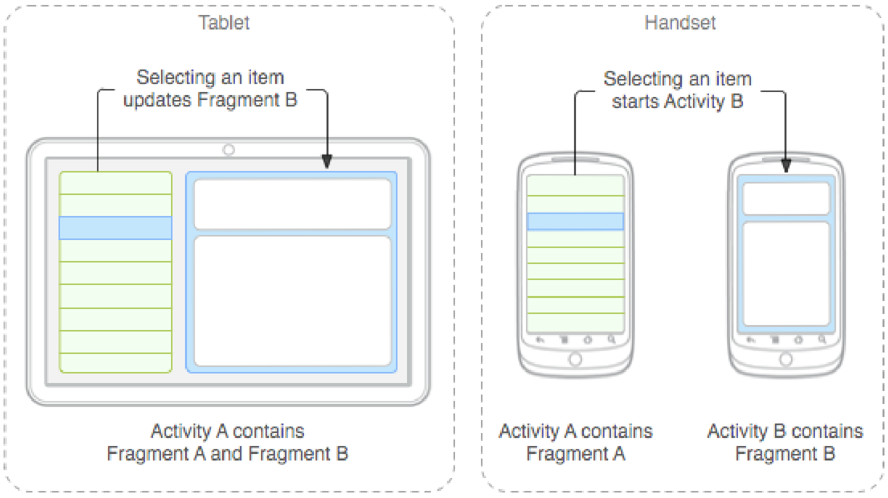
Tablet
Selecting an item updates Fragment B
Activity A contains Fragment A and Fragment B
Handset
Selecting an item starts Activity B
Activity A contains Fragment A
Activity B contains Fragment B

Figure 1: An example of how two UI modules defined by fragments can be combined into one activity for a tablet (and/or landscape) design, but separated for a handset (and/or portrait) design.

1 Fragments basics and lifecycle

In the first part of this session, we will create the launcher activity and define a fragment to display the list of superheroes.

- Create a new project and add a blank activity (without fragments) called `MainActivity` as launcher activity. Override all lifecycle methods and add log messages (like you did in the previous lab session). You are free to choose any name for the XML layout file. For convenience, in the remainder of this document we will assume you have chosen the name `activity_main.xml`.
- Create a new blank fragment called `MainFragment` **without** XML layout, factory methods or interface callbacks. Add this fragment to a `<RelativeLayout>` root container in your `activity_main.xml`

layout file by adding the appropriate `<fragment>` tag. This tag has two required attributes: a unique `android:id` and an `android:name` that refers to the file name of your `Fragment` class implementation. For example: `be.ugent.oomt.lab2.MainFragment`. If the class is in your default application package, you can shorten the value of the `android:name` tag to the class name prepended with a dot (.).

By adding the fragment to the XML layout of the encompassing activity, the fragment will be automatically instantiated when the `MainActivity` is created. This is our intended behavior, since we want the list to be displayed in any orientation.

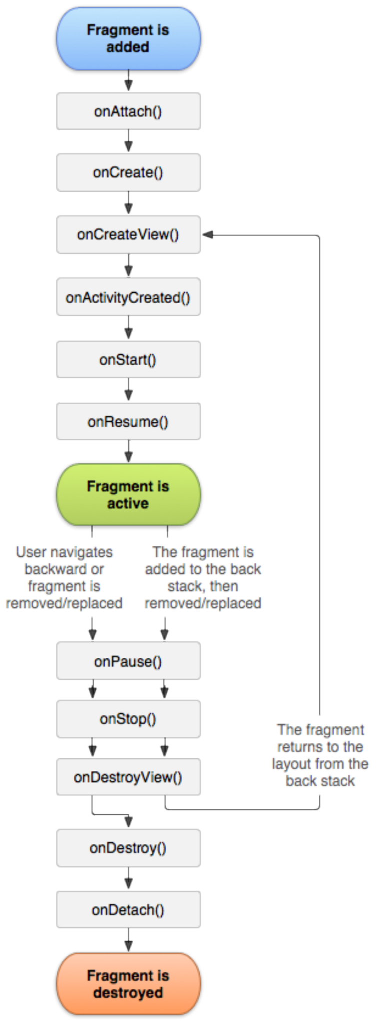

Figure 2: The lifecycle of a fragment when it is embedded in a running activity.

- Override all lifecycle methods of the `Fragment` class and add log statements.

Question: What is the correlation between the lifecycle callbacks of the fragment and its encompassing activity?

Question: Which lifecycle methods of the activity and the fragment are called on device rotation?

2 Dynamically adding a fragment

Currently, our app only shows one fragment even when our device is in landscape mode. We will now add a second lay-out file for our main activity, that leaves room at the right side of the display for a fragment with detailed information of a selected superhero. In portrait mode, you will display these fragments in a second activity (`DetailActivity`).

- Create a new blank fragment called `DetailFragment` **with** an XML layout but without factory or interface callbacks. This fragment contains a `ScrollView` as the root `ViewGroup` with one vertically oriented `LinearLayout`. The `LinearLayout` contains two `TextViews`, one for the name and one for the history of our superhero.
- Add a new resource layout file for landscape orientation of the launcher activity. This file must have the same name as your first lay out file (`activity_main.xml`) and you have to add an orientation qualifier for landscape orientation. The new layout file should consist of a horizontal `LinearLayout` with on the left (1/4 of the screen) the `MainFragment` and on the right (3/4 of the screen) a `FrameLayout` with a required attribute id (For example: `container`). We will use this container to add fragments dynamically. Hint: use `android:layout_weight`.
- To test your current set-up, create a new instance of `DetailFragment` and put it in the `FrameLayout` container of the `MainActivity` when the activity is created. Don't do this by adding a `fragment` tag to the XML layout file, instead use the `FragmentManager`.

  Question: why don't we use the XML approach here?

  Hints:

  - study the difference between the add and `update` methods of the `FragmentManager`
  - the `FrameLayout` is only available in landscape mode! In portrait mode, you will encounter null pointer exceptions if you try to put a fragment in this container.

Question: There are two possibilities to prevent these null pointer exceptions. Either you check the presence of the `container` element, either you check for the device orientation. What is the most appropriate strategy and why?

Question: What is the most appropriate location in your code to perform this check?

3 Finalizing the app

In the last part, we will create and populate the list and specify the lay out for displaying details of a selected superhero. Then we will add the second activity that is only used in portrait mode.

- The `MainFragment` will show a list of superhero names. Showing a list is a common pattern for a fragment, so Android has built-in support for this. Extend the `ListFragment` class instead of `Fragment` and remove the `TextView` instantiated in the `onCreate()` method.
- Each list needs a list adapter that holds the data and knows how to lay out individual list items. We will use a simple `ArrayAdapter` which reads the data from a resource file. Download the superheroes file from Minerva and add it to your `values` resource folder. You can access resource arrays with the `getResources().getStringArray(id)` method. The id is stored in R.array.

  The `ArrayAdapter` has three parameters (context, resource id and the string array). The resource id is a reference to the layout file for list items. Android provides some default layouts for list items, we will be using `android.R.layout.simple_list_item_activated_1`. At this point you should be able to view the superhero names in the list fragment.
- Create a new blank activity called `DetailActivity` to show the `DetailFragment` when a user selects a superhero and is in portrait mode. Do not forget to select the hierarchical parent in the wizard. This is used to enable 'Up' navigation in your application. To display the detail fragment you use `FragmentManager` instead of the generated layout file. Make sure that the correct fragment is shown when the activity is displayed. For this, you need to forward to your `DetailFragment` the arguments that

were passed to the `DetailActivity`. You can use `android.R.id.content` as id for the container where you add the fragment to with the `FragmentManager`. This id always gives you the root `ViewGroup`, so you do not need to know name/type/ID of any other element. It is typically used in fragment transactions. Do not add the fragment to the XML layout file of the activity.

- Finish this activity when it is viewed in landscape orientation, to avoid that a large detail fragment is shown. Instead, switch back to the main activity.

Question: There is one specific case where this behaviour happens. Which one?

- Override the `onListItemClick()` method of `ListFragment` (in `MainFragment`). Note the inconsistency in the naming, since this method will be called if the users *touch* an item rather than *click*ing on it. Implement this method with the following requirements:
  - In portrait: start a new `DetailActivity` and pass the index of the selected item.
  - In landscape: replace the current `DetailFragment` with `FragmentManager` and pass the index of the selected item.

  In your `DetailFragment` get the index from the arguments and set the title and history of the selected superhero.
- Currently your application forgets the selected superhero on recreating the main activity, e.g. on device orientation change. Save the current selection in the state of your fragment (`onSaveInstanceState`) and retrieve this information when (re)starting the fragment.
- Set the choice mode of your list view to `ListView.CHOICE_MODE_SINGLE` so the selection is highlighted when the details are shown.
- Finalize your application by adjusting font size, padding, margins, etc. to have a nice, user-friendly layout in both orientation modes.