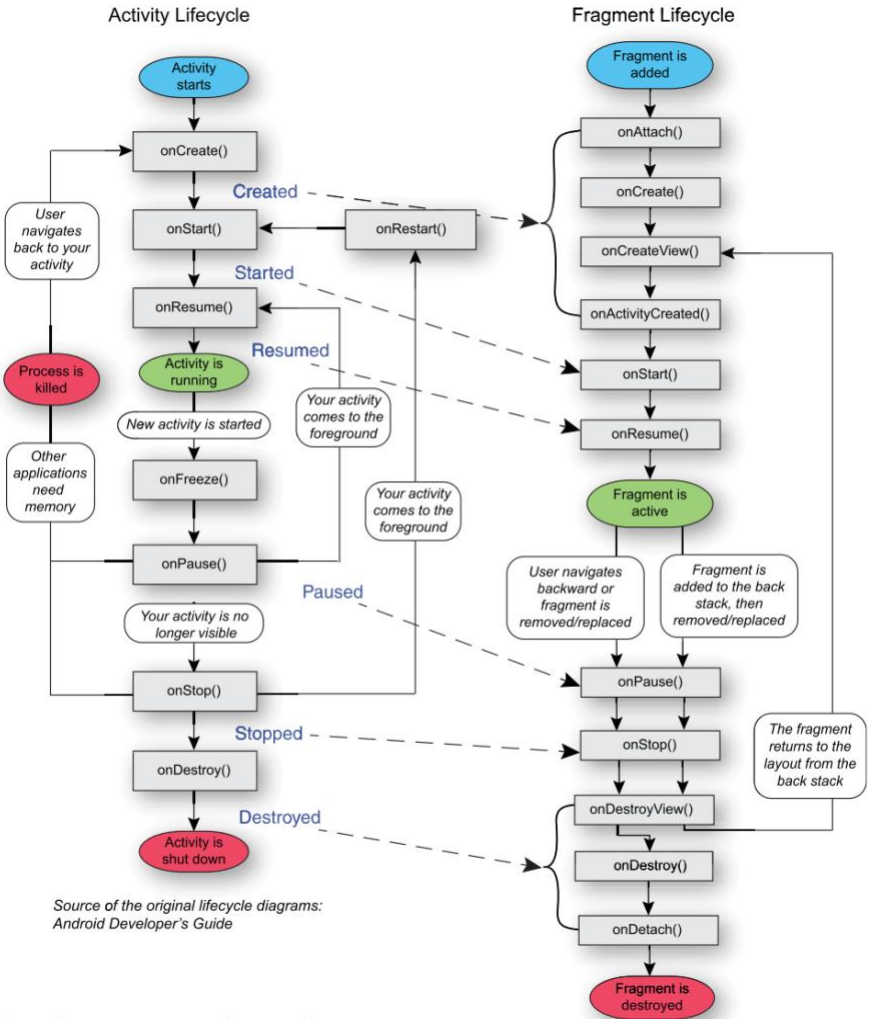
CMPS 312 Mobile Application Development

LAB 7: Fragments

Objective

In this lab you will learn how to use fragments to represents a behavior or a portion of user interface in an Activity. By the end of the lab you will learn,

1. How to combine multiple fragments in to a single activity to build a multi-pane UIs
2. How to reuse fragments in multiple activities,
3. How to send data from Activity to Fragments, from Fragment to Activity and from Fragment to Fragment.
4. How to add fragments statically and dynamically,
5. And the lifecycle of fragments



Overview

You are required to apply the concept shown in Figure 1 by using fragments. You will using the solar application that you created in Lab 4.

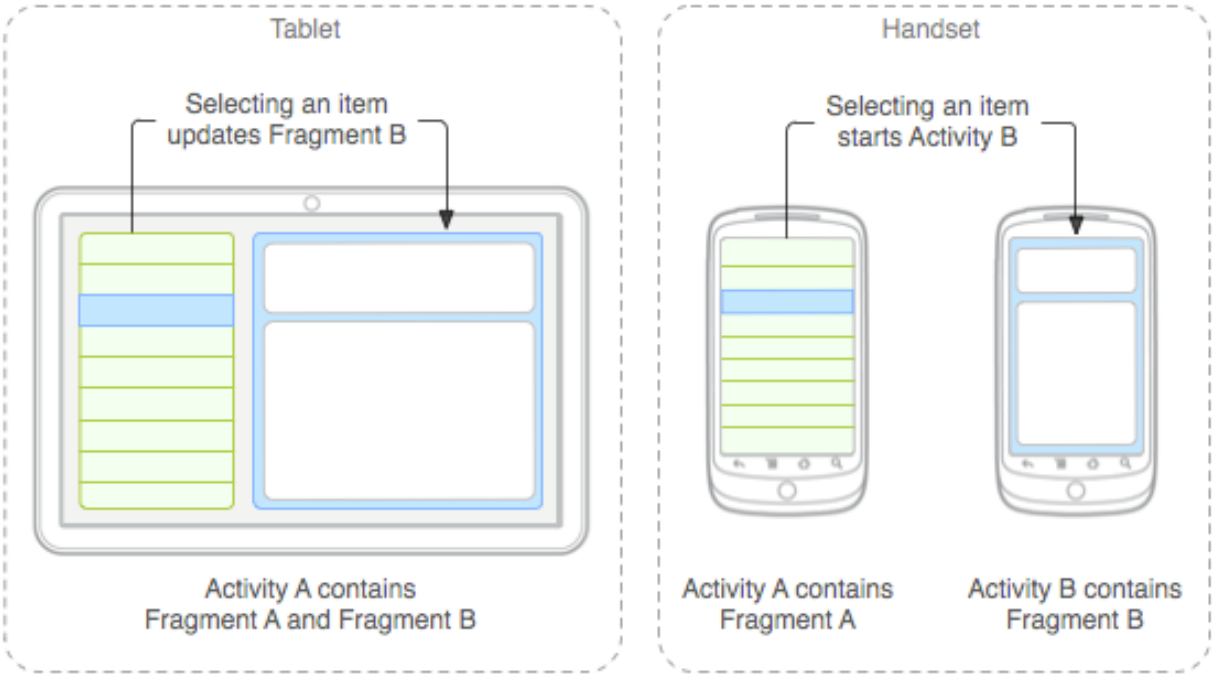


Figure 1 An example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design. [from: https://developer.android.com/guide/components/fragments.html]

PART A – Tutorial

In PART A you will learn how to work with fragments. We will be creating the following simple application that shows how to add, delete , replace a fragment

Using xml to create fragment

1. Create a new project and name it "**FragmentTutorial**"
2. Make a copy of the XML file of the main activity "activity_main.xml" and name the new copy "**new_fragment_one"**
   ➔**[This is to show you a fragment and activity are the same in terms of design]**
3. Create a java class and call it "**MyNeFragmentOne**" and make it extend **Fragment** from adroidx
4. Create the default constructor for the MyNewFragmentOne class
5. Override the following two methods onCreate and onCreateView
6. Inflate the fragment layout inside the onCreateView
7. Once you complete your code should look like this

```
public class MyNewFra  Unused import statement  Fragment {

    public MyNewFragmentOne() {
        // Required empty public constructor
    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.new_fragment_one, container,  attachToRoot: false);
        return rootView;
    }

}
```

8. Once you complete your fragment java code , go back to the activity_main.xml layout and inject this fragment into the layout

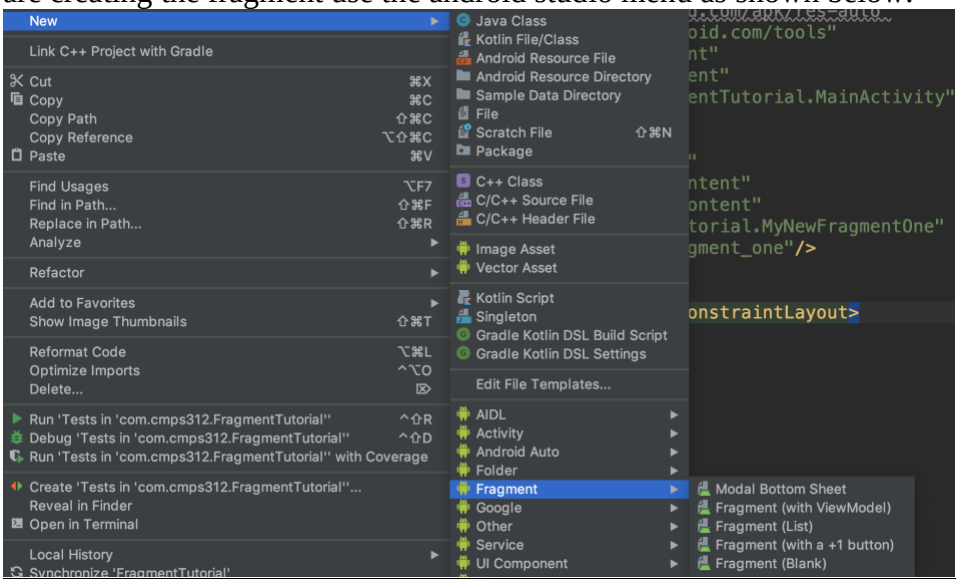
```
<fragment
    android:id="@+id/fragment_one"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    class="com.cmps312.FragmentTutorial.MyNewFragmentOne"
    tools:layout="@layout/new_fragment_one"/>
```

9. Run your code

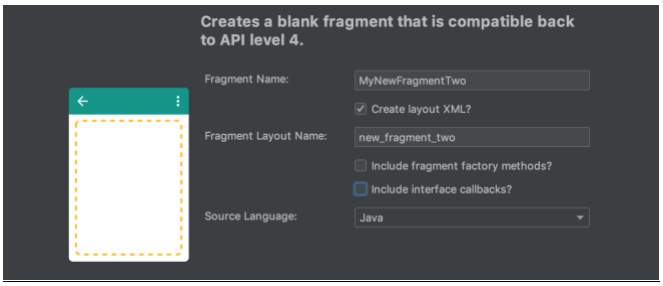
DYNAMICALLY ADDING FRAGMENT USING FRAGMENT TRANSACTION

Now that you know how to add fragments using XML let us modify the previous code add a new fragment dynamically using the fragment transactions.

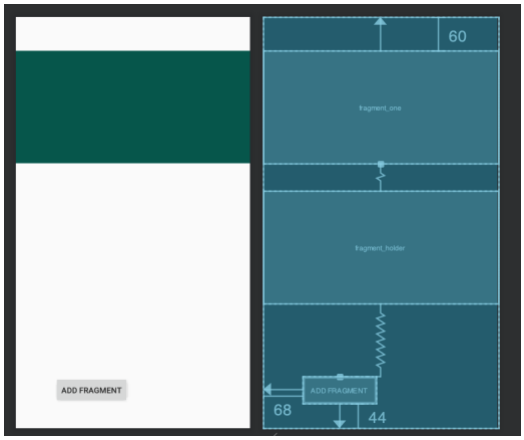
1. Create a new fragment class and call it "**MyNewFragmentTwo".** This time when you are creating the fragment use the android studio menu as shown below.



2. Change the fragment and layout names to the following names. Also, do not forget to remove the two checked boxes that include the callback and factory methods.



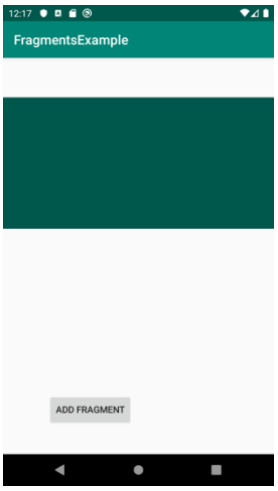
3. Modify the **activity_main.xml** and add a **FrameLayout** with an id **fragment_holder** and a Button called **Add Fragment**. The design should look like the image shown on the right side. Note the framelayout is not showing but if you look at the blue screen you will be able to see it.
4. Register an onClick listener for the add fragment button

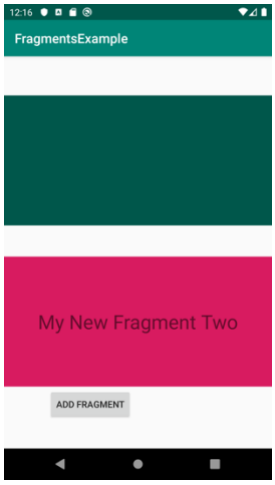


5. In the main activity add the **MyNewFragmentTwo** once the user clicks on the add Button. The code is shown below.

```
public void addFragment(View view) {
    MyNewFragmentTwo fragmentTwo = new MyNewFragmentTwo();
    getSupportFragmentManager().beginTransaction()
            .add(R.id.fragment_holder, fragmentTwo)
            .commit();
}
```

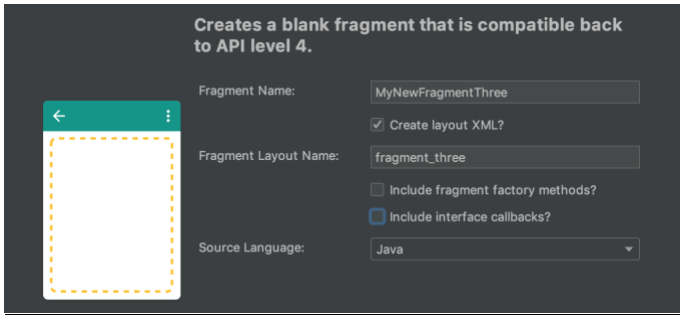
6. Run your application and test. Once you click on the add fragment you should see the second fragment on the screen.



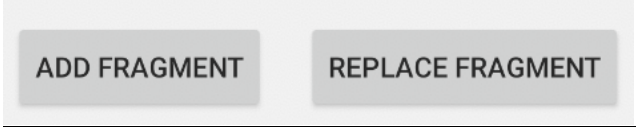


REPLACING FRAGMENT DYNAMICALLY

1. Create a new fragment class and call it "**MyNewFragmentThree".** Use the same technique you used in the previous section to create the fragment.



2. Add a new button to the activity_main.xml and call it "**Replace Fragment**" next to the add button.



3. When the user presses on the replace fragment button , change **MyNewFragmentTwo** to **MyNewFragmentThree**. The code you will be writing here will be exactly the same as the add. All you need to replace is the **add** method to **replace**.

```
public void replaceFragment(View view) {
    MyNewFragmentThree fragmentThree = new MyNewFragmentThree();
    getSupportFragmentManager().beginTransaction()
            .replace(R.id.fragment_holder, fragmentThree)
            .commit();
}
```

4. Test your code.

ADD FRAGMENT TO BACKSTACK

Android provides back stack functionality to fragments. In order to activate this feature you should call the **fragmentTransaction.addToBackStack(null) or (fragmentName)** method when executing the fragment transaction in your replace/add methods.

```
public void addFragment(View view) {
    MyNewFragmentTwo fragmentTwo = new MyNewFragmentTwo();
    getSupportFragmentManager().beginTransaction()
            .add(R.id.fragment_holder, fragmentTwo)
            .addToBackStack(null)
            .commit();
}
```

COMMUNICATION FROM ACTIVITY TO FRAGMENT

1. Modify **MyNewFrgamentTwo** class by overriding the following the **newInstance** factory method inside.

```
public static MyNewFragmentTwo newInstance(String message) {

    Bundle args = new Bundle();
    args.putString("message", message);

    MyNewFragmentTwo fragment = new MyNewFragmentTwo();
    fragment.setArguments(args);
    return fragment;
}
```

2. Inside the onCreate method read the passed arguments and safe them into a member variable called message [**message = getArguments().getString("message");]**
3. Inside the OnCreateView display the message into the **textview** of **MyNewFrgamentTwo**
4. Modify the code of the MainActivity addFragment Method and use the newInstance method instead of the default constructor
5. Pass any message that you would like to the newInstance factory method
6. Test your code

COMMUNICATION FROM FRAGMENT TO ACTIVITY

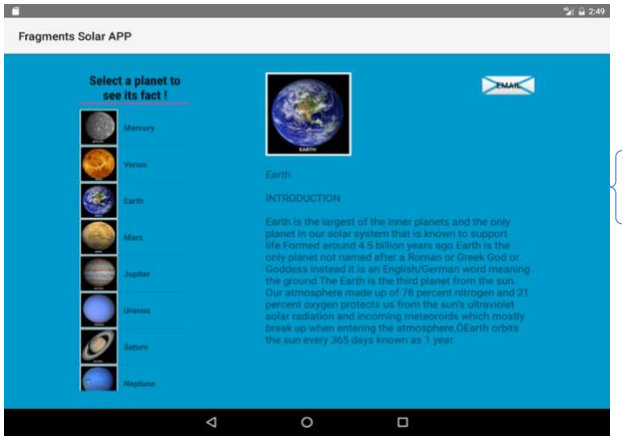
1. Add an interface class to the **MyNewFrgamentTwo** class and call it **FragmentTwoListner**
2. Inside the interface declare a single method called **sayHello(String message)**
3. Create a field of type **FragmentTwoListner** and name it **listener**
4. Instantiate the **lister** inside the **onAttach** Method by assigning it to the **context**
5. When the user clicks on the textview send a message by calling **lister.SayHello("Sometext")**
6. Implement **FragmentTwoListner** interface in the main activity.
7. Inside the method **sayHello** toast the message that was sent by the **MyNewFrgamentTwo**

EXERCISE 1

Now that you learned how to communicate **from activity to fragment** and **from fragment to an activity**, try to create the following simple application that sends message from one fragment to another fragment and vice versa.

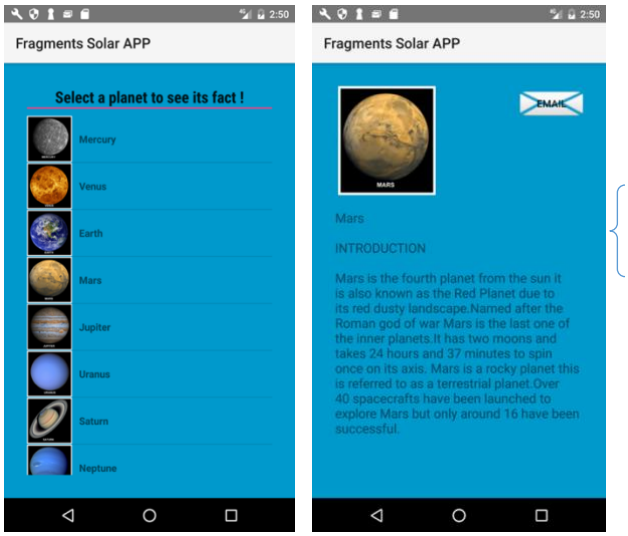
EXERCISE - 2

In this exercise you will modify the "**Solar application**" provided to you. This application is the same application that you created in "**LAB 4 intents and permission**" . The application displays two fragments side by side when it is on tablet as shown below. And it displays a single fragment when it is on phone. The list is implemented using Recycler View. All the assets are provided for you under the lab folder.



On Tablet

Figure 2-A Tablet View



On Handset

Figure 2-B Handset View

Figure 2: Solar APP using fragments and recyclerView

CONSTRAINTS

When the application is in tablet it should show both fragments side by side as shown in figure 2-A and Figure 1(Tablet). However, when the application is in handset you will only show a single fragment inside the entire Activity as shown in Figure 2-B and Figure 1(Handset). The general

flow of the application is shown in the overview section Figure 1 and the final look of the Application is shown in Figure 2 above.

Important Note: Do not use any template. You should implement everything on your own.