Name: _____

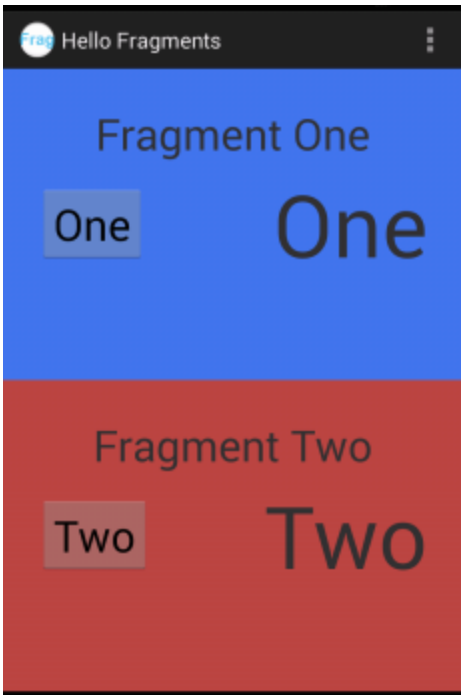
ROSE-HULMAN INSTITUTE OF TECHNOLOGY **CSSE483 Android Application Development**

Lab 6: Hello Fragments

The purpose of this lab is to learn how to use Fragments in an app. To get this lab checked off, show the finished app to your instructor or assistant.

Introduction

Your goal is to make an app that uses two fragments in the same activity and view that communicate with each other. Pressing the button in the top fragment will adjust the text field in the bottom fragment. Pressing the button in the bottom fragment will randomize the color of the top fragment.



This app will have one Activity, two Fragments and two Interfaces.

MainActivity	Activity that controls the fragments
FragmentOne	Top fragment with a button and text field

FragmentTwo	Bottom fragment with a button and text field
OnDataPassToTwoListener	Callback interface used for communicating from the fragment to the activity
OnRandomizeColorListener	Callback interface used for communicating from the fragment to the activity

Part A: Creating the Layouts

1. Activity_main.xml - This xml file will simply contain two FrameLayouts. FrameLayouts are used to block out an area of the screen for one particular item.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >

    <FrameLayout
        android:id="@+id/fragment_one_container"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" >
    </FrameLayout>

    <FrameLayout
        android:id="@+id/fragment_two_container"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1" >
    </FrameLayout>

</LinearLayout>
```

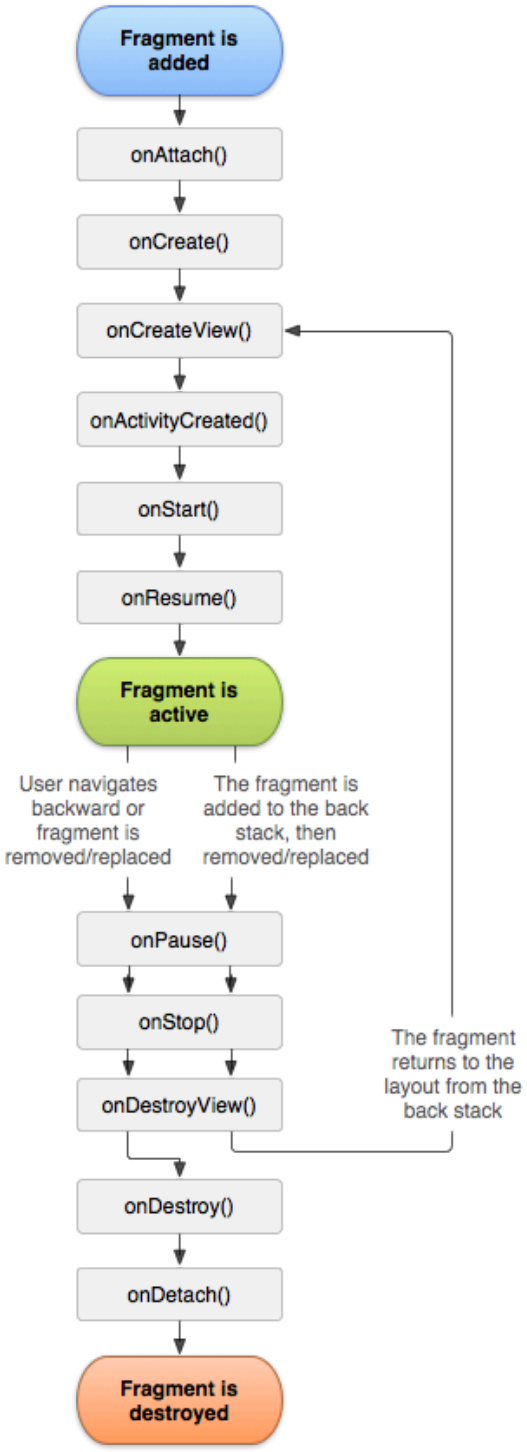
Notice that we set the width to fill the entire screen, and the height just to wrap_content. The layout_weight field will ensure that both FrameLayouts are allotted equal space. Later we will add the fragments to these FrameLayouts in code.

2. fragment_one.xml - The individual fragment layouts will contain a TextView that displays the title of the fragment (Fragment One or Fragment Two), a button and a TextView beside the button. Use a RelativeLayout here within the Fragment. Since we will put this entire layout into half of a screen, this will allow for easier adjustment. Add the colors blue and red (“#47e and #b44, respectively) to a new colors.xml file and set the view’s background using the “android:background” field.Make the sizes of the text 36 sp (and 72 sp for the text to the right).

3. fragment_two.xml - This file will be identical to the fragment_one.xml, except with the adjusted text and a red background.

Part B: Fragment lifecycle

The following diagram shows the lifecycle of a fragment, from creation to destruction. Some of the methods are familiar, like `onCreate()`, and some are new. `onAttach()` is particularly important, since it allows us to associate the creating activity to the fragment.



Part C: Creating the Fragment Classes

Create two classes, FragmentOne.java and FragmentTwo.java that extend Fragment. For now we will simply have these fragments inflate their appropriate view. A fragment's

`onCreateView()` method is called whenever the Fragment is created (or the view needs to be refreshed). This is the natural place to inflate the view from xml. The `onCreateView()` method of FragmentOne.java will look like this.

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_one, container, false);
}
```

Add the appropriate onCreateView() method to FragmentTwo.java as well.

Part D: Adding the Fragments to Containers

Since we will be using the fragments later on, we will create fields for the fragments in MainActivity.java.

```
private FragmentOne mFragmentOne;
private FragmentTwo mFragmentTwo;
```

We will now add the fragments to their appropriate FrameLayout containers in the onCreate() method of MainActivity.java, and assign these fragments to our fields.

We will use a FragmentManager to create a new FragmentTransaction. A FragmentManager manages the Fragment stack (to move back to fragments that have been previously viewed) and creates FragmentTransactions. The FragmentTransaction is a set of changes you want to make to the FragmentManager at any given time.

Add the following code to `onCreate()`.

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction ft = fragmentManager.beginTransaction();
```

Next we will create instances of the Fragment classes we made in Part B.

```
mFragmentOne = new FragmentOne();
mFragmentTwo = new FragmentTwo();
```

Now we need to add these fragments to the FrameLayout containers.

```
ft.add(R.id.fragment_one_container, mFragmentOne);
ft.add(R.id.fragment_two_container, mFragmentTwo);
```

Finally every FragmentTransaction has to be closed off with a commit.

```
ft.commit();
```

The whole method should look like this:

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    FragmentManager fragmentManager = getFragmentManager();
    FragmentTransaction ft = fragmentManager.beginTransaction();

    //When we create the fragment, it's onAttach method is called, associating it with this activity
    mFragmentOne = new FragmentOne();
    mFragmentTwo = new FragmentTwo();

    ft.add(R.id.fragment_one_container, mFragmentOne);
    ft.add(R.id.fragment_two_container, mFragmentTwo);
    ft.commit();

}
```

At this point you should be able to run the application and see the two fragments shown together:

Part E: Creating the callback interface

In order to communicate with the Activity from each fragment, we will need to implement a *callback interface*. The MainActivity class will be required to implement this interface, and each of the fragments will have a field for an instance of the interface.

First create an interface called OnDataPassToTwoListener.java. This class will have only one method, which will pass data from FragmentOne to FragmentTwo. The whole interface should look like this:

```
public interface OnDataPassToTwoListener {
    public void passDataToTwo(String data);
}
```

Now we tell our MainActivity class to implement this interface.

```
public class MainActivity extends Activity
    implements OnDataPassToTwoListener,
```

Leave its required methods empty for the time being, since we first have to make some adjustments to our Fragments.

Part F: Communicating with the calling Activity

Now that our MainActivity implements the OnDataPassToTwoListener, we need to adjust our Fragments. Starting with FragmentOne.java, create an OnDataPassToTwoListener private field.

```
private OnDataPassToTwoListener mListener;
```

Now we will be using a new method to associate this listener with the Activity that creates this fragment. The “onAttach” method is called once the fragment is associated with its activity. Even though we know which activity is creating the fragment, we will use good form and surround this assignment with a try/catch statement. Add the following code into your FragmentOne.java class.

```
@Override
public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
                mListener = (OnDataPassToTwoListener) activity;
        } catch (ClassCastException e) {
                throw new ClassCastException(activity.toString()
                                + " must implement OnDataPassToTwoListener");
        }
}
```

We are casting the creating activity to an OnDataPassToTwoListener, therefore the creating activity must implement OnDataPassToTwoListener. Now that we have the correct interface saved, we need to make some other changes to FragmentOne.

Start by having FragmentOne implement OnClickListener, and associate the button you created in fragment_one.xml with this listener. Since we are in a fragment we do not have access to the

findViewById method directly, therefore add code like the following to the onCreateView() method:

```
View v = inflater.inflate(R.layout.fragment_one, container, false);
((Button) v.findViewById(R.id.button1)).setOnClickListener(this);
return v;
```

Instead of directly returning the view, we are temporarily assigning it to a variable. We then use that variable to gain access to the necessary findViewById method (you have done this with Dialogs in the past).

Since we implemented OnClickListener we need to add in the familiar onClick method. Here we will use that saved interface and pass the appropriate data to the second fragment.

```
public void onClick(View v) {
    mListener.passDataToTwo(getString(R.string.button_one));
}
```

Currently the method in MainActivity corresponding to this is empty, but we will fix that shortly.

We need one more change: we need to change fragment two's TextView when fragment one's button is pressed, so we need a method that the activity can call to do this. Add in the following setText method to FragmentTwo.

```
public void setText(String text) {
    ((TextView) getActivity().findViewById(R.id.textView1)).setText(text);
}
```

Once again, notice how we cannot use findViewById directly. However, since the fragment is attached to an activity, we can use getActivity() to access the findViewById method.

Now have passDataToTwo() call the setText() method. **Clicking the top fragment's button should cause the second fragment's text to change.**

This is kind of boring, so you'll probably feel the urge to change what you pass to two so that it changes each time you click the button in FragmentOne - for example, the count of how many times the button was clicked.

Part G: Implement Random Color Changes

Your turn to practice. When the bottom fragment's button is clicked, cause it to change the color of the top fragment to a **random** color. Follow the same process, but have MainActivity implement this interface also:

```
public interface OnRandomColorChangeListener {
    public void randomizeColor();
}
```

Hint: you can build a random Color from 3 random bytes using Color.rgb(red, green, blue).

CONGRATULATIONS, your app should now be fully functional.

Part H: Examine a Google Sample

The example in the developer tutorials is more complicated, but shows the same ideas as this lab: creating Fragment classes, loading them dynamically in code using FragmentTransactions, and communicating via the enclosing Activity via the interface methods. However, it shows yet another idea: different layouts for phones and tablets. Please download the sample here:

http://developer.android.com/training/basics/fragments/index.html

You should run the code on both a phone and a tablet (use the emulator as needed). What a difference! Ask yourself (at least!) these questions. You don't need to submit them, but you are responsible for knowing them. Ask if you have questions.

1. Where are the different layouts specified?

2. The MainActivity needs to know which layout is being used so that it can either replace a fragment or just update one when a headline is selected. How does it determine which one is used?

3. What is the back stack? Where is it used in this sample?

4. How does it figure out which text to use for a given article?