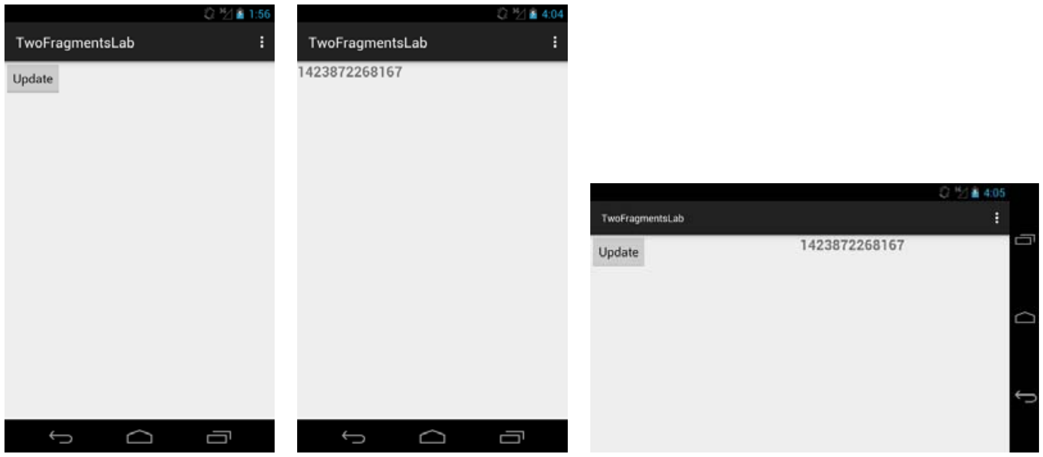
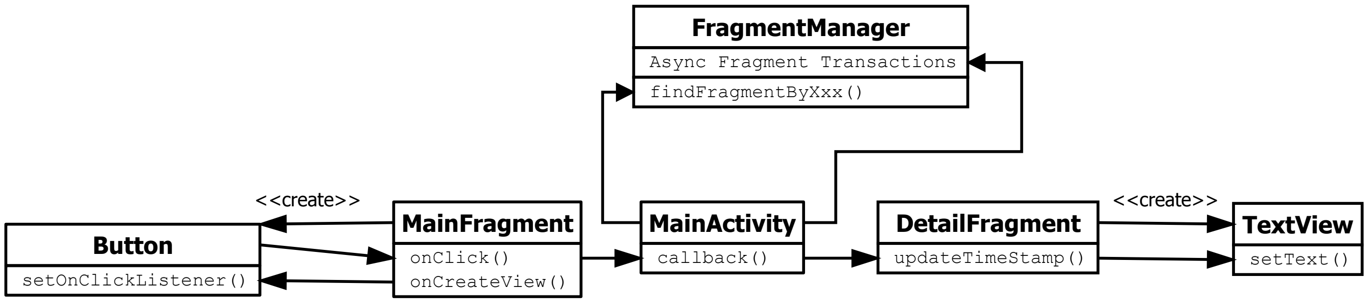
CSCD 372: Android Development
Lab 6: Fragments

There won't be any UI bling in this lab, just enough to illustrate both the static and dynamic use of Fragments. We'll begin with two static fragments, and will then modify the app to switch between fragments dynamically in portrait mode. At the time of this writing the API docs do not have an example for this scenario. They have something similar that uses two activities for the dynamic view, but we will be clever and use a single activity. Each fragment contains a single widget. The *Main* fragment will have a button that samples the system time and sends it to a text view in the *Details* fragment. Both fragments will be shown in landscape mode. For portrait mode, we'll pretend we don't have room for both. Pressing the Update button will bring the *Details* fragment to the foreground, and we'll arrange for the user to get back to the *Main* fragment using the back button. This is a bit superficial, but I'm trying to keep it simple and to the point, so imagine an app that has a long list of items in the *Main* fragment, and selecting one of those brings up a large list of details about that item in the *Details* fragment. This lab lays the framework for such things.



This is about as complicated as Fragments get, and I recommend you follow the development sequence highlighted here, or you'll miss some important lessons about fragments. The following UML diagram summarizes some of the important communication (method calls) that will occur in this lab. Note that the timestamp acquired by the button handler, in the main fragment, is communicated to the detail fragment via a callback through the activity. Note also that the activity should avoid keeping references to fragments, and should instead ask the fragment manager for references to fragments. Fragments that reside in a layout placeholder can be obtained using getFragmentById, others using getFragmentByTag. If the fragment manager returns a null, then the fragment does not exist and should be constructed before executing a fragment transaction such as add() or replace(). Finally, note that such fragment transactions are asynchronous.



Task 1: Create a project shell and two fragments

1. This write up should work for either ActionBar or Toolbar versions as long as you pay attention to the difference between activity_main.xml and content_main.xml. If you use the Blank/Basic Wizard, do not check the "Use a Fragment" box, because in this case it will actually save us more code-writing if we generate our main fragment AFTER the template is. You can remove the Hello World TextView, the pink button, and the associated code.
2. Right-click somewhere in your app directory structure and create a blank Fragment (New→Fragment→Blank). Call it something like MainFragment. Check the options to generate an XML layout and interface callbacks. You don't particularly need the factory method (newInstance) here, but it never hurts to have it, so I'll leave that up to you. If your generated Fragment imports android.support.v4.app.Fragment, then please note that whenever you need to get the Fragment manager, you should call getSupportFragmentManager(), not getFragmentManager(). If it is importing android.app.Fragment (without the support.v4 part), then make sure the generated onAttach() method receives an argument of type Activity (not Context). That will be marked as deprecated, but you won't otherwise get the onAttach() call, and there is no other reason to need support fragments here. There's a bug report on this with some rather lame explanations from Google, along with some flaming.
3. Remove the generated Hello TextView from the Fragment layout and put on a button labeled "Update". You don't absolutely have to, but in general you'll want to rename the generated interface and interface method to something more appropriate (in two spots each). It will be generated at the bottom of the class, and I also like to move that to the top so that it is in the face of any Activity planning to work with the fragment ("hey dude, implement this if you want to use me"). This is a good time to try out the Refactor→Rename option on the right-click menu after selecting some text (you might need to press Shft-F6 after that to get the rename dialog). Don't confuse this with a handler for the button. This is an interface that we are requiring in any Activity that uses the fragment so the Fragment can make a callback to it. Change the interface method so that it takes a String (instead of a Uri), and remove the TODO comment. I henceforth refer to this as the "callback method".
4. Generate another blank Fragment with a layout and call it DetailFragment. You won't need any interface callbacks for this one. You can leave the Hello TextView on this one's layout, but you should change the font size to something larger, and give it an id.
5. Change your main layout (which might be content_main.xml) to a horizontal linear layout, and place two <fragment> widgets side by side. You'll find that widget under the "Custom" list, and it will ask you to choose the class when you select it. This is why I had you create the class shells first. Put your MainFragment on the left and your DetailFragment on the right. Change the id for the left one to something like mainHolder, and the right one to something like detailHolder. For each, set the layout_width to 0 and the height to match_parent. Give each a layout_weight of 1. They should then split the available space.
6. Modify your main Activity so that it implements the interface declared in MainFragment. Just leave the handler method empty for now.

Task 2: Handle the button and relay the result to the Details fragment via the Activity

1. Hook up a handler for the button click using your favorite approach. Note that it is the View returned by the inflater that knows how to find your button object. Have your button handler call:
   ```
   String.valueOf(System.currentTimeMillis())
   ```
   And pass that string to the Main activity via the callback method.

2. Add a method to the Detail fragment that sets the value of the TextView to a passed String. The findViewById() method needs to be made on an object that derives from Context, and Fragment does not. You can call getView() here to obtain a context for the Fragment. That returns the root of the view hierarchy for your Fragment, *as long as it has already been inflated*, or in other words, *as long as onCreateView() has been executed and returned a root view.*
3. In your Activity, have the interface handler call the method you just added to the Detail fragment, shuttling the string it received from the Main fragment. In order to do so, it will need a reference to the Detail Fragment, which you can obtain by calling findFragmentById (on the Fragment Manager), passing the id of the holder for the detail fragment.
4. Do a test run. The button should update the display. Yay, but we're not even close to being done with this. While we want both fragments in landscape mode, we want to switch between the two in portrait mode, and that means changing portrait mode to using dynamic fragments.

Task 3: Switch portrait mode to dynamic fragments

1. Create a new layout file with the same name as your main layout, but specify layout-land as the directory. Copy the contents of your main layout file to this one, in text mode.
2. Back in your main layout (which will now be only for portrait orientation), delete the second <fragment> holder. For the first, replace the <fragment> tag with a <FrameLayout> tag. Remove the name (or class) property, but keep the width, height, weight, and id. Change the id to something like portHolder. Please note that resource IDs must be unique.
3. In your Activity onCreate(), instantiate a MainFragment if this is an initial run, as opposed to something like a rotation (Android will rebuild your fragments for you on rotations and other configuration changes). In the past we have distinguished an initial run from configuration changes by testing whether the incoming bundle is null.
4. We'll need a way to determine which orientation we are in. There is another way, but the following method is clever. Right-click on your res/values directory and select New→Values resource file. Give it a filename of config(.xml) an put it in the values directory. Then do it again, except specify values-land as the directory. In the first (portrait) version, enter this resource item:
   ```
   <item type="bool" name="dual_pane">false</item>
   ```
   In the other (landscape) version, do the same with a value of true. You can now determine whether you are in the two pane (landscape) orientation with this call (which I put in a helper method):
   ```
   getResources().getBoolean(R.bool.dual_pane)
   ```
5. Now we need to modify our callback from the MainFragment, because in portrait mode there is no visible detail fragment to forward the string to. If you're in the dual pane orientation, simple forward the time string as before. If you're not in dual pane mode, use findFragmentByTag() to ask the Fragment manager if it has a DetailFragment with the tag "DETAIL_FRAGMENT". We're not trying to locate the Fragment by Id, because we already know, by virtue of the fact that we are responding to the button press, that the single placeholder in portrait mode currently holds the main fragment. If it comes back non-null, then great, but it might be useful to generate a Log entry indicating that an existing detail fragment was actually found. If it comes back null, that means it doesn't exist, so instantiate one and generate a log entry indicating that a new detail fragment was created. No matter how you got ahold of a detail fragment, execute a replace transaction on the holder. Use the version of replace() that allows you to pass a tag for the fragment, and set that tag to "DETAIL_FRAGMENT." You might want to note that such tags are generally declared as finals. After the fragment has been replaced, make the call to set the text view.

6. Take a deep breath. Do a test run. Press the button. You should get a crash and an exception. Trace through the logcat window to see where it occurred. Can you figure out why this failed?

Task 4: Fix the crash on setting the text view

1. The call to "commit" a fragment transaction (you did include that, didn't you?), is asynchronous, which means it schedules the replace and returns immediately, rather than blocking until the new view hierarchy for the new fragment has been completely built. This means that the view hierarchy for the detail fragment hasn't been built when you made that call to set the text view, which means you tried to call a method on a null reference for the TextView. So what to do about that?
2. The usual (but not only) approach is to create an argument Bundle, put the string in it, call setArguments() on the new fragment, and then execute your replace transaction. Go ahead and do that. If you chose to generate the factory method, then note that the appropriate place for this to happen is in the newInstance() method for the fragment, and that there is some template code for that. If you did not generate that code, then feel free to build the argument bundle in your callback method.
3. Now check for the existence of that argument bundle in the detail fragment's onCreateView(). If it exists, grab the string and set the TextView. You have to do this after inflating the root view, of course. Note also that you haven't yet returned the inflated view, so you can't yet make use of getView(). Don't forget to get rid of that call to the set method from the callback in your Activity.
4. Make a test run. Pressing the button should replace the main fragment with the details fragment. Yeehaw! Press the home (not the back) button to navigate away from your app. Go to the recent apps screen and select your app again. It should come back with the last time stamp still shown. Note that you didn't have to do anything here, the view content is restored automatically. Thank google for small favors. Before you do too much celebrating, how do we get back to the main fragment? Let's try pressing the back button. Oops, that exited us. You shouldn't be surprised, as that is the usual function of the back button. We could add some fragment navigation to our app, but I'm not going to throw that at you here, as we still have plenty to do. A simple solution is to repurpose the back button to mean "go back within the app" as opposed to "go back within Android OS".

Task 5: Provide a way to get back to the main fragment

1. Before you commit the replace, add a call on the transaction to addToBackStack(null). For future debugging purposes, I should warn you that "popping an entry off the back stack" reverses the entire transaction, so beware of compound transactions (replace is actually a remove followed by an add, but that won't hurt us here).
2. OK, make a test run. The back button should get you back to the main fragment now. Great. Do this a couple of times and monitor the logcat screen, watching for the Logs you generate on the fragment replacement. I asked you to instrument that because while google's documentation of Fragment and FragmentManager discusses the tear down and rebuilding of the associated view hierarchy, it doesn't tell you when your fragment objects are and are not retained (and we know from lecture that some guarantees about that ARE made by FragmentPagerAdapter). So what can you conclude about FragmentManager and your existing code at this point?
3. You might want to take a short break with your favorite beverage, by way of celebration, before you try this. Exit your app. Rotate the device to landscape and then execute the app again. I'm guessing it crashes on you. If I'm right, move on to the next task with a heavy sigh. If I'm wrong, congratulations – you are a smart cookie indeed, and you get to skip over the next Task.

Task 6: Fix startup in landscape mode

1. You're here because it crashes when you try to start in up landscape. So what's wrong? Look through your logcat. You're unlikely to find a reference to a line of your code (marked in blue) for this problem, but you will find a good hint having to do with one of your Fragments (no View found). Take a look at what your activity's onCreate() method is doing in light of what I just asked you to test. Keep in mind also that we left our fragments static in landscape mode. Without thinking about it, we simply followed the usual pattern of creating a fragment and adding it to a container when savedInstanceState is null, indicating an initial run. But if we start in landscape mode we do NOT want to instantiate any fragments, because Android will instantiate static fragments automatically as part of inflating the layout. The lesson is to not rely on old habits or magic solutions without thinking about what's going on and how it applies to your situation...
2. So, savedInstanceState isn't telling you what you need to know here. You simply need to know whether you're in dual pane mode. Modify your code to instantiate and add a main fragment only if you are not. At this point your Activity's onCreate() logic should be something like this:

```
if we're in single pane mode ...
    ask the fragment manager if there is a fragment in the portholder
    if not, add a new main fragment
else (we're in dual pane) ...
    nothing needs to be done (not yet anyway, that's coming soon)
```

3. Test again starting from a landscape orientation. It should run fine. Press the update button to make sure the update is working in landscape. Rotate back to portrait. Press the update button. Rotate. Hmmm. It'd be nice if our last time stamp from the portrait mode were showing, especially since we rotated from that view...

Task 7: Get rotations into landscape and back to show the last time stamp (and one last problem)

1. It might occur to you to save the timestamp in DetailFragment#onSaveInstanceState() and restore it in onActivityCreated(). Because we're about to hit page 6, I'll save you some time by simply telling you that won't work, because the state being saved is associated with the fragment in the portrait container, and you need to update a fragment associated with the landscape container (they are not the same). What we need to do here is transfer information between two different fragments being managed by FragmentManager. The easiest solution in this case is to save the time stamp string to a member variable in your Activity's callback method, and save that to the outgoing Bundle in its onSaveInstanceState(). Don't forget to call the base class version in onSaveInstanceState() or you'll lose the automatic rebuild of the portrait view state.
2. Then, in the Activity's onCreate(), if you're in dual pane mode, test for a non-null Bundle and pass the string on to the detail fragment (which you can find by id). A clever idea would be to use the same tag as earlier and pass the savedInstanceState bundle to the DetailFragment as an argument bundle. Again, to save time I'll tell you that won't work either, because you won't be allowed to attach an argument bundle to a fragment that is attached to a static view (it'll tell you the fragment is already active). So here's what I did. I modified my set text method in the detail fragment so that it saves the incoming text to a member variable, and attempts to modify the actual view only if getView() is non-null, indicating that it already has an inflated view. Then, in its onCreateView() I check for a non-null value of the string member variable. If I find one then I set it in the inflated view. Now I can call the set text method from my Activity's onCreate() method, passing the latest time stamp. Make these changes (or take some other approach) and ensure that the preceding problem is solved.

3. Note that given these changes, we no longer need to create an argument bundle in the callback for portrait mode, nor do we need to check for an argument bundle in the detail fragment's onCreateView(). Feel free to back out of those strategies, but it's not required.
4. Test by rotating into landscape from the detail view and make sure the timestamp is retained. Now update that timestamp and rotate back to portrait, which should take you back to the detail view. Make sure the timestamp is retained here as well. If it is not, the solution is similar.
5. One more test: launch in portrait mode, update the time stamp, rotate to landscape, hit the back button. It should exit here, but probably won't. If it doesn't, try hitting the back button again. The problem is that we still have an entry in the back buffer. The solution is to override onBackPressed() and if you're in dual pane mode and the back stack entry count is greater than 0 (ask the fragment manager), then pop the back stack. Do that, test again, make sure you have an about box, and submit your apk file along with your java source for the Activity and both Fragments.

Additional Comments

1. You may be wondering if we could have also made the landscape fragments dynamic, and moved the detail fragment between containers from portrait to landscape. The answer is yes, and in fact I first wrote this lab assignment that way. However, it is significantly more complicated and really isn't worth the bother.

2. If you do ever find yourself wanting to move a fragment from one container to another (maybe because it has a rather large member variable footprint), be aware that it will have to be removed() from its existing container before the FragmentManager will allow you to add it to another. Those must be done in separate transactions, and you will need to make sure the remove transaction completes before doing an add or replace on another container. You can do that by forcing the FragmentManager to complete any pending transactions immediately, by calling: executePendingTransactions().