Project Number 2 – ColorFragments App

Overall: Create, run, and test an app named **ColorFragments**. The app is to have one activity – the **MainActivity**, and two fragments named **ChoiceFragment** and **ColorFragment**. These fragments are intended to be *dynamic* fragments – which are *not* listed in the XML layout of the Main Activity (activity_main.xml) – but rather are added into the layout by the Kotlin code in MainActivity.kt – *once the app is up and running*. At a high level, the idea is to provide a simple mechanism where a user can choose which form of **ColorFragment** s/he wants to see – *red or blue*. The choice fragment is always present for the user to see – and this fragment has two text view fields – one for a red fragment and one for a blue fragment. The user can then click on either of those text views – in which case the program then creates a Color fragment of the chosen color and displays that fragment on the main activity screen below the choice fragment. These color fragments are created on the fly as the user makes her/his selection – then are overlaid atop one another in such a way that use of the back button can revert the lower panel to the previously chosen fragment!

(**) UIDs and Keeping Count: The app is aware that it is making multiple copies of the **ColorFragment** – and it gives each instance of the **ColorFragment** that it creates a unique ID (UID). These UIDs are created as the app keeps count of each time it creates a ***new*** Color Fragment – and when it has just created a new instance, it then advances the count, and it gives the new instance the ID set to the current value of the count. So, your app needs to have two areas of data:

1. A global *static* variable **fragmentCount** – of type **Int** – that keeps count of the number of Color Fragments that have been created. Every time a new ColorFragment is created, the app increments this fragmentCount. Since you want this to be a Kotlin style static variable - this should be declared in a companion object in the ColorFragment.kt code. We emphasize: this variable fragmentCount keeps count of how many ColorFragment instances have been made – and this variable is common to all instances of the class **ColorFragment** (that's what static means!).
2. A global non-static variable/property **myIndex** (also Int) which is the index ID that will be given to the particular instance of **ColorFragment** just created. That is – each instance of ColorFragment has its own variable/property myIndex – which is its UID.

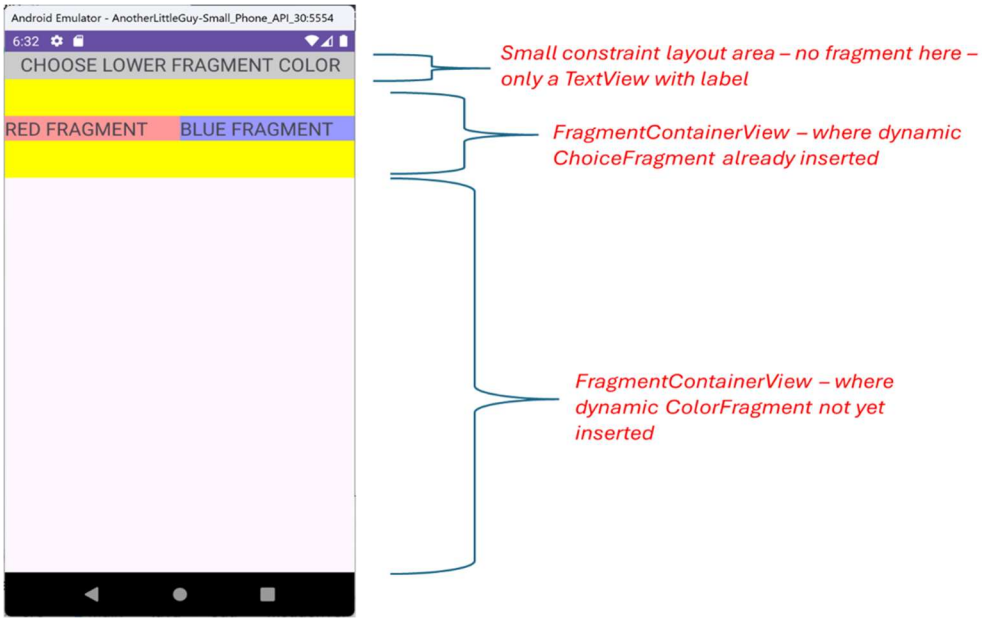
Each time a new **ColorFragment** instance is created, increment the variable fragmentCount, and then set the instance's **myIndex** value (its UID) to the new/current value of the variable **fragmentCount**. It is required that – each ColorFragment when displayed meets the following rules:

1. It is painted in its appropriate color (**Color.rgb(150, 150, 255)** for blue and **Color.rgb(255, 150, 150)** for red)
2. It says in text that it is a Red Fragment or a Blue Fragment
3. It displays its Unique ID (ID) in text on the screen

Warning: Since **onCreate()** can be called it times other than when the fragment is first created – be careful to increment the fragment counter only the *first time* onCreate() is called for a given fragment – and the bundle passed in is **null**.

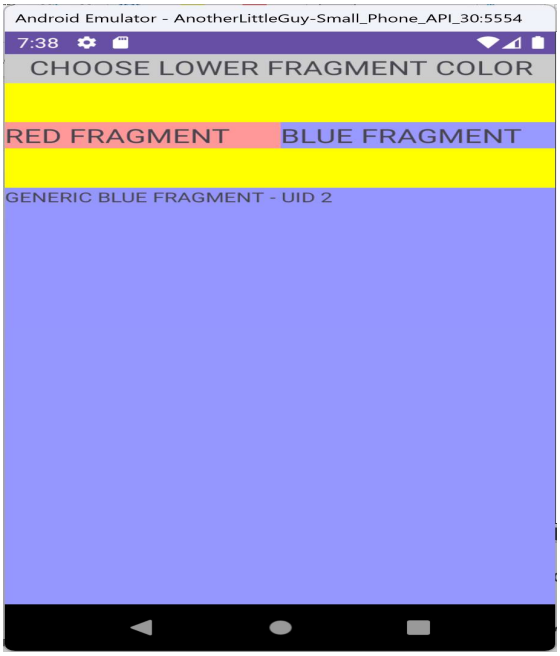
INITIAL APPEARANCE ==========

The app should have the appearance below when it first comes up:



NORMAL APPEARANCE WHEN ONE COLOR FRAGMENT COLOR HAS BEEN GIVEN ====

Note the textual material presented – in this example, this has ID of 2.



Overall Main Activity Screen Layout (for activity_main.xml) =====

Use a vertical linear layout with proportional sizing for height. The idea is that the overall screen has three sections: top, middle, bottom – where the top does not hold a fragment, and the lower two sections will need to each hold a dynamic fragment. The top section only needs room for the required label "CHOOSE LOWER FRAGMENT COLOR" in a TextView. You can proportionally size the height of the 3 sections with weights 1, 1, 4 – respectively – or you could force the top section of have fixed height – say 30dp – and let the proportional sizing only apply to the lower two sections – say, weights 1 and 4. But overall – the lowest fragment – *the displayed color fragment* – gets the most space.

The layout_main.xml should map out these three sections in the LinearLayout whose lower two sections are simply **<androidx.fragment.app.FragmentContainerView>** tags – into which the actual fragments will be inserted by Kotlin code later.

Top section: You could use a standard constraint layout with only one **TextView** within. That TextView can laterally span the whole screen and wrap around its content – which is the required label "CHOOSE LOWER FRAGMENT COLOR"

Middle Section: This is a slot where the dynamic fragment **ChoiceFragment** will be embedded by code in the MainActivity.kt file. Since this is a slot where a fragment will later be added in, use the XML element/tag: **androidx.fragment.app.FragmentContainerView** – exactly as we used in class for the **DynamicFragments** exercise. This XML FragmentContainerView should have ID set by:

android:id="@+id/upperfragment_container"

When the overall app first comes up, the **onCreate()** code in MainActivity will create an instance of the ChoiceFragment class – and insert that fragment into the **FragmentContainerView** slot. More on that below. But the FragmentContainerView tag should be given the ID:

android:id="@+id/upperfragment_container"

Width should span entire screen – use match_parent.

So the content of this middle section will be determined by the layout of the **ChoiceFragment** – which is discussed below.

Bottom/Lowest Section: This is a slot where the dynamic fragment **ColorFragment** will be embedded by code in MainActivity – but later in life. Since this is a slot where a fragment will later be added in, use the XML element/tag: **androidx.fragment.app.FragmentContainerView** – exactly as we used in class for the DynamicFragments exercise.

Each time the user selects the blue text or the red text – by clicking on it – the **ChoiceFragment** Kotlin code detects the click – and passes that knowledge – together with a flag for what color was chosen – on to the MainActivity. It is the latter that then creates a new ColorFragment of the selected color and proper labeling – and injects that created fragment into the lower/bottom section FragmentContainerView. This lowest section FragmentContainerView element should have the ID given by:

android:id="@+id/lowerfragment_container"

There is more below on the layout of the **ColorFragment.**

High Level Flow – Clicking on a Color View - Operation of the app ========

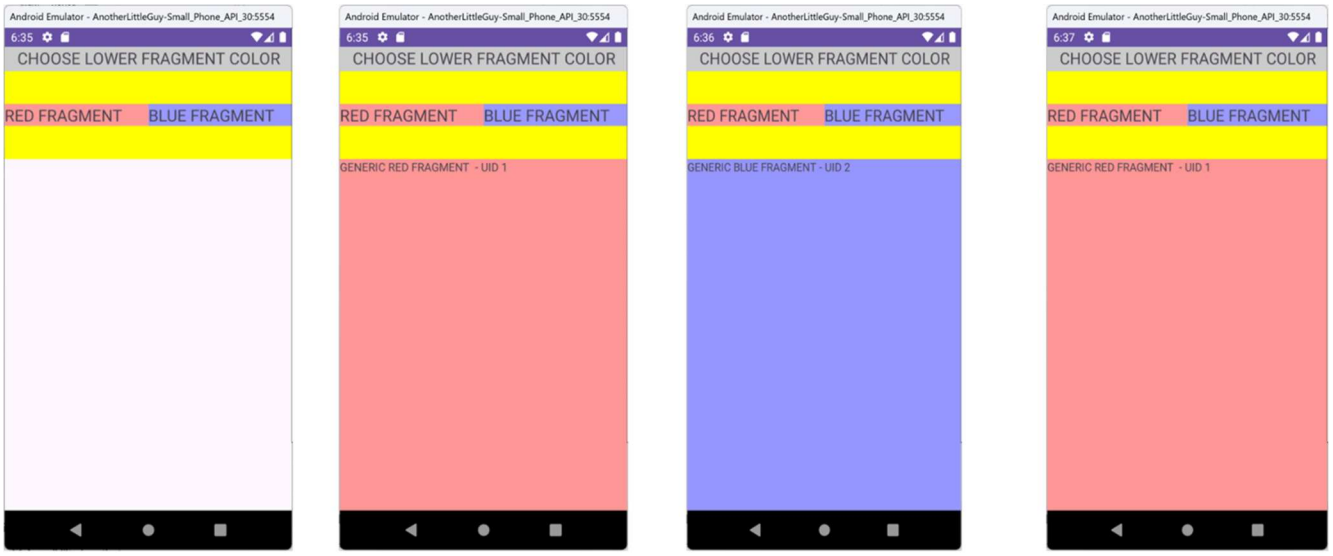
As stated above, the users clicks on one of the "RED FRAGMENT" or "BLUE FRAGMENT" views, and in response, the app generates and inserts into the lower section the appropriate ColorFragment. If the user clicked on RED FRAGMENT view, then the ColorFragment appears – colored red (255, 150, 150) with text label:

"GENERIC RED FRAGMENT – UID ##" – where ## is the correct UID.

If user click Blue, then the fragment would appear blue color (150, 150, 255) and the text:

"GENERIC BLUE FRAGMENT – UID ##" – where ## is the correct UID.

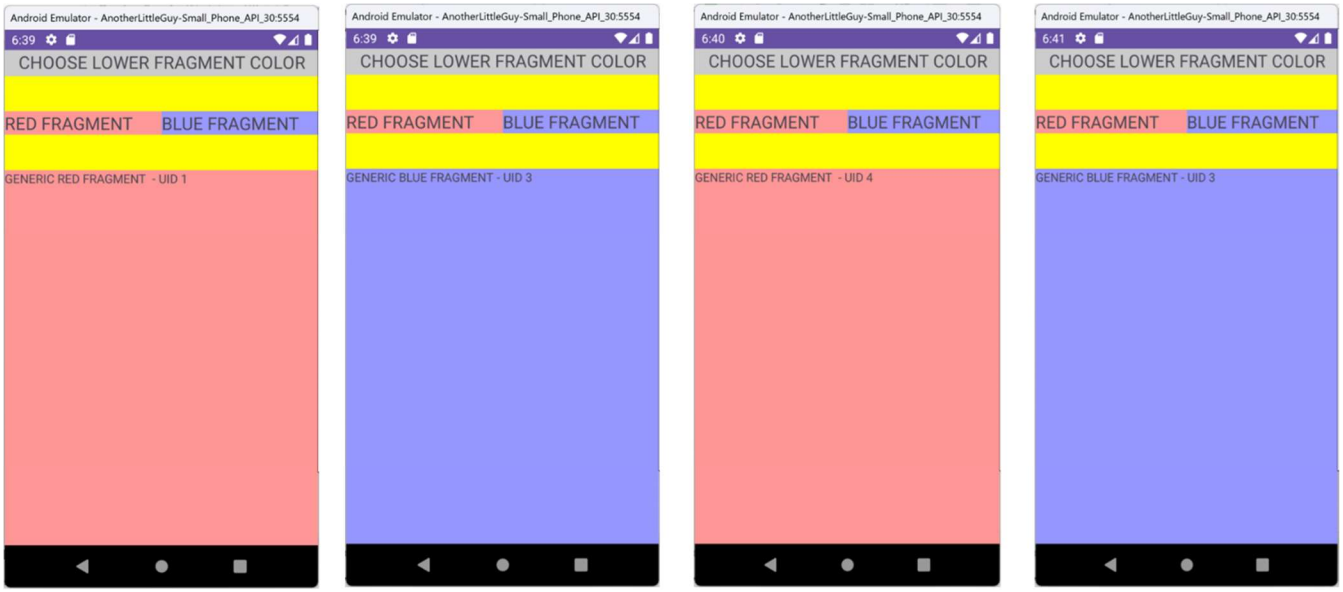
Each new ColorFragment should see a new UID and should overlay (in the stack) the predecessor fragment that was in the lower section. Hitting the back button should cause the lower section to revert to the fragment that was previously there – or emptiness, if there was no predecessor fragment. The pictures below illustrate one possible sequence of actions.



User clicks on RED FRAGMENT

User clicks on BLUE FRAGMENT

User clicks on Back Button



User clicks on BLUE FRAGMENT

User clicks on RED FRAGMENT

User clicks on Back Button

ACTION STARTS IN CHOICE FRAGMENT: Internally, the main components of the Kotlin code share the work of the overall app – with the ChoiceFragment first receiving the click on either the RED View or the BLUE View in it – and this fragment sending that choice to the MainActivity by calling its function onSelect() with one argument – 1 for BLUE and 2 for RED.

(***) THEN IN MAINACTIVITY: Then the MainActivity code is engaged, in its **onSelect()** function. This function should get an object **fcv** for the FragmentContainerView with ID: lowerfragment_container (findViewByID() call). It should also create a new Instance **colorFragment** of a ColorFragment – calling ColorFragment.newInstance() passing in the color choice (1 or 2). Finally, it should use the **supportFragmentManager** to create a transaction that replaces the current fragment in the FragmentContianerView with the newly created fragment **colorFragment** with code roughly:

supportFragmentManager.beginTransaction().replace(**fcv**.id, **colorFragment**).addToBackStack(null).commit()

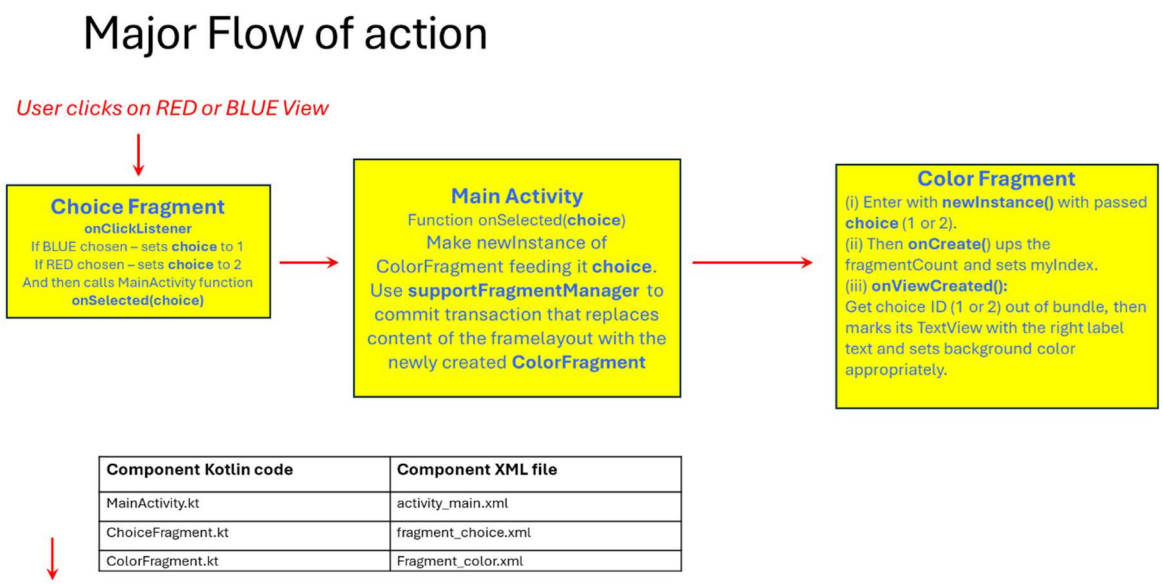
Note the use of the addToBackStack call – to assure that the pteviuos fragment is put on the stack!

THEN TO COLORFRAGMENT: Now the action moves to the ColorFragment Kotlin code, where we enter the newInstance() code packing the choice ID into the bundle. Put the choice ID as the value in a key-value pair whose key is "COLOR CHOICE"

Then onCreate() is entered – here, *if appropriate*, you increment the fragmentCount and set myIndex to the current fragmentCount. "If appropriate" means if this is the first time up for this fragment instance – which is detected by the passed in bundle being null.

Still in ColorFragment.kt, we now go to onViewCreated(). Here we need to extract the choice ID from the key-value pairs in the passed-in bundle savedInstanceState. We want to do **getInt()** to get the value that goes with the key "COLOR CHOICE" – getting either 1 for BLUE or 2 for RED. We then get an TextView object that does with XML TextView that goes in the ColorFragment – which has ID mycolor (use findViewById()). Now the code sets the background color and the written text in that TextView appropriately – remembering to write n the UID of that fragment also!

The picture below summarizes the flow:



More on ChoiceFragment ==========

XML: The XML layout for the choice fragment (upper one on the screen) should be minimal – but it does need some work. When you create the Fragment in Android Studio, AS will give you a FrameLayout. You could stick with that – and put inside it a horizontal linear layout inside of which to put the two TextViews for the RED FRAGMENT and the BLUE FRAGMENT, respectively. The LinearLayout should extedn across the full parent. These TextViews should be equally weighted laterally and wrap their content and colors in their appropriate respective color. These two XML TextViews should be given the IDs: **redText** and **blueText** respectively. Color the FrameLayout yellow – so that the two TextViews are clearly visible.

KOTLIN CODE: *The important functions for this fragment to implement are:* ***onAttach()*** *and* ***onViewCreated()***.

The Kotlin code for the Choice Fragment has to solve two issues. **First**, this fragment needs to respond to the click that the user makes of one of the RED or BLUE TextViews. One way to do this is to get – for example – an object reference **redT** for the RED TextView and then use **redT.setOnClickListener() {... }** to create the code to respond to the red TextView click. Treat the blue choice similarly. In this click listener code – in the above braces – you will know which TextView has been clicked – and can set the choice index to 1 if blue and 2 if red. Also – in this code, you need to alert the MainActivity of the choice – and so you need a reference to the main activity – and a function in that activity to call.

Second – so we need a reference to the Main Activity. Here we can use the onAttach() function that will be called for the ChoiceFragment when the Main Activity starts it. When onAttach() is entered, it is given one argument, a Content type variable that points back to the activity that started it. The ChoiceFragment code should save this value as a reference to the MainActivity and then later use it to call a function in the main activity to alert it that the choice has been made.

But the interface needs to be a bit more subtle – we want to do it just like we did in the DynamicFragments app in class. The MainActivity class needs to listen to choices made by the user and sent from the ChoiceFragment. So, MainActivity should have an interface defined called **ChoiceListener** with one function in it – called **onSelected(choice: Int)**. Thus - **MainActivity** implements this interface – and through this function, it hears about which choice was made in ChoiceFragment.

Back to the code in ChoiceFragment: this code can maintain a class property/variable called **choiceListener** – which it uses to refer to the Main Activity that started it. Then, when **onAttach()** is entered, and we receive the **context** variable that refers to the activity that started this fragment, we can check if it is a **ChoiceListener**, and if so, save that context in the variable **choiceListener**.

For **onViewCreated()** – this fragment should set up object variables **blueT** and **redT** to refer to the two TextViews that are clicked. For example – you could set **blueT** by:

```
var blueT = view.findViewById<TextView>(R.id.blueText)
```

Also in onViewCreated(), you should set click listeners for blueT and redT. For example, you might say:

```
blueT.setOnClickListener() {

        var choice : Int = 1

        choiceListener.onSelected(choice)

}
```

Do the same to get a listener for **redT.**

More about Main Activity ==========

For guidance on the XML needed in layout_main.xml – see sections in this document above describing Top Section, Middle Section and Lowest/bottom section.

For the Kotlin code needed: you need to create an interface ChoiceListener with something like:

```
interface ChoiceListener {

  fun onSelected(id: Int)

}
```

Also – you need to indicate that this class MainActivity implements **ChoiceListener** (in the Kotlin way!).

Otherwise, the Kotlin code for MainActivity needs to do two things:

1. Create and attach the **ChoiceFragment** – and
2. Implement the function **onSelected()** as needed for the ChoiceListener interface and to receive the choice selection from the ChoiceFragment.

For #1 – in the onCreate(), if this is the first time up (savedInstanceState == null), then create and attach the ChoiceFragment by doing three things:

- Use **findviewById()** to get an object reference to the FragmentContainerView with ID upperfragment_container – you could call this variable **cfv**.

  ```
  var cfv =
  findViewById<FragmentContainerView>(R.id.upperfragment_container)
  ```

- Make a new **ChoiceFragment** instance – save a reference to it is a variable **choiceFragment**

  ```
  val choiceFragment = ChoiceFragment()
  ```

- Use the **supportFragmentManager** to do a simple transaction to add this new Fragment to the cfv fragment container view:

```
supportFragmentManager.beginTransaction().add(cfv.id, choiceFragment).commit()
```

For #2, you need to implement onSelected(). This function takes one argument – the integer flagging the choice the user made (1 for blue, 2 for red).

```
override fun onSelected(choiceID: Int) { . . . }
```

Now – follow the guidance of section **(***)** above to implement this function.

More on ColorFragment ==========

The XML for this is very simple, with just a single TextView – with a text field where you can put the correct label and index number for the fragment – and setting the background color of that TextView to the blue or red color. Give this TextView the ID **mycolor**.

The Kotlin code for ColorFragment needs to have three aspects:

1. A static **newInstance()** function in its companion object to start it up with the integer parameter telling what choice was made
2. Code to maintain and manage variables
    a. **fragmentCount** – static Int to keep count of how many instances of ColorFragment have been made (define this in companion object)Code to
    b. **myIndex** – the value of the UID for this particular instance
3. Code in **onViewCreated()** to retrieve the **Int** telling the choice the user made and then set the text field and the background color of the TextView appropriately.

For #1 – for the **newInstance()** function needs one argument – an Int variable **choice** used to pass the choice made by the user – 1 for blue, 2 for red. Like we did for the **DynamicFragments** app, you could do:

```
fun newInstance(choice: Int) =
    ColorFragment().apply {
        arguments = Bundle().apply  {
            putInt("COLOR CHOICE", choice)
        }
    }
```

For #2, follow the guidance in the section **(**) UIDs and Keeping Count** – above.

For #3, get the value of the choice made into a variable choiceID.

```
val choiceId = arguments?.getInt("COLOR CHOICE", 0) ?: 0
```

Now – based on whether that value is 1 or 2, set the proper text field and background color. So, for the Red case, you might say:

```
chosenColorTV?.setText("GENERIC RED FRAGMENT  - UID $myIndex")
```

```
chosenColorTV?.setBackgroundColor(Color.rgb(255, 150, 150))
```

where **chosenColorTV** is the object reference for the **TextView** in the **ColorFragment** XML. It is a good idea to make this reference a global variable define inside and at the top of the **ColorFragment** class.

```
private val chosenColorTV: TextView?
  get() = view?.findViewById(R.id.mycolor)
```

GRADLE CONSIDERATIONS ==========

Follow the things we did for the **DynamicFragments** app in class- slide 42 of the Fragments slide deck.