Android: Fragments and Navigation

Submission. Recall the rules for flipped classrooms: do the work alone or with a partner that is unique for the semester. You can find the github link on Ed.

Only one student needs to submit this code, but both students may do so if they wish. This assignment should be submitted through github (sorry, partners can't share the repository). Fill in the README, identifying both students in the Name and eid field.

Video and Functionality Watch the video to see the app in action and get a basic tour of the code. The app displays four images, each with a label. When you click an image, it "takes over" the whole screen. Even the action/navigation bar and the system bars are gone. This is called "immersive mode" in Android UI parlance. When you click anywhere on the image that has taken over your screen you go back to the four image list.

There is a gear icon in the menu when viewing Four Image Fragment. That gear icon takes you to the setting page. The settings page has an immersive mode toggle switch and a spinner for showing the labels on the images. Hey, the gear icon has disappeared from the Toolbar when I'm in the settings fragment. I give you that code, so you don't even need to write it yourself.

If you set Show labels to false, then you don't see the labels in the Four Image Fragment view anymore. If you set immersive mode to false, then when you click on an image, you retain the Toolbar. You can press the up button on the toolbar, or you can use the phone's back button to return to the Four Image Fragment. Of course you can still click the image to get back also.

Fragments. Many apps have different "screens." Initially, these were different activities in Android. But activities have trouble sharing data and they have trouble adapting to different screen layouts. Fragments are parts of activities that can control the entire screen or part of the screen.

An activity takes over the whole screen. That keeps it conceptually simpler than fragments. We need some way to manage fragments, for example, we need a way to transition the entire screen from one to another. It is also possible to have more than one fragment controlling a single screen, but we do not do that in this exercise.

Navigation is a great way to transition among different fragments. Please check out this introduction.
`https://developer.android.com/guide/navigation/navigation-getting-started`

The RecyclerViewFrag demo showed three fragments and a bottom navigation controller that switched among them. The Navigation controller is really nice when we have a menu (the bottom icon list) and each menu item corresponds to a fragment. It is also really good for the drawer layout.

We will use navigation to transition from the FourImageFragment to the OneImageFragment. Read the XML in nav_graph.xml. See how there is an argument for OneImageFragment? You need to make an action to take you from FourImageFragment to OneImage-

Fragment and provide that argument. The argument is the index of the image we want to display.

You should read this about how a transition between fragments generates a Directions class so you can pass arguments in a type safe manner. You will do this in the code. `https://developer.android.com/guide/navigation/navigation-navigate#example-safeargs`

Layout. In activity_main.xml we have a standard toolbar layout. In `content_main.xml` we have a `fragment` object taking up the whole screen in a constraint layout. The `fragment` layout object will hold our two different different fragments.

The `four_image` layout is a simple vertical linear layout with image views and text views. It uses weights and the special `0dp` sized height for the image views. You need to provide the last ImageView definition just to make sure you are following along.

`one_image.xml` is dead simple. Note the black background. No changes needed.

`settings.xml` has the settings fragment layout, which is a mess. It has a toggle switch an a spinner. The spinner holds a binary choice, which is better represented with a toggle switch, but I wanted to provide you some custom spinner code. In that way this is more a demo than an FC. No changes needed.

`spinner.xml` has the text view for the custom spinner. No changes needed.

Menu. `menu/main_menu.xml` has the menu definition. There are no changes needed in this file, but you will need to know about the id of the menu item.

Navigation. `navigation/nav_graph.xml` defines the navigation graph, which consists of three activities. You need to define an action to transition from FourImageFragment to OneImageFragment. Your action needs an argument, which is an integer. The integer indicates which image is going to be displayed in OneImageFragment.

Note that the startDestination is FourImageFragment. That is why the app starts with the FourImageFragment view.

Starting in MainActivity. `onCreate` loads the jpg files, decodes them into bitmaps, and does the normal work of setting up the screen. We provide the code to initialize the navigation object. But please pay attention, you will see these calls again.

We also give you the code for immersive mode (`hideBars()` and `showBars()`. I had to look that up.

You have to fill in some code in `initMenu()` that responds to the user clicking on the gear icon. You need to get a hold of the navigation controller and navigate to the settings fragment. You can use `R.id.settingsFragment`, which is defined in your navigation graph.

Note that we initialize an array of bitmap during startup. We read jpeg files from storage and decompress them into bitmaps. This is a terrible thing to do on the main thread, and

it causes us to fail to render several frames during startup. But it is simple, and because it happens once at initialization, it does not produce any visual artifacts.

FourImageFragment. `onCreateView` has to do the standard view binding thing for a fragment. We saw this in RecyclerViewFrag.

`onViewCreated` is where the action starts. It should display the four images and set any onclick listeners (using the helper function `makeImageClickable`. You can access `mainActivity.bitMap` directly because it is not `private` and this file is in the same package.

The `makeImageClickable` routine sets an onclick listener that navigates to `OneImageFragment`. This is the fragment that takes over your screen. You can always call `findNavController()` within a fragment. You will need to figure out how to use a Directions object. Check this out `https://developer.android.com/guide/navigation/use-graph/safe-args#example-safeargs`

You also have to control the visibility of the text views. You should look up the difference between `View.INVISIBLE` and `View.GONE`.

OneImageFragment. `onCreateView` is your standard.

In `onViewCreated` you can see our code to call `hideBars` in `MainActivity`. Pretty cool how the activity is available to the fragments. `hideBars` gets us into immersive mode, if immersive mode is enabled. You can read about immersive mode here: `https://developer.android.com/develop/ui/views/layout/immersive`.

We set an onclick listener to bring us back to the 4 image view. How do we navigate back? How do we `finish()` a fragment? The best way is to call `findNavController().popBackStack()`, which is kind of weird.

Every time you navigate somewhere, the controller keeps a stack of where you have been. To kill the current fragment, we pop the stack. What is nice about `popBackStack()` is that you don't have to pass it any arguments. It knows it has to kill the current fragment.

Write the code to display the proper image. You will need the args object that is declared with this great bit of Kotlin code. Part code, part magic.

```
private val args:  OneImageFragmentArgs by navArgs()
```

SettingsFragment. `onCreateView` is your standard.

`initSpinner` is mostly provided as demo code. It combines things we have learned about like adapters with other tricks like accessing the companion object in MainActivity. You have to set `MainActivity.showText` based on what item the spinner selects. The `position` variable can be 0 or 1 given that the spinner has two options.

`initMenu` is for your enjoyment. See the gear disappear!

`onViewCreated` needs a few lines of code to deal with immersiveSwitch. `setOnCheckedChangeListener` is how you provide a lambda that gets informed about when switches are checked or unchecked. Look at the `isChecked` property.

Fragment manager. Before navigation we had the fragment manager. The fragment manager is a more confusing way to transition among fragments, but it is flexible. It is the only option if, for example, you want multiple fragments on the screen at once. We will mostly avoid the fragment manager though, because users find it confusing and navigation is the path forward.

Hints

- **Pattern match.** Our starter code contains idioms that are repeated in the code that you write. Read our code carefully and pattern match.
- The code that decompresses four jpg files on the main thread is terrible! We are doing this now for simplicity, but what production code should do is start a background thread, do the decoding of jpg to bitmap on the background thread, then notify the main (UI) thread when that job is done and display the bitmaps.

  As you can see, doing this job properly is complicated, so we will slowly fill in the pieces we need to do it as the course progresses. We need a coroutine and LiveData.