CSCI 448 – Lab 05A
Wednesday, February 15, 2023
LAB IS DUE BY **Tuesday, February 23, 2023 11:59 PM**!!

The App has been doing really since the last round of updates. However, the team has been noticing an increasing number of low star ratings and reviews due to the questions being too hard. Users are asking for a cheat mechanism that will give them the answer if they don't know the correct answer.

The team reluctantly agrees to put in such a feature, hoping to increase positive reviews and retain users. The decision is made to add a cheat button to the question screen that when pressed will pull up a new screen confirming that the user wishes to view the correct answer.

Step 0 – Add the Dependencies

Now that the app will be comprised of multiple screens and the user needs to navigate between these screens, the navigation dependency will need to be included. This will also require the Kotlin version be updated to a more recent version to avoid conflicts.

Part 0.I – build.gradle (Project: Quizler)

In the project gradle file, set the `compose_version` to be `1.3.3` and the `org.jetbrains.kotlin.android` version to be `1.8.0`.

Part 0.II – build.graded (Module :app)

In the module gradle file, set the `kotlinCompilerExtensionVersion` to be `1.4.1`. This will remove the `jdk7` and `jdk8` redeclaration/redefinition errors.

Then add the following dependencies:

```
implementation 'androidx.compose.material3:material3:1.1.0-alpha06'
implementation 'androidx.navigation:navigation-runtime-ktx:2.5.3'
implementation 'androidx.navigation:navigation-compose:2.6.0-alpha05'
implementation 'org.jetbrains.kotlin:kotlin-reflect:1.8.0'
```

Be sure to sync Gradle and it's set to go.

Step 1 – The Question Screen Specification

Before creating the new Cheat Screen, some reorganizing and new structure needs to be put in place. We currently have two packages nested inside of the `presentation` package: `question` and `viewmodel`. The `question` package contains all the components for the question screen's content. The `viewmodel` package contains all the components for the View Model.

Add a new package corresponding to `presentation.navigation.specs`. We'll use this to house all of the screen specifications – recall the technical requirement that all sealed interface instances must be declared in the same package.

Part 1.I – Create `IScreenSpec`

In our new package, create a `sealed interface` name `IScreenSpec`. Begin by adding the abstract declaration for our route string as well as a function declaration for our composable content. The composable will need the View Model and the `NavController`.

```
val route: String

@Composable
fun Content(quizlerViewModel: QuizlerViewModel,
            navController: NavController)
```

Part 1.II – Create `QuestionScreenSpec`

In the same package, create a concrete `object` named `QuestionScreenSpec` that implements the `IScreenSpec` interface. Override and set the `route` to be `"question"`. Then override the `Content` method to call our `QuestionScreen` function.

Part 1.III – Add `QuestionScreenSpec` To `IScreenSpec`

Return to `IScreenSpec` and add `companion object`. Add a member to get all the object instances for the sealed subclasses. Also set the root of the NavGraph to be a string set to "quizler" and specify the start destination as the question screen.

```
companion object {
  val allScreens = IScreenSpec::class.sealedSubclasses.map { it.objectInstance }
  val root = "quizler"
  val startDestination = QuestionScreenSpec.route
}
```

Part 1.IV – Create The `QuizlerNavHost`

We'll now create our custom `NavHost` for this application. In the `presentation.navigation` package (not in the `specs` package alongside the screen specifications), create a new file named `QuizerNavHost`. This will contain a composable function that accepts two parameters: (1) A `NavHostController` (2) A `QuizlerViewModel`.

```
@Composable
fun QuizlerNavHost(navController: NavHostController, quizlerViewModel: QuizlerViewModel)
{ ... }
```

For the body of our composable function, call through to `NavHost` providing the `NavHostController` object and setting the `startDestination` to be the `root` of `IScreenSpec`.

```
NavHost( navController = navController, startDestination = IScreenSpec.root )
{ ... }
```

We're now ready to specify the NavGraph. Add the `navigation` function call, setting the `route` to the `root` of `IScreenSpec` and the `startDestination` to the `startDestination` of `IScreenSpec`.

In order to create each destination of our NavGraph, we'll loop through all of our screens contained within IScreenSpec.

```
IScreenSpec.allScreens.forEach { screen ->
  if(screen != null) {
    // call composable()
  }
}
```

Note: that the resultant screen object may be null. It is not guaranteed when reflecting through all the sealed subclasses that an object instance will exist (look back to where `allScreens` get assigned in Part 1.III). In our case, they will exist because we are only creating object subclasses, but that is not an implementation requirement.

Now for the `composable()` function call to set up the destination. Set the `route` to be the `screen`'s `route` and then call through to the `screen`'s `Content()` method providing the necessary arguments.

Part 1.V – Render The `QuizlerNavHost`

In order to the full navigation package into use, we must use the NavHost. `MainActivity` controls the content of our app and it currently is calling `QuestionScreen` directly. We need to replace the direct call with our NavHost container – which will load it's start destination, which is `QuestionScreen`.

First, retrieve the `navController` object.

Then invoke the `QuizlerNavHost` composable.

Build, deploy, and run the app – and it should look exactly the same as it did before! But

Step 2 – The Cheat Screen Specification

The Cheat Screen will be a very simple layout. Begin by creating a new package `presentation.cheat`.

Part 2.I – Make `CheatScreen`

Create a new file in the cheat package called `CheatScreen`. The `CheatScreen` composable function will accept a `QuizlerViewModel` as a parameter. Create the following composable tree:

- `Column`
  - `Text - "Are you sure you want to cheat?"`
  - `Button`
    - `Text - "Cheat!"`

We will add more to this screen later.

Part 2.II – Make `CheatScreenSpec`

Now that there is a new screen created, we need to make the specification that corresponds to the screen. In the `navication.specs` package, create a second object that implements `IScreenSpec` called `CheatScreenSpec`.

Set the `route` to be `"cheat"` and the content calls through to `CheatScreen`.

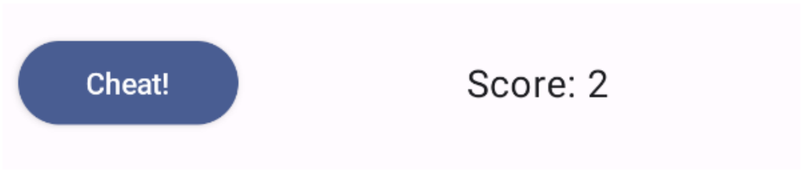
To test our new screen, temporarily change `IScreenSpec` to have it's `startDestination` be the `CheatScreenSpec` route. Build, deploy, run. You should now see the cheat screen! Navigation in action! Revert `IScreenSpec` to start on the question screen and let's now allow the user to navigate between screens.

Step 3 – Navigate Between Screens

Since our app starts on the question screen, we need to add a trigger that brings us from the question screen to the cheat screen.

Part 3.I – Add the Event Trigger

Add a cheat button at the top of the screen alongside the score. We will flow the `onClick` event up, so add another parameter to the `QuestionScreen` function. This will correspond to a function object that takes no input and returns no output. Assign this parameter to the `onClick` argument of the Button composable.



Part 3.II – Specify the Event Trigger

The screen specifications control the content of the screen (the `Content()` composable), how to navigate to the screen (the `route`), and how we transition between screens (the event handlers). When the QuestionScreen composable is set up, we can now provide the implementation for our `onCheatButtonClick` function object.

Specify a lambda expression that tells the `navController` to navigate to the `CheatScreen`.

The time building the navigation infrastructure will now pay off.

Step 4 – Deploy Your App

Build, deploy, run, and press the cheat button. Hello cheat screen! Press the back button. Hello question screen! Now that the screens are linked, you're ready to complete the ticket (and maybe a little something of your own doing as well).

LAB IS DUE BY **Tuesday, February 23, 2023 11:59 PM**!!