CSCI 448 – Lab 04A
Wednesday, February 08, 2023
LAB IS DUE BY **Thursday, February 23, 2023 11:59 PM**!!

Buried in your mockups of a clean landscape layout, you don't notice the CTO enter the dev floor. You hurriedly cover up your drawings and give your attention.

The CTO informs you the app has gained great traction and trending on the weekly download charts. However, we've begun to lose users that have tired of only answering True/False questions and the uninstall rate is increasing lately.

The Corporate Think Tank had the idea to add Multiple Choice questions that support up to four answers.

Feigning enthusiasm, you start to mentally think through how this will be done. You soon realize, there's not too much refactoring that needs to be done and much of the framework is already in place.

Step 0 – Add the Strings

The content of the new questions needs to be stored centrally in the `strings.xml` file. Create a custom multiple-choice question with four possible choices, or use the example below:

String Name	String Value
question6	Who won the Heisman Trophy in 1987?
q6_choice1	Barry Sanders
q6_choice2	Andre Ware
q6_choice3	Tim Brown
q6_choice4	Bo Jackson

Do use the string names for consistency, but the values can vary.

Step 1 – Update the Model

You open back up the LGGDG to review the Model structure. Our prior `Question` consisted of the text Id and a `Boolean` to store if the answer is true or not.

Given the following reworkings to the `Question` class, which would be best for our current situation and future extendibility?

A) Create an abstract Question class with a questionType value. Create concrete implementations that set the questionType value as appropriate and then add additional members as appropriate. In the View we'll work with the abstract type. We'll need to check the current questionType, cast to the appropriate concrete type and display the necessary elements.
B) Create an uber Question class that contains all potential fields a question could need. Have a questionType value. In the View, check the questionType value and access only the values that correspond to the associated type.
C) Refactor the existing Question class to hold four choices, the last two being optional. Change the Boolean to an integer.

Provide your answer in the Lab04A Refactoring Canvas Survey. The access code is **SOLID**.

If you choose Option C above, then that is the approach we'll take!

Part 1.I – Refactor `Question`

Change `isTrue` to be named `correctChoiceNumber` of type `Int`.

Then add two immutable members named `choice1Id` and `choice2Id`, these are both integers that correspond to string resource IDs. Default `choice1Id` to be the True string and `choice2Id` to be the False string.

Finally add two more immutable members named `choice3Id` and `choice4Id`, these are both nullable integers also corresponding to string resource IDs and both default to `null`.

Part 1.II – Update `QuestionRepo`

`QuestionRepo` will now need minimal patching. If the answer was marked as `true` previously, pass an argument value of `1` to correspond to the true choice being first. If the answer was marked as `false` previously, pass an argument value of `2` to correspond to the false choice being second. With our choice IDs already defaulting to a True/False style question, the repository is patched.

Add a sixth question to the list corresponding to the new multiple-choice question we are setting up. (If using the provided question text, then the correct answer is `q6_choice3`.)

Step 2 – Update the View

You've already grown accustomed to changing the View layer whenever the Model layer changes.

Part 2.I – Patch `QuestionDisplay`

There should already be some errors jumping out. Update the prior True Button to use the question's first choice ID. Update the False Button to use the question's second choice ID. That'll make the labels happy, but the `onButtonClick` needs to be handled in a better way.

Create a private top-level function in the `QuestionDisplay` package (i.e. outside of the `QuestionDisplay()` composable function) called `checkAnswerChoice`. It will accept four parameters:

1) The integer for the choice chosen
2) The integer for the correct choice
3) A function object of what to do on a correct answer
4) A function object of what to do on a wrong answer

The implementation will then check if the chosen choice equals the correct choice. If yes, invoke the `correctAnswer` function. Otherwise, invoke the `wrongAnswer` function.

We'll use this function as our event handled for the buttons. When the first-choice button is clicked, provide arguments of one (since this is the first button), the question's correct choice, and then our `onCorrectAnswer` & `onWrongAnswer` function objects.

The second-choice button will be very similar but provide an argument of two for the choice chosen.
The app should successfully build, and functionality should appear as it was. Except, when the user gets to our multiple-choice question there is no ability for them to select the correct answer!

Part 2.II – Expand `QuestionDisplay`

With a little UI logic, we'll now optionally display more buttons if needed. After the `Row` composable that is holding our first two `QuestionButton`s, add a conditional to check if `choice3Id` is not `null`.

If it's not, then we'll make a new `Row` to hold more buttons. Add another `QuestionButton` corresponding to `choice3Id` and use the appropriate `onButtonClick` lambda.

Additionally, check if `choice4Id` is not `null`. If it's not, then add another `QuestionButton` for `choice4Id`. With the proper nullability checking, we now have auto-support for a question with only three options as well!

Add a preview composable for a question that would have four choices.

Step 3 – Update the View Model

Open up QuestionViewModel and realize, there's nothing to be done here! Proper abtraction of layers. Thank you, Little Green Games Development Guide!

Step 4 – Deploy Your App

Build, deploy, run, and test the app. Send it up to QA to push out to production. And back to you landscape layouts you go...

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