

Project 5: Geography Quiz - Additional hints and explanations

Phase I

- Your program must be in the package `edu.uga.cs1302.quiz`.

Create a Maven project for Geography Quiz, update the `pom.xml` file, and remove the sample Apps. You should add a suitable dependency fragment to the `pom.xml` file in order to handle the Apache Commons CSV jar file automatically. Maven is very good in this regard. Here is the needed fragment:

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-csv</artifactId>
  <version>1.8</version>
</dependency>
```

Place it right after the dependency for `junit` in the `pom.xml` file.

- All classes described below should be placed in the correct directory in your Maven project.

Keep in mind that this document should be treated only as a guide and not as a requirement on how to create your project 5 solution. You can have fewer or more classes, with different names and objectives. Only the `GeorgaphyQuiz` class with the `main` method is required.

- Start by creating the `Country` and `QuestionCollection` classes. The first should represent a single country with its name and continent. The second should have all of the countries stored internally on an `ArrayList` of `Country`, as required. You should have the usual constructors, as well as setters/getters. The `QuestionCollection` class should read the countries from the CSV file and for each row, create a new `Country` class instance and add it to the list. This class should also have a method to return a country at a specific index position (0 through 195). Test it by printing all countries stored in the `QuestionCollection` class.

I created a simple example for reading from a CSV file. It is on `odin` in my `~kkochut/cs1302/csvreading` directory. It is a Maven project, so you may want to see how to include a dependency for the CSV reading library in the `pom.xml` file. There is a `README.txt` file which explains how to compile and then run this example. Please, read this file carefully.

- Create a class called `Question` to represent a single country question with 3 possible answers. This class will be a bit more involved, as you will need to randomly pick possible answers. Note that generating a quiz composed of 6 questions will be done in a different class, so randomly selecting countries will not be done in the `Question` class – just the possible continents as the possible answers.

I would suggest doing all the work in the constructor. The constructor could accept a `Country` object as its argument (to be randomly generated elsewhere) and then, based on the country, randomly generate the additional 2 continents. You may use either the `Math` class or the `Random` class, as stated in the project description. However, I would suggest the `ThreadLocalRandom` class, which is a preferred method of generating random numbers in Java 8 and newer. The call to generate a random integer value within a range 0 to `maximum` (exclusive, i.e., the highest generated value may be `maximum-1`):

```
int randomValue = ThreadLocalRandom.current().nextInt( maximum );
```

So, given an array `continents`, with the 7 continents (including Antarctica), you could generate a continent at random by executing:

```
int randomValue = ThreadLocalRandom.current().nextInt( continents.length );
```

Make sure you avoid duplicates! Simply, store the already generated (previously) value and if the next generated value is the same, generate again (until you get a different value).

Finally, create an array for storing the 3 continents (possible answers) and then randomly generate an index (0, 1, or 2) where you'd store the correct continent and store the correct answer there. Then, store the remaining 2 continents in the other 2 array cells (this should be easy enough).

Add a print method to output a question for easy testing.

- Create a class called `Quiz` to represent a single quiz with 6 questions. This class should create a new quiz in its constructor. You may want to use the same random number generating class as described above. Now, you will be drawing countries from your `QuestionCollection` class. For each randomly generated `Country`, create a `Question` by calling the constructor with the selected `Country` as argument and add it to the list of questions (`LinkedList` or `ArrayList`). Make sure to avoid duplicating countries in the same quiz, as described above.

You should have a method returning individual questions from the `Quiz`. Also, add methods to keep the total score for a quiz as the player is taking it.

Add a print method (to print the 6 questions) for easy testing.

- Create a `QuizResult` class to represent quiz scores and dates. Remember to use a list (`LinkedList` or `ArrayList`) and to state that it implements the `Serializable` interface so that objects can be written to disk.

MILESTONE

- At this point, you should be able to consistently generate quizzes with properly selected answers. Create a simple `main` method (temporarily) to generate a quiz and print it out to verify that all is working properly. You should also create a few quiz results and store them in the `QuizResult` class. Write an object of this class to disk and read it back the next time you run the `main` method to verify if writing/reading is working properly.
- In order to run your `main` method, follow the directions included in the `README.txt` file in the `csvreading` example on `odin` (in my directory `~kkochut/cs1302/csvreading`).

Phase II

- Design the GUI scenes using suitable controls, as described in the project assignment. The GUI programming should not be difficult.
- Start by creating the main JavaFX class called `GeorgaphyQuiz` (extending from `Application`). The main window should have the 4 required buttons, as described in the project description. You may start by implementing the handler for the `Quit` button and test it.
- Add the necessary instance variables to represent the quiz data objects, as created in Phase I.
- Implement a handler for the `start a quiz` button. It should open a new modal window. You should study carefully a multi-window JavaFX example in my directory on `odin`:

`/home/myid/kkochut/cs1302/NewWindow`

There are 2 examples, one without a delay and one with a delay of 3 seconds before closing a child window. However, you do not have to implement a delay in your quiz game (project 5 description discusses an alternative way to display the correct/incorrect result after the player answers a question).

In the new window, include suitable controls, at least a `Text` for the question text and a group of 3 `RadioButtons` for the answers. A `Submit` button should be there, as well.

Think how to show the correct/incorrect response after the player's answer and implement it.

Implement the `Submit` button handler to score the answer, update the total quiz score, and update the window with the new question and answer choices. In this handler, you should display the correct/incorrect response, with a delay or without. If you don't implement a delay, the correct/incorrect response info should appear on the next window update.

If you want to change a collection/layout of controls within an existing `scene` object, you can use the method `scene.setRoot(newLayoutRoot)` and provide a new layout, e.g., using a different `VBox`, `HBox` or `FlowPane` containing a different arrangement of controls. In this way, you could reuse the same `Scene` and `Stage` objects that you would create to show individual quiz questions (where the controls would include the `Text` or `Label` of the question, a group of 3 `RadioButtons`, and a `Submit` button) by a layout suitable to show the overall quiz result (with a `Text` showing the final quiz score and a `Close` button).

If the last question was answered, display the last answer correct/incorrect info, and update the final quiz score. It should be displayed to the player, of course.

Finally, you should add the result to `QuizResult` and write this class object to disk.

At this point, you may want to add the code to read this object from disk when the game program starts up.

- Implement a handler for the past quiz results button. It should open a new modal window. Again, look at the example on `odin` to see how to implement a scrollable content in JavaFX. The scene for this window should simply display the content of the restored `QuizResult` object.

Add the `Close` button and implement its handler, as illustrated in the example on `odin`.

- Implement a handler for the help button. It should open a new modal window. Again, look at the example on `odin` to see how to implement a scrollable content in JavaFX. The scene for this window should simply display some help information about your Geography Quiz game.

Add the `Close` button and implement its handler, as illustrated in the example on `odin`.