

## Programming Project 5: Geography Quiz

Due: August 3, 2021

### Project overview

In this project, you will create a simple quiz game program with a Graphical User Interface (GUI), called the Geography Quiz. The program will test the player's knowledge of countries and continents on which they are located (you might be familiar with a popular Sporele game called Countries of the World, which is somewhat similar).

The functioning of the quiz program is simple. It randomly selects 6 countries (from among the nearly 200 countries it knows about) and asks the player on which continent each of the selected countries is located. For example, the quiz question is clearly stated as "On which continent is Peru located?" (or "Name the continent on which Peru is located", or similarly) and includes 3 possible answers, including the correct continent (South America) and 2 additional incorrect ones (e.g., Africa and Asia). The 3 possible answers are shown in random order for each question. The user selects one of the listed countries as the answer and if the answer is correct, the user gets one point, otherwise, the user gets 0. The program records the answer and moves on to ask the next question. After the user answers the last question (with the sixth country), the quiz program displays the overall result (for example, 5 of 6 correct) and stores it for the record (just the overall result is stored along with the quiz date).

After completing a quiz, or right after starting the program, the user can request a list of past quiz results for review and the program displays them in date order, from the newest to the oldest. The program should allow the user to take several quizzes, one after another, if the user chooses to do so.

Internally, the program must maintain its data in lists. All countries (nearly 200) and their continents should be kept on a list. A quiz, containing the 6 randomly selected countries with their correct and 2 incorrect continents should also be represented on a list. Furthermore, the past quiz results and their dates should be represented on a list. In addition, the list of past quiz results should be stored on disk, so that past quiz results can be retrieved at a later time. After completing a quiz, its result and date should be added to the list and the list should be written to a file for storage.

Once you complete the project, submit it directory called `project5` to `csci-1302a` on `odin`, using the `submit` command. The `project5` directory must include:

- All the source code, i.e., `.java` files. JavaDoc comments are not required. Usual "program logic" comments are required in your source code, as well. Furthermore, your source code should be well formatted.
- A `README.txt` file explaining how to compile your program and how to run it.

### Project requirements

1. You must use Maven to organize your project. However, no tester program or Javadoc are needed.
2. You must use JavaFX to implement your project. **IMPORTANT:** You will have to switch from the default OpenJDK to Oracle's JDK on `odin`. To do this, you will have to add the directory `/usr/local/alt-java/bin` to your `PATH` definition in `~/.bash_profile`, ahead of other directories listed on the `PATH`. As an example, your `PATH` definition should look like this:

```
PATH=/usr/local/alt-java/bin:$PATH:$HOME/.local/bin:$HOME/bin
```

Also, add the following two lines to your `.bash_profile`:

```
JAVA_HOME=/usr/local/alt-java
export JAVA_HOME
```

Finally, make sure to test your `.bash_profile` changes by running `source .bash_profile` to test it and correct any errors before you log out from `odin`.

**IMPORTANT:** If you want to redirect the graphical window of your quiz program running on `odin` to your own computer, you must invoke the Java Virtual Machine (the `java` command) supplying `java` the option `-Dprism.order=sw`. For example:

```
$ java -Dprism.order=sw GeograhQuiz
```

This should be done for any JavaFX programs running on `odin`. Otherwise, you will see GLX errors and the program will not run at all.

3. All quiz program classes must be in the package `edu.uga.cs1302.quiz`.
4. The main method of your geography quiz program should be placed in the `GeograhQuiz` class. The first action should be to open the main window and display a brief paragraph about the purpose of the quiz and explain the basic rules (there must also offer a help screen). Also, at startup, the program should read the countries and continents data from a CSV file and read the prior quiz results from a file (if available – see below).
5. The CSV file with countries and their continents, `country_continent.csv`, is available on `odin`: `/home/myid/kkochut/cs1302/country_continent.csv`. All countries must be represented using a `java.util.ArrayList<E>` containing suitable class objects (create a class `Country`). You can use Microsoft Excel or LibreOffice Calc or even a text editor to view CSV files (they are text files). There are 195 countries and 6 continents in the file. The continents in the CSV file are Africa, Asia, Europe, North America, Oceania, and South America. There are no countries in Antarctica, which is a continent, as well. Note that in this CSV file, Oceania is a continent, while in some other publications it is a geographic region. You *must not* use an array to represent any quiz data.  
  
You must learn how to work with Comma-Separated Values (CSV) files in Java. You must use the Apache Commons CSV library to read the CSV file. The current version is 1.8 and the needed jar file, `commons-csv-1.8.jar`, is available on `odin` in my directory `~kkochut/classes`. You will need to learn how to use this library to read the CSV file with countries (it is quite easy).
6. After startup, in addition to a brief explanation of the purpose of the game, the main screen should have buttons to:
  - start a new quiz in a new window,
  - view the results of the past quizzes in a new window,
  - display help screen with information about using the program, and
  - exit the program.
7. When the user wants to start a new quiz, view the past results, or display the help information, the program should open a new window to present the requested information. The window must be a modal window, with `APPLICATION_MODAL` modality, which will prevent any events on other windows in the program. That is, the new window must be closed before the user is able to return the main window to invoke other functions, as described in point 5 above.

8. The other functions of the program must not be available, once a quiz, viewing of past results, or displaying help information has been started. This should be achieved by opening a modal window.
9. You should use `java.util.Random` and its `nextInt(int bound)` method with `bound` equal to the number of countries. Having all countries represented in a `java.util.ArrayList<E>` will allow you to quickly obtain a country at a randomly selected index position. For each selected country, the program should randomly select 2 additional continents (you may use Antarctica among the additional continents even though it has no countries and is not included in the CSV file). The program should store the selected 6 countries and their associated continents on a list (using either Java's `ArrayList` or `LinkedList`) and store the current date as the quiz's date (use `java.util.Date`). Be careful about randomly selecting the countries and continents, as the chosen countries and continents should not have duplicates. The player will be quizzed on the selected countries. Again, you *must not* use an array to represent any quiz data.
10. After creating a new quiz and opening a new window for it, the program should quiz the player, asking about the continent of each of the 6 countries. For each country, the question should include the three possible continent choices, including the correct one and the two additional ones. Each quiz question should be clearly stated (e.g., “*On which continent is XXX located?*” “*Name the continent on which XXX is located*”, or similarly), and the possible choices should be shown below, in the same window. Implement the continent choices as radio buttons, with suitable labels. The user should select one of them as the answer. The window should have a submit button (e.g., “Submit”, or “My answer”, or similar) to indicate that the user has made a choice and is ready to move to the next question.
11. Once the player clicks on the submit button, and the program should accept the player's answer and score it (the program should keep the overall score of the current quiz). Also, the program should report the answer as correct, or incorrect, in the same window. In case the answer was incorrect, the correct continent should be shown (it is a learning quiz!). For example:

Incorrect! Correct answer is South America

After a delay of, say 3 seconds, the program should then move on to the next question. The next questions should be displayed in the same way, in the same window.

As an alternative, the correct/incorrect answer assessment can be shown within the next question content. If this is implemented, a delay is not needed, since the player will have enough time to consider a correct answer displayed by the quiz program.

12. Once the player answers the last (sixth) question, the program should display the final result of the quiz. This should be after the answer to the final question. A delay after the last question should also be implemented for the user to review the result of the last question, and after the delay, the result should be displayed, also in the same window. For example:

Your score is: 6 out of 6

At this point, a Close button should be displayed in the window to allow the user to close it and return to the main window.

As an alternative, the correct/incorrect answer assessment from the last question can be shown within the content of the window with the overall quiz score. If this is implemented, a delay is not needed, since the player will have enough time to consider a correct answer displayed by the quiz program.

13. At the end of a quiz, the program should store the result of the quiz and the date on the list of past quizzes (using `ArrayList` or `LinkedList`). The list should be written to a file so that it could be read the next time the program is started (the past quiz results should be restored from file). You must use a relative path name. Using an absolute path for the file will likely prevent others from using your program!
14. When the user wants to view the past quiz results, the program should open a new modal window and display the past quizzes in date order, from the newest to the oldest. For each result, the quiz score and date should be printed, one per line. Please note that you do not have to sort the past quiz results by date. If you add a quiz result to the front of the list, they will be automatically sorted.  
  
You can format the dates & times using the `DateTimeStringConverter` class from JavaFX. Alternatively, you can also use `SimpleDateFormat` to format a `Date` object.
15. The past quiz results window must have a Close button and the user should be able to dismiss the window using it to return to the main window.
16. When the user wants to view the help information, the program should open a new modal window and display quiz instructions in sufficient detail. You should provide a reasonable explanation for how to take your geography quiz. This window must have a Close button and the user should be able to dismiss the window using it to return to the main window
17. When the user wants to quit the program (only possible from the main window), the program should terminate.
18. You must implement saving and restoring of past quiz results. More specifically, the program should store each quiz result and its date. You should not implement saving of partially completed quizzes, only the finished ones. Simply, any partially finished quizzes should be discarded. Take a close look at the Program of Study example in your textbooks (in chapter 15). The file with quiz results should be called `quizzes.dat`. and be in the current working directory (project5 directory if the program is started there).

### Project Implementation Hints

1. You will have quite bit of freedom in how you implement this quiz program.
2. Implement the `GeographyQuiz` class, as it is required. Also, you may want to consider creating the `QuestionCollection`, `Country`, `Question`, `Quiz` and `QuizResult` classes, but it is up to you how you will handle the representation of quiz data. `GeographyQuiz` must contain the main method for your project and create and open the main GUI window. `QuestionCollection` should have a list of all countries and their continents, read from the CSV file. As already mentioned, your program must rely on the Apache Commons CSV library to read the CSV file. The `Country` class should represent a country and its continent. Remember to use `java.util.ArrayList<E>` to store all countries. The `Question` class should have a country and 3 continents as possible answers. The `Quiz` class should have a list of Questions but only for the current quiz. Remember to use `ArrayList` or `LinkedList` to store the questions.
3. You may want to create a class `QuizResult` to represent quiz results. Remember that quiz results should be stored using `ArrayList` or `LinkedList`. Remember that all of your classes that you will want to write to a file should implement the `Serializable` interface.

4. Again, you must implement saving and restoring of past quiz results (`QuizResult`). More specifically, the program should store each quiz result and its date. You do not have to implement saving of partially completed quizzes, only the finished ones. Simply, any partially finished quizzes should be discarded.

**Things to note:**

- All quiz classes must be in the `edu.uga.cs1302.quiz` package.
- You must use the `java.util.ArrayList<E>` class to store all countries read from the CSV file.
- Your program must read the countries and continents from the CSV file in my directory on odin. You must use an absolute file name. You must use Apache Commons CSV library.
- You must use `java.util.LinkedList<E>` or `java.util.ArrayList<E>` class to store the randomly selected questions in a quiz and also to represent past quiz results. You *must not* use arrays to represent any quiz data.
- You must store the past quiz results on disk in *binary format*. The file must be called `quizzes.dat` and be located in the current working directory (usually, `project5`).
- Your design should be reasonably efficient, and in accordance with object-oriented design principles (encapsulation, information hiding, inheritance, etc.).