|  |  |
| --- | --- |
| Phase 3 Source – MODULAR STUDY APPLICATION | ABSTRACT  A Project Design for the development of a Modular Study Application for students. The design outlines the interface and functionality, application structure, build instructions and UML Diagram for the application.  Kory Bennett / Brock Allton / Dylan Sawyer / Justin Casey / Dan Page / Cale Ward  UMUC 495 6380 1 March 2020 |

TABLE OF CONTENTS

|  |  |
| --- | --- |
| **Revision History** | **2** |
| 1. **Milestones** | **3** |
| 1. **Problems Encountered** | **4** |
| 1. **Re-Evaluation of Decisions** | **4** |
| 1. **Key Functionality** | **5** |
| 1. **Application Structure** | **6** |
| 1. **Build Instructions** | **7** |
| 1. **UML Diagram** | **8** |
| 1. **Progress** | **9** |

Revision History

|  |  |  |
| --- | --- | --- |
| Date | Description | Author |
| 24 Feb 2020 | Initial Draft | K. Bennett |
| 25 Feb 2020 | Updated milestones/problems encountered & reviewed | B. Allton |
| 01 Mar 2020 | Reviewed updates for P3 | C. Ward |
| 01 Mar 2020 | Provided detailed documentation for several sections related to coding progress including screenshots | D. Sawyer |
| 01 Mar 2020 | Reviewed and edited | B. Allton |
| 01 Mar 2020 | Phase III Doc review | J. Casey |
| 01 Mar 2020 | Reviewed and edited several sections | D. Page |
| 01 Mar2020 | Final Doc Review / Format and edit | K. Bennett |
|  |  |  |

Phase 3 Source

1. **Milestones**

For the Phase 1 Design week the milestones were to have the GUI of the program function developed to the point of being able to launch the program and select the subject to test on, choose the number of questions to be tested on, and progress through the question database with a score result displayed once all questions have been answered and submitted.

This team has been performing on task and on schedule since the first week of class. The coding of the project is progressing at an acceptable rate. A few re-evaluations have been noted with exceptional turnaround each time. The overall team dynamic is good with good communications throughout the week vie Hangouts chat, and a standing video teleconference (VTC) every Monday and Saturday. During the VTC, goals for the week are discussed with expectations set for the end of the week with any last-minute changes being discussed on Saturdays.

For the phase 2 design week, the milestones were to have the GUI and the database setup and begin the process of having the GUI and database begin communicating with one another. Test questions should start coming up and answers should start being recorded for display at the end of the testing process.

The team continues to communicate very well with one another throughout the week. Many messages are exchanged regarding the programming of the code, and how best to work through any situations that may arise.

For the phase 3 design, the milestones were to make additions to the number of questions per subject, run the test cases making sure the program functions correctly, and do any debugging on any problems that may have arisen in the program. Some problems have been successfully taken care of, while others persist. Milestones were not fully attained during phase 3 and will continue to be adjusted before submission of the final product.

As in the past, communication between team members has been exceptional. Ideas are passed back and forth with ease and given adequate attention if they can be utilized or not. Contact is still maintained primarily through text messaging via Hangouts.

1. **Problems Encountered**

An issue that was discovered is that Java is not able to reuse and create new objects on the fly as Python is able to do. The results need to be parsed into a loop utilizing a tricky index integer that would recreate a question object every 4 loops which quickly became muddled.

Having a group of people working on a repository of code proved somewhat of a challenge. With each person having an established area to work, tidying up was somewhat difficult as there were always different versions available through GitHub. Ensuring the most current version of code available required coordination of all members. Even with having group members on opposing shifts we made sure to adjust as needed to involve all members with the process.

The major issue that was encountered for the phase 3 portion was when the program was executed more than one time. The first time, the code would execute perfectly with questions displayed correctly and answer highlights working as intended; the score is posted as a correct percentage. The second attempt yields the answer highlights stop working. The third test run brings up incorrect scores as well as highlights still not working and repeated popups. Several attempts were made to correct the issues, which were likely the results of multiple instances of class objects or un-reset variables.

1. **Re-evaluation of Decisions**

To mitigate the situation with Java not being able to reuse and create new objects like Python can, the database was changed to the format of String question\_text; String option 1, option 2, option 3, and option 4; and int correctOption.

A Choice class was created to help with our Question class. This decision was made to make keeping track of scores easier.

An initial thought process in the development of this program was to use the GUI to provide the “heavy lifting” of communicating between the user and the database. This was quickly realized to put a lot of undue stress on the GUI program which could result in some errors and problems very quickly. The decision was made to incorporate a Main class to allow communication between the GUI and the database.

The GUI this week was working and integrated with the database up to the point of questions and choices being displayed to the User. After collaboration throughout the week on keeping three separate classes that hold each JFrame for the GUI, a decision was made to hold all the GUI inside of one class called Main. The GUI and database integration is now fully working inside of Main with the score being tracked for the User-based answers provided to all the questions throughout the quiz. This decision to use the Main class led to three of the classes that held the original GUI to be deleted.

This decision made tracking variables and handling code flow much easier, however, the ability to reset all instances and static variables became an issue. The program runs flawlessly the first time through, but upon reset will start to have strange issues with certain methods not being called and variables not having proper values (see problems encountered section for more details). During Phase 3, the exact cause of these issues was unable to be located but several sections of code were optimized so that updates in real-time were possible from the database itself rather than hard-coded. The reason this problem may not be fixed is possibly due to the code structure itself being unable to reset all static variables or instances, the restart button may have to simply be removed in favor of a simpler procedure of closing and restarting the whole program itself allowing for the program to function as intended.

Phase 3 Source

1. **Key Functionality**

Team Modular Study created a Graphic User Interface (GUI) software that allows a user to access a database with questions and be tested with multiple-choice options to answer. A score is provided to tell the user the success of the test. The application works by allowing the user to select a course topic and importing those questions from a database, displaying them for the user to be tested on in random order. Functionality includes:

* A drop-down pane that holds the subjects that a User can be tested on
* An area for the user to input how many questions they want to be tested on
* Multiple choice questions in random order with four options the User can select from
* The results of the test showed
* An option for the User at the end of the test to exit or restart the application

1. **Application Structure**

**Main.java**

* Started by the user
* Initializes and handles GUI.java
* Queries Maria DB for the above information through Database.java
* Parses database results for optimal viewing in GUI
* Receives requests from GUI.java
* Sends results to GUI.java
* Serves as the primary interface to the database
* Responsible for randomizing questions
* Tracks questions completed out of total
* Tracks number of correct answers out of total

**Database.java**

* Responsible for maintaining the connection to SQL database
* Receives queries from Main.java
* Sends information to Main.java
  + Available Databases
  + Number of Questions within each database
  + Individual Questions

**Question.java**

* Helper class for storing question information
* Created per question, generated from Database.java
* Resulting questions are sent to Main.java

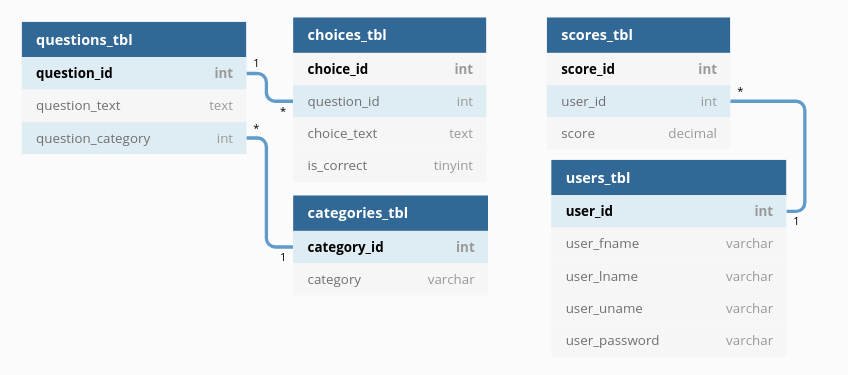
**Choice.java**

* Helper class for storing answers to questions
* Created per choice, generated from Database.java
* Resulting choices are attached to corresponding questions
* Choices are compared by Main.java to determine if correct

**SQL**

* Stores all subject and question information
* Only interfaced by Database.java
* Hosted remotely to mitigate database setup complications

Entity Relationship Diagram



1. **Build Instructions**

**cd into the ‘Code’ directory**

* 1. **Compile your Java code, generating class files**

*javac -cp “.:./mysql-connector-java-8.0.19.jar” <ALL\_JAVA\_FILES>*

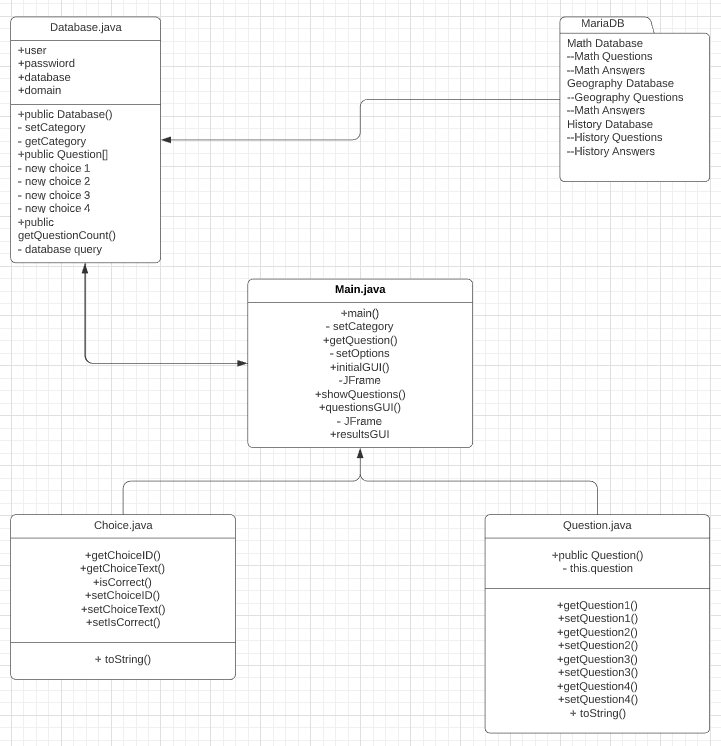
* 1. **Create a JAR file called study**

*jar cmf manifest study <ALL\_CLASS\_FILES>*

* 1. **Execute the JAR file**

*java -jar study*

1. **UML Diagram**

****

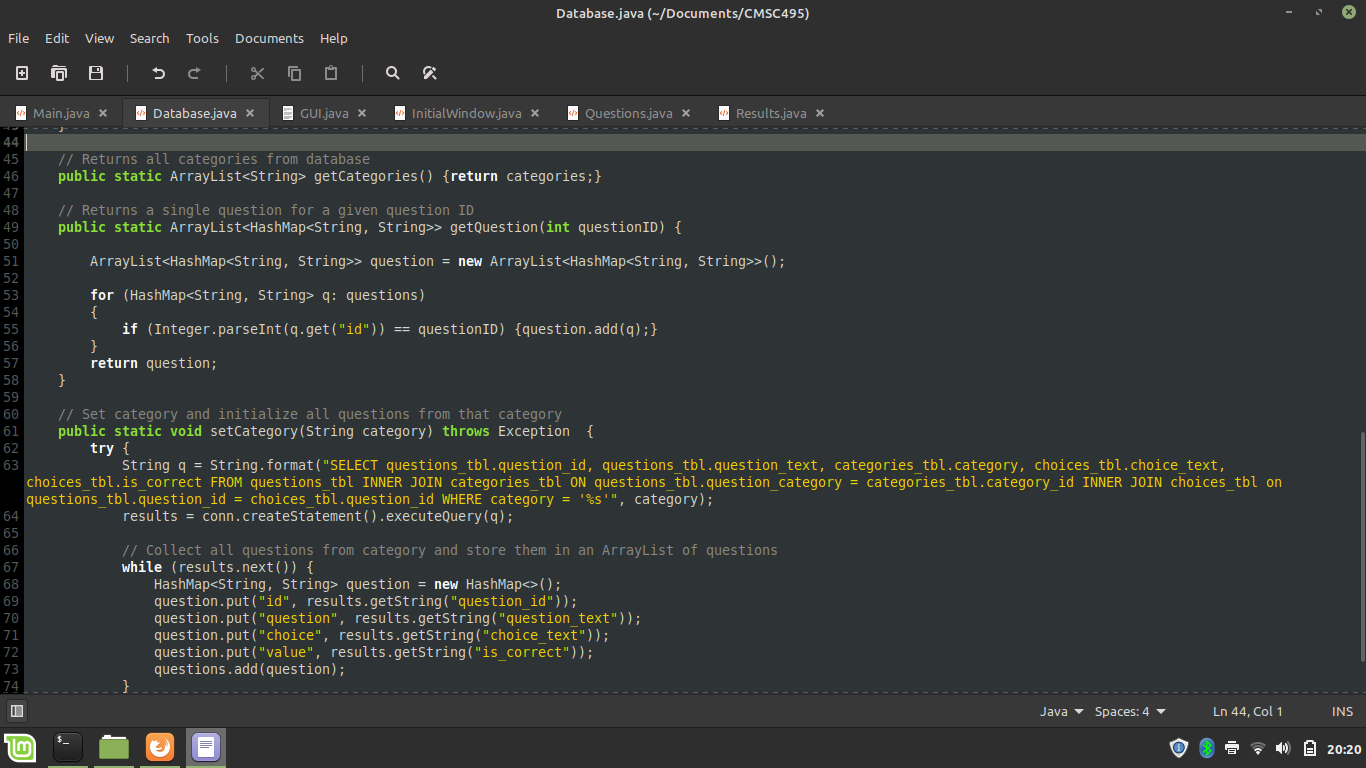
1. **Progress**

Some decisions that were made early in the project were to develop a username/password login, however, this process was quickly deemed to be unnecessary and removed from the program. Another decision was to add the Database.java to remove some of the logical load on the Main and GUI components. The database class will be responsible for connecting and querying the SQL database directly, where the Main and GUI components simply request subjects and questions from Database.java in a user-friendly format. Screenshots of operable code below for Database communication: (Screen Shots below)

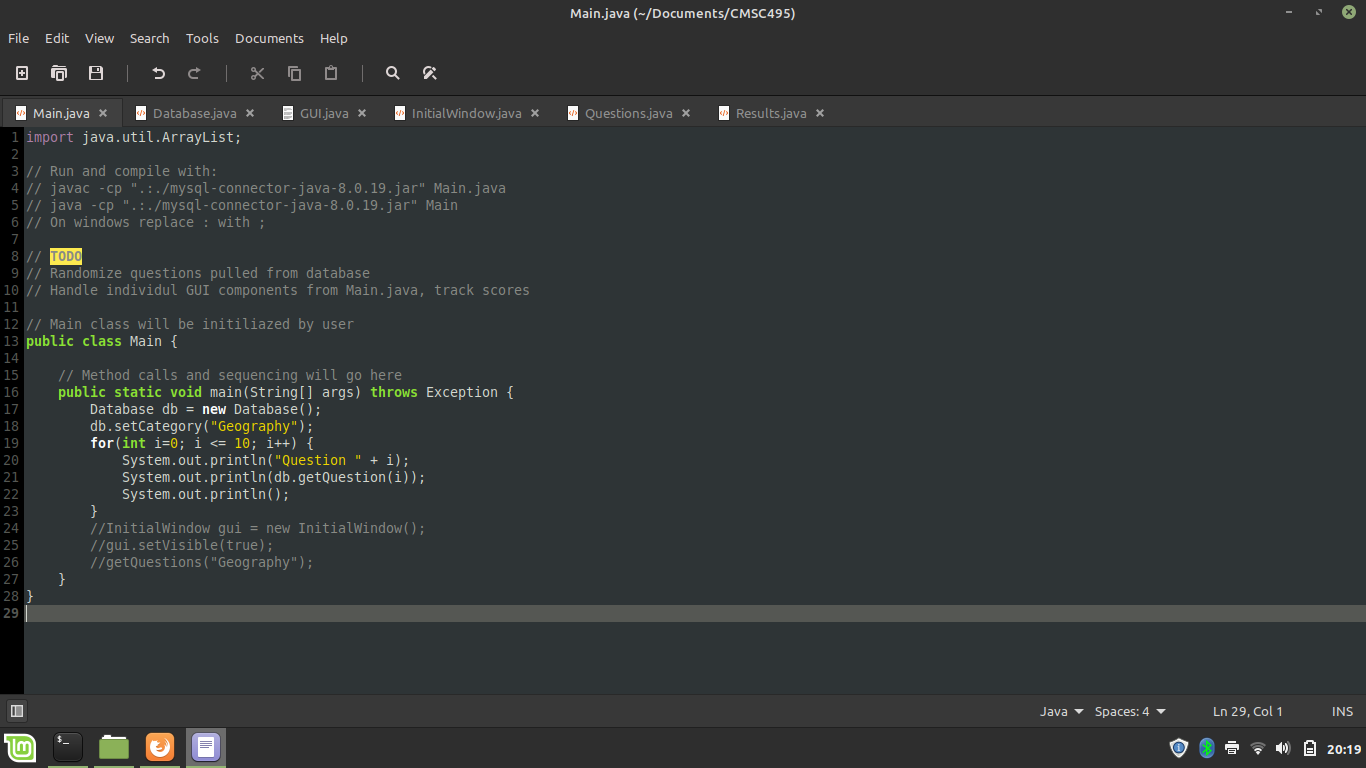
For Phase II, the GUI components were combined into Main.java. This was done to avoid complications with passing variable values back to Main.java from the 3 separate GUI components. For example, passing questions to the GUI, then receiving a response from the GUI without interrupting the user’s quiz or tracking variables inside the GUI class itself. An alternative was to setup Remote Methods which would be a bit complicated. To compensate, the program now has a separate Question.java class for each individual question, as well as Choice.java for each individual choice. Additionally, the Database.java remains intact. At this point several bugs have been noticed in terms of logic, however, all components are currently operational and communicating with one another. (Screenshots below)

For Phase III, the GUI was connected to the database in real-time for updating GUI components such as question numbers and drop-down list. A few minor changes to speed up logic were added, as well as highlighting answers during the quiz itself. The quiz runs flawlessly the first time through, with an optional restart button at the end to select another test. This feature was not thoroughly tested in the previous phase so we’re unable to determine the time of origin. Restarting the quiz seems to not reset all static variables, even though it is coded to do so, or may be creating multiple instances of GUI logic. The exact cause is still unknown and a few attempts have been made to narrow down where the error is coming from. The resulting errors are detailed in the screenshots below. If problems persist, and we cannot fully reset all variables and methods while inside the Main class itself, the simplest solution would then be to remove the restart button, allowing the user to simply exit and restart the program from scratch, which should solve all related bugs.

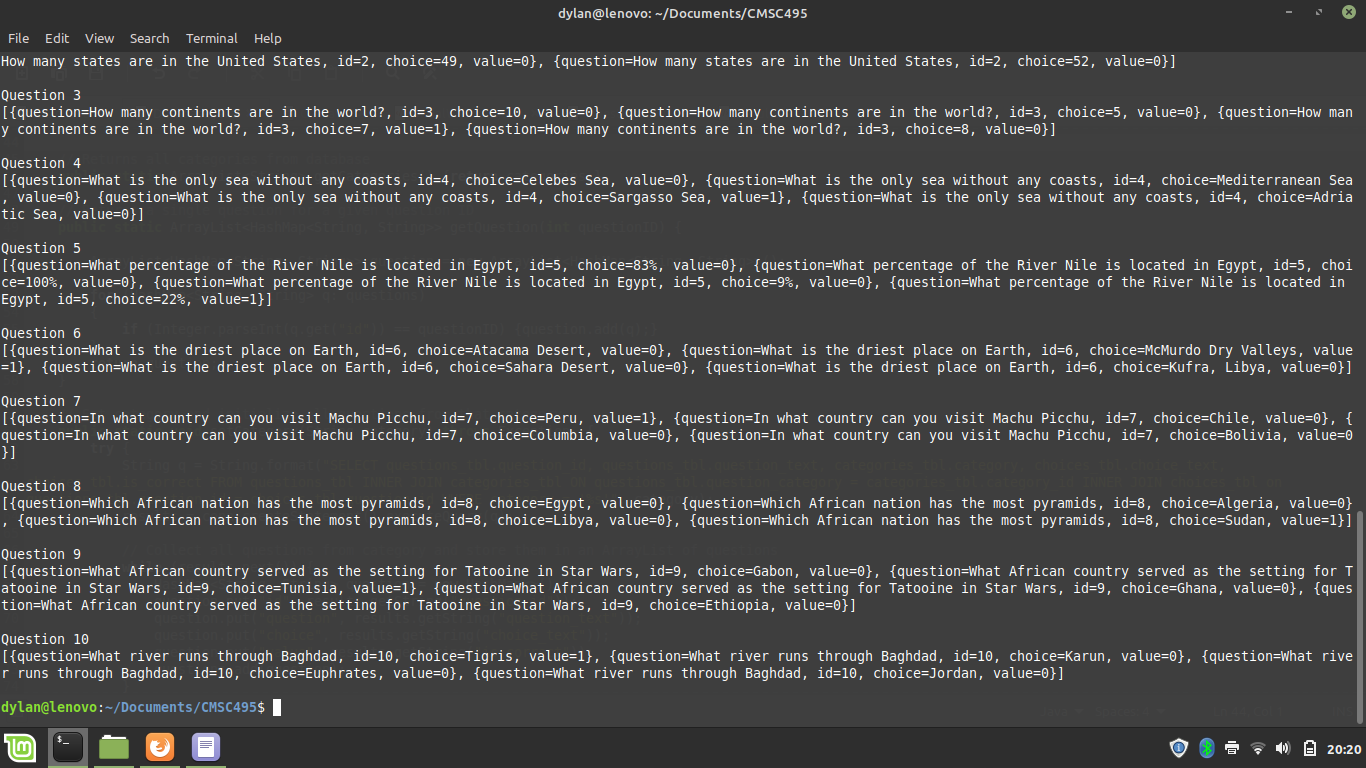
**Phase I Screenshots**



In the above screenshot of the Database.java source code, it shows an example of a query to the SQL database that selects all questions for a given category (requested by the user) and parses them into ArrayLists containing HashMaps. This format was chosen for simplicity and readability within the source code.

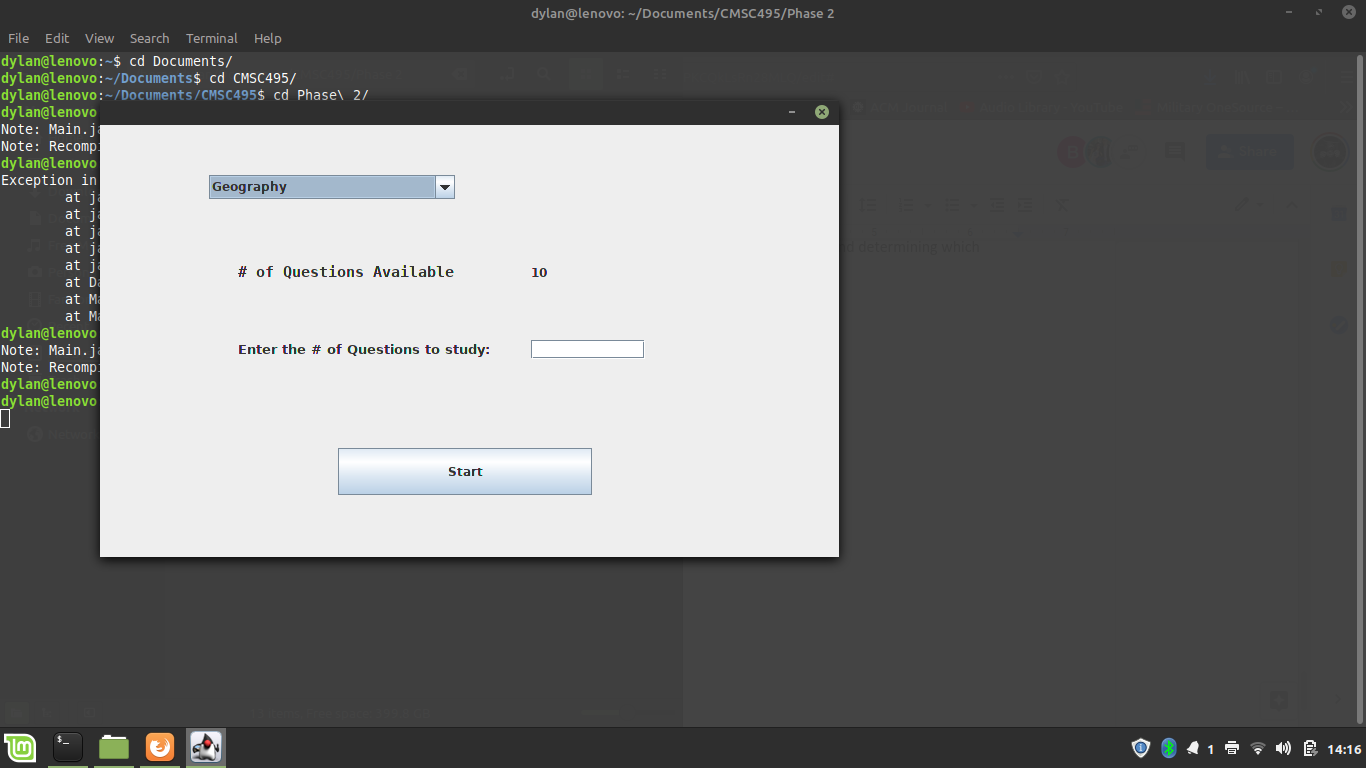


The above screenshot details an excerpt from the Main class that operates the Database.java. In this case, selecting the Geography database and requesting the first 10 questions in numerical order. Later this numeric choice will be randomized.

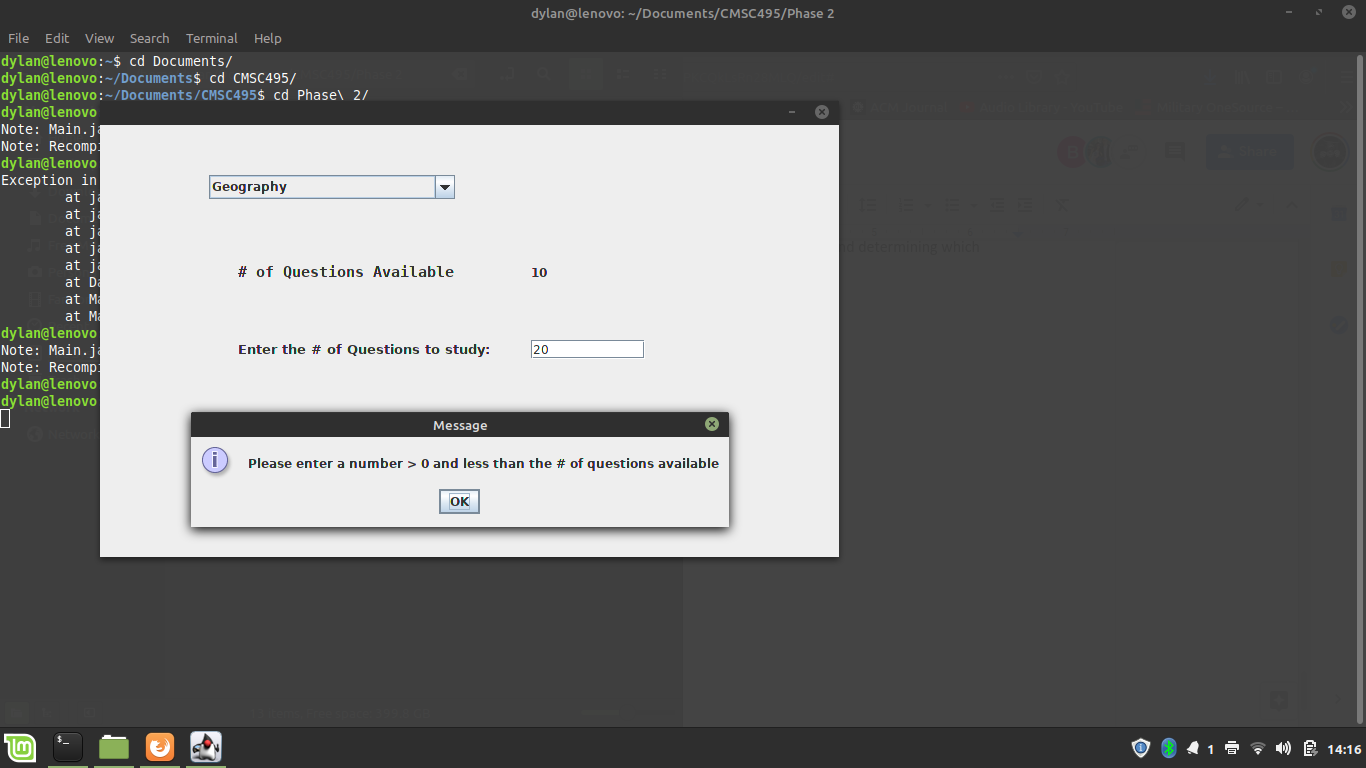


The above screenshot showcases the output of the Database.java requests in a programmatic context. It may not be very reader-friendly for a human but works very well when matching four choices to a given question (Choice A, B, C, D) and determining which choice is correct for user scores.

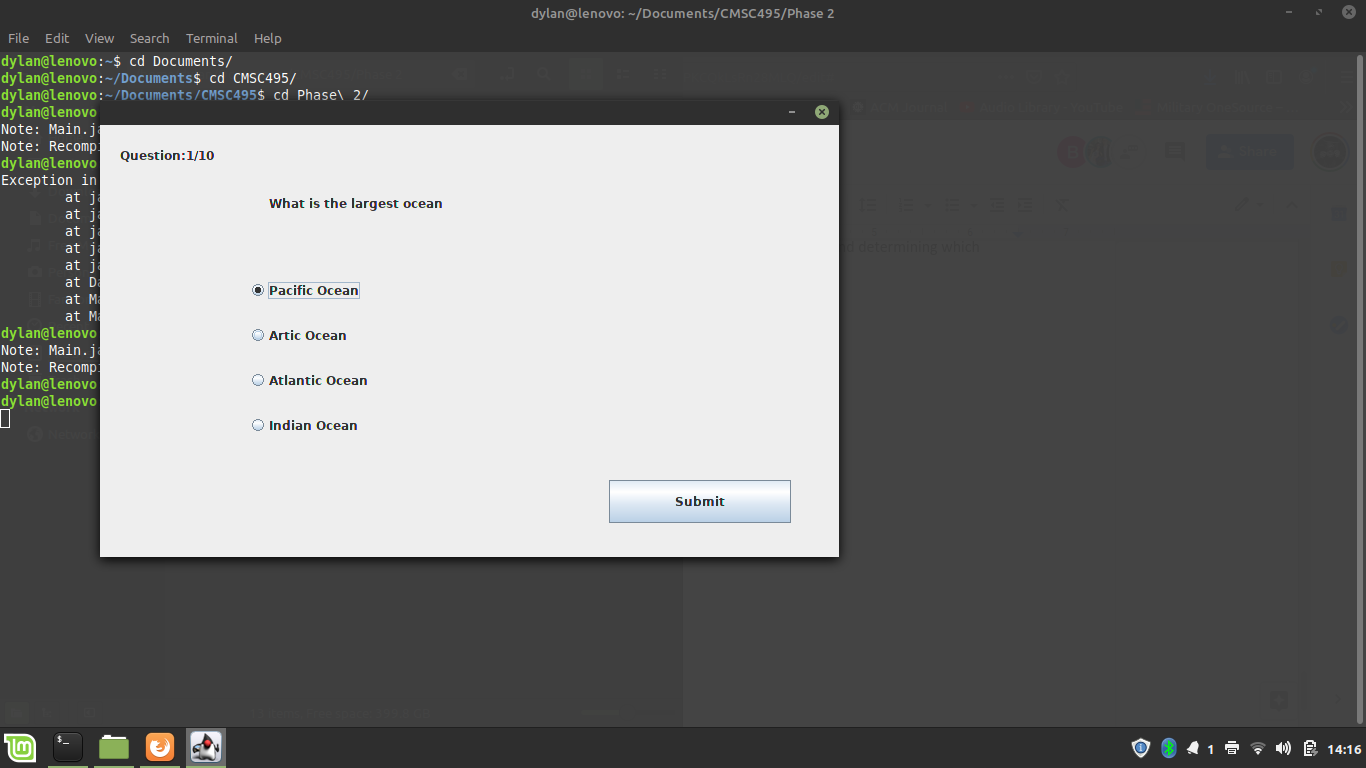
**Phase II Screenshots**



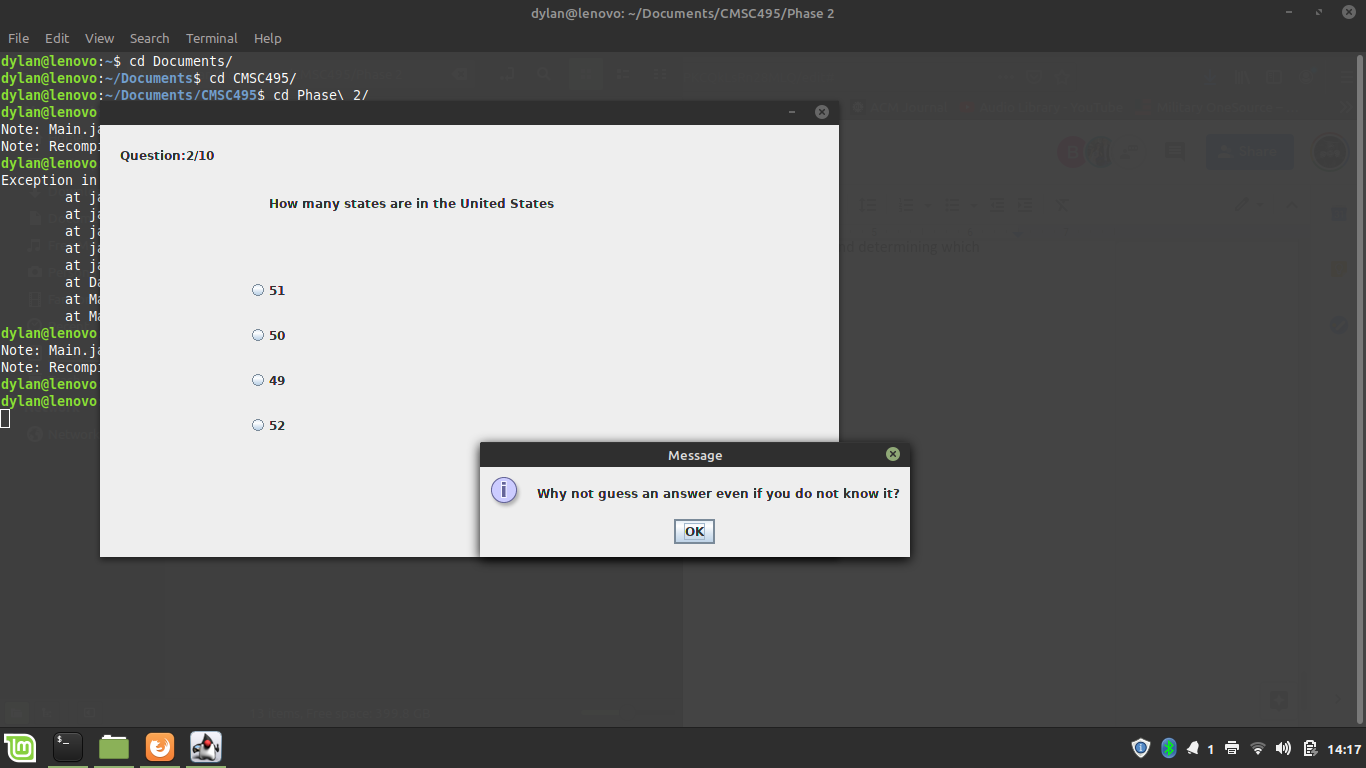
The above screenshot shows the GUI window for Phase II, which is now a part of the Main class.



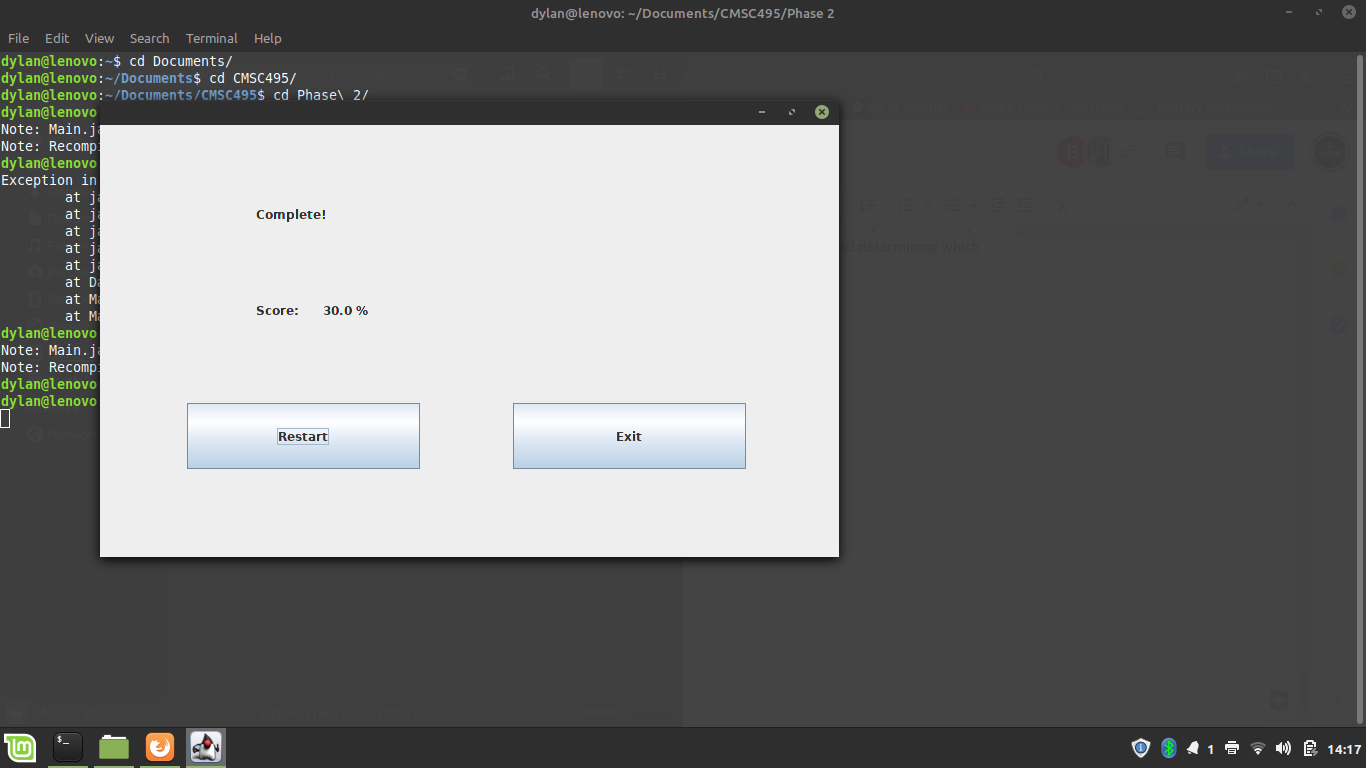
Phase II GUI still alerts the user if too many questions were entered.



Phase II GUI question example.

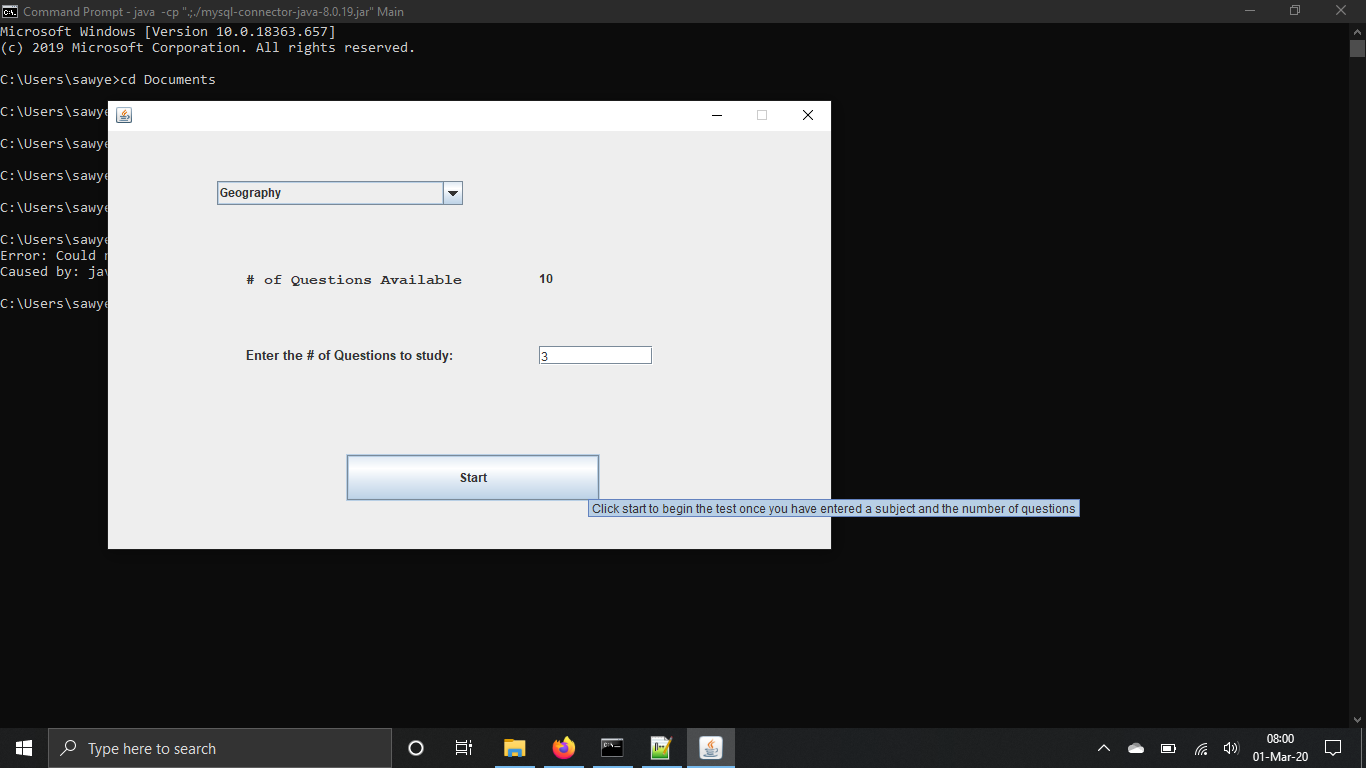


Phase II GUI question error example, if a user does not submit an answer.

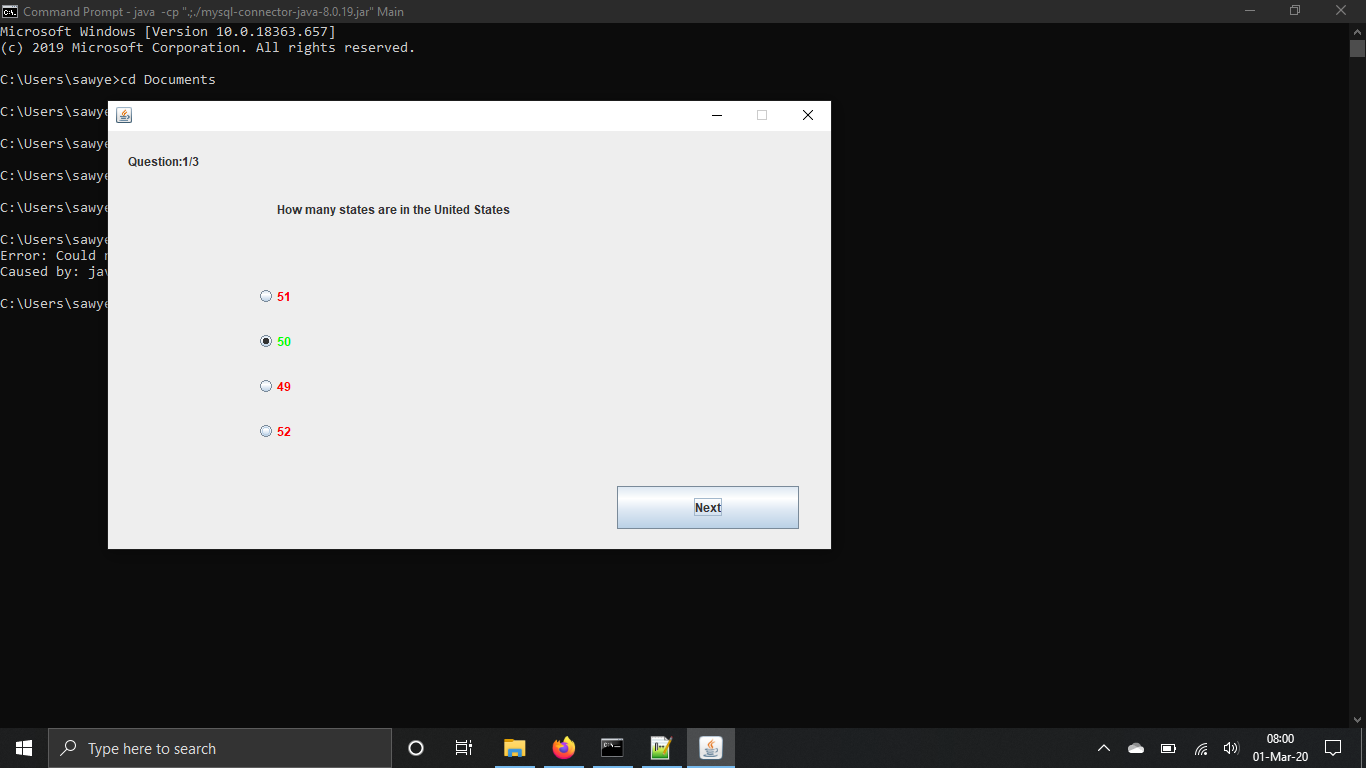


Phase II GUI end screen results. Restart takes the user back to the first window, exit will halt the program.

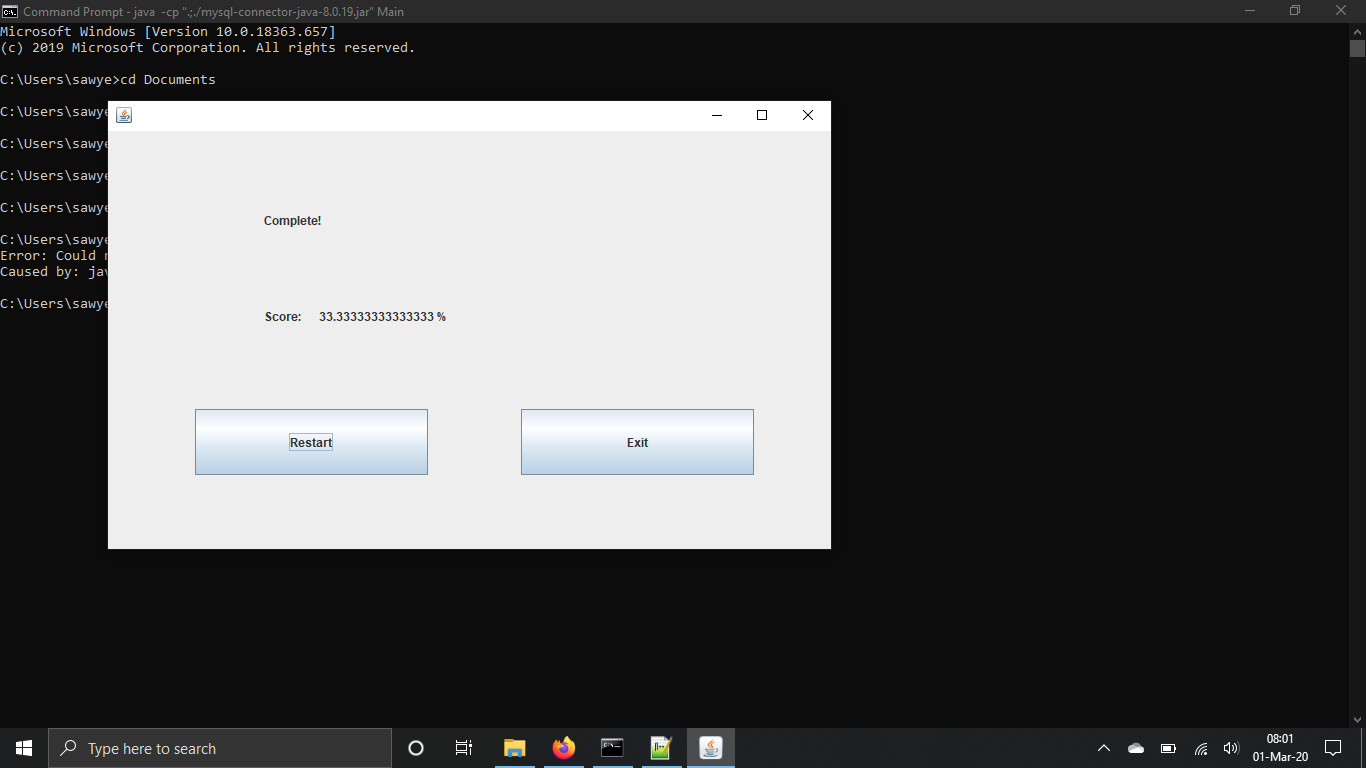
**Phase III Screenshots**

****

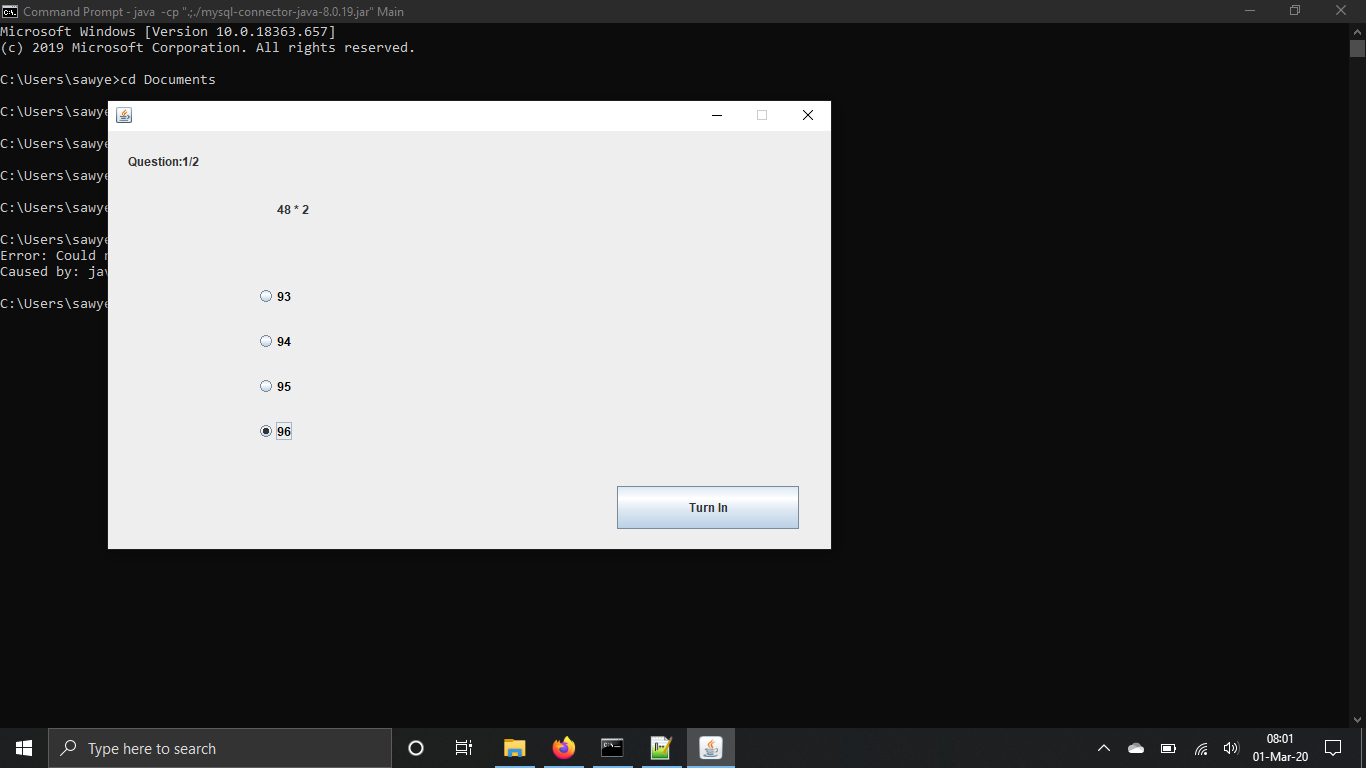
The initial GUI window works as expected. This time categories and the number of questions were updating directly from the database in real-time rather than hard-coded as we had been testing before.

****

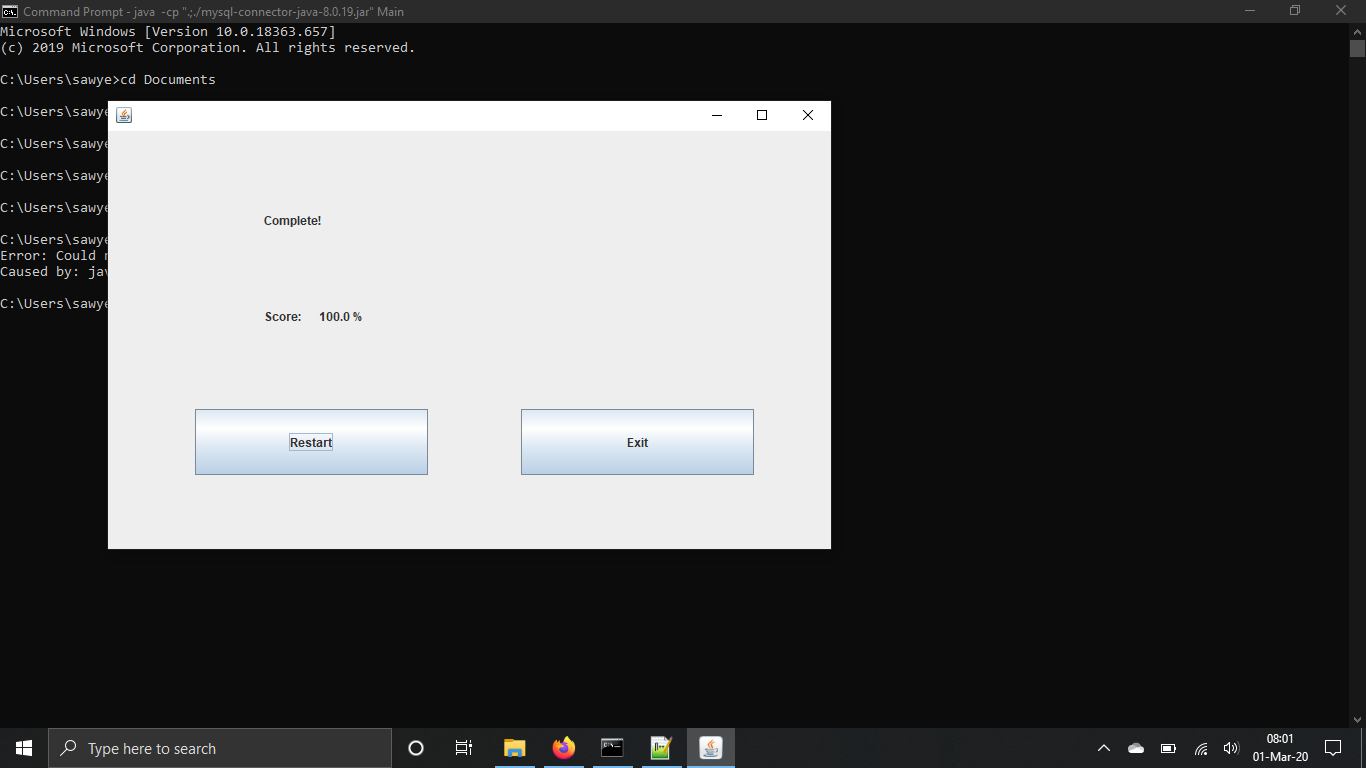
Answer highlights work properly, which highlights correct and incorrect answers via Choice and Question helper classes. (above)

****

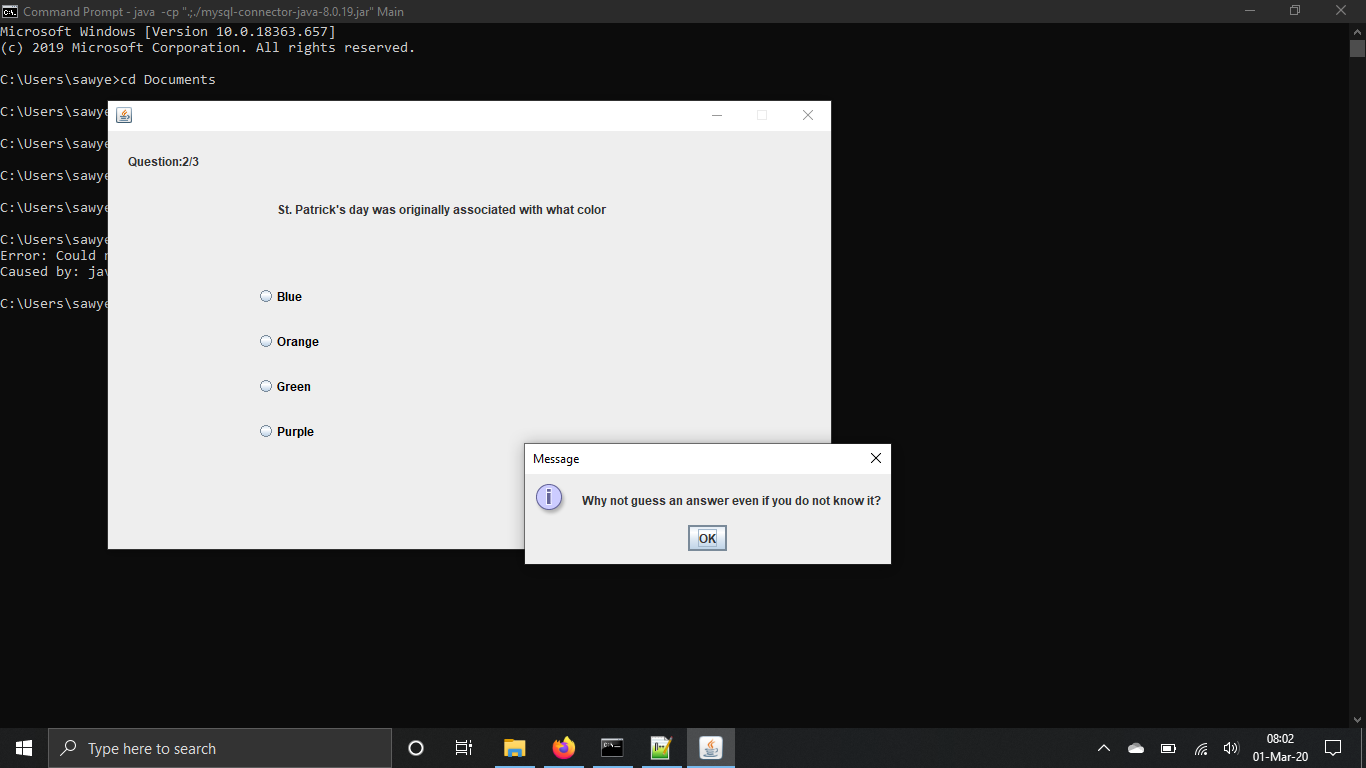
Scores generate correctly. Limiting the score to two decimal places in the future could be a possibility.

****

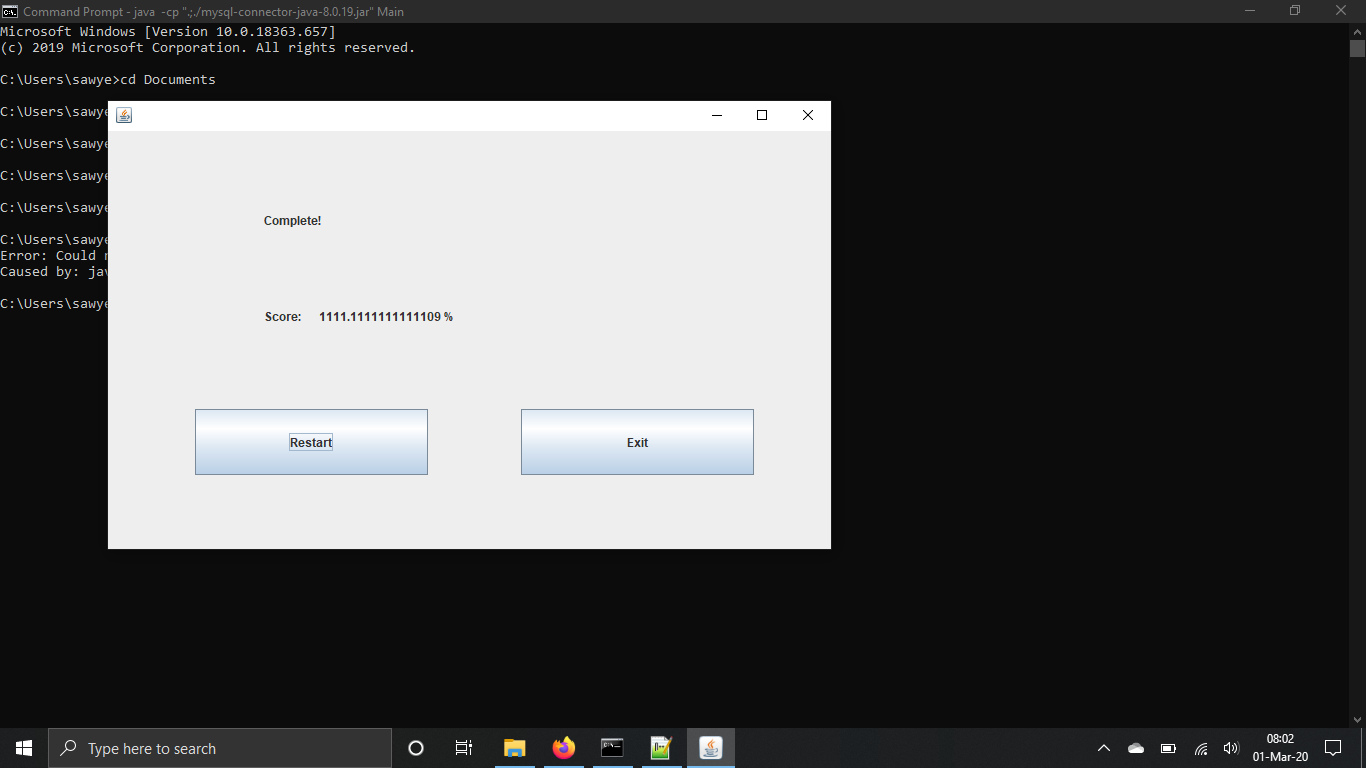
Upon restarting the quiz program, via the restart button, the answers no longer highlight and the first question is assumed to be the last question, as noted by the button that says ‘Turn In’ rather than ‘Submit’. (above)

****

However, upon the end of the second round of questions, the score generates correctly.

****

Round 3 of tests see the same errors as round 2, but also a popup behavior as if the user did not enter a question, even if he or she submitted an answer, regardless if it was correct or not. (above)

****

Round 3 scores also start varying wildly, typically ranging in the thousands.