Quizzer

A quiz application for Discrete Maths, Computer Organisation and Computer Science Foundation

Fred Sheppard - 23361433
Nanda - 23070854
CS4421 - Problem Solving with Computers

Table of Contents

Product Features

This application is a lightweight proof of concept that demonstrates the capabilities of a potential quiz application. The following features are supported:

- Secure login system that encrypts user passwords upon account creation, so that passwords may not be read from the file system by malicious outsiders.
- 18 questions based on Discrete Maths, Computer Organisation and Computer Science Foundation. Questions are read from the file system, so additional questions may be easily added by the client with no recompile required. New topics can be included by creating a new text file in the relevant directory.
- 3 game modes are included; Random, Escalation and Redemption. Random asks the questions in an arbitrary order. Escalation asks the questions in increasing order of difficulty. Redemption asks your worst-answered questions first, allowing you to correct your past mistakes. These three modes provide variety for the user, offering a more enjoyable experience.
- The CLI version clears the console at each menu to create a user-friendly, intuitive experience. Invalid input clears the input dialog, which is more appealing to the user than errors or crashes.
- The CLI version also contains a shutdown hook, which displays a friendly message if the program shuts down, even if it crashes.
- Statistics including the mean, median and standard deviation are calculated for each user, allowing them to be compared on a leaderboard. This creates a competitive experience.
- The GUI version creates a new window with an adjustable size to create an easy-to-use, easy-on-the-eyes experience.
- The GUI also replaces the window on screen with the required prompt instead of creating a new window, preventing cluttering of the user's screen.
- The GUI's methods have been written such that they can be called with 1-4 lines of code, making them easily reusable across different methods and classes with ease.
- The GUI code is easily compatible with the CLI version of the code, requiring only a couple lines of code to work with the GUI instead of the CLI.
- The codebase is modular and scalable. Each class handles a specific feature, and care was taken to write methods in a way that allows them to be ported from a CLI to a GUI or other API with ease.

Project Plan

Iteration	Feature	Details	Implementer
1	Project Plan, Feature layout	Work out what features we will include, how we will iterate upon these features, who will implement what	Joint
2	Question Bank	Create 18 questions in 3 topics: Computer Foundation, Computer Organisation and Discrete Maths. Each question will have one correct answer and 3 incorrect answers, along with the difficulty.	Nanda
3	Storing and Loading Questions	Create one file per topic which contains all the questions and answers for that topic, along with question metadata (difficulty etc.)	Fred
4	CLI	Create a prototype quiz which can be run from the CLI. This allows different quiz types to be tested and iterated upon without creating GUI elements.	Fred
5	Login & Users	Allow the user to create an account which will save their details on file. This will involve file I/O, hashing and elements of cybersecurity.	Nanda*
6	Basic Quiz Types	Implement the basic quiz types.. Random will shuffle the questions. Escalation will sort the questions by difficulty.	Random - Nanda* Escalation - Fred
7	Advanced Quiz Type - Redemption	A quiz type that remembers your worst answered questions and serves them up first.	Fred
8	GUI	Create a modern, sleek interface using JavaFX to allow the user to play the quiz.	Nanda
9	Statistics	Calculate mean, median and standard deviation. Implement a leaderboard in the GUI.	Calculations - Fred Leaderboard - Nanda

** Originally meant for Nanda, implemented by Fred due to external circumstances.*

Control Flow & Design

From the beginning, Quizzer was designed with modularity in mind. The goal was to place all UI-specific logic in main and use external classes to handle logging in, account creation, loading files and asking questions. This way, the same code could be used to create a CLI or a UI. This reusability became central to the design of the project.

Upon program launch, files containing users and topic questions are loaded into memory. The user is presented with a login prompt, where they may either log in to their existing account or create a new one. Password hashing is used to avoid writing credentials to files in plaintext. Upon login, a history file is created for the user if one does not exist already. This tracks how many times they have answered a question incorrectly across sessions.

The user is then asked to choose a quiz topic. These topics are read from a directory on the file system, and so any amount of custom topics may be uploaded. After choosing a topic they then select a gamemode. Game modes include Random, Escalation and Redemption, as explained in the list of features. Once a topic and a mode are selected, the quiz may begin.

The quiz begins with a QuestionLoader. This class handles file I/O and the potential errors that come with that. Each file in the questions folder represents a topic. The QuestionLoader reads each file and creates a new Question object for each line. In the Question constructor, the line is parsed to give the question, answer and wrong answers. Each question is stored in an ArrayList for easy sorting and shuffling.

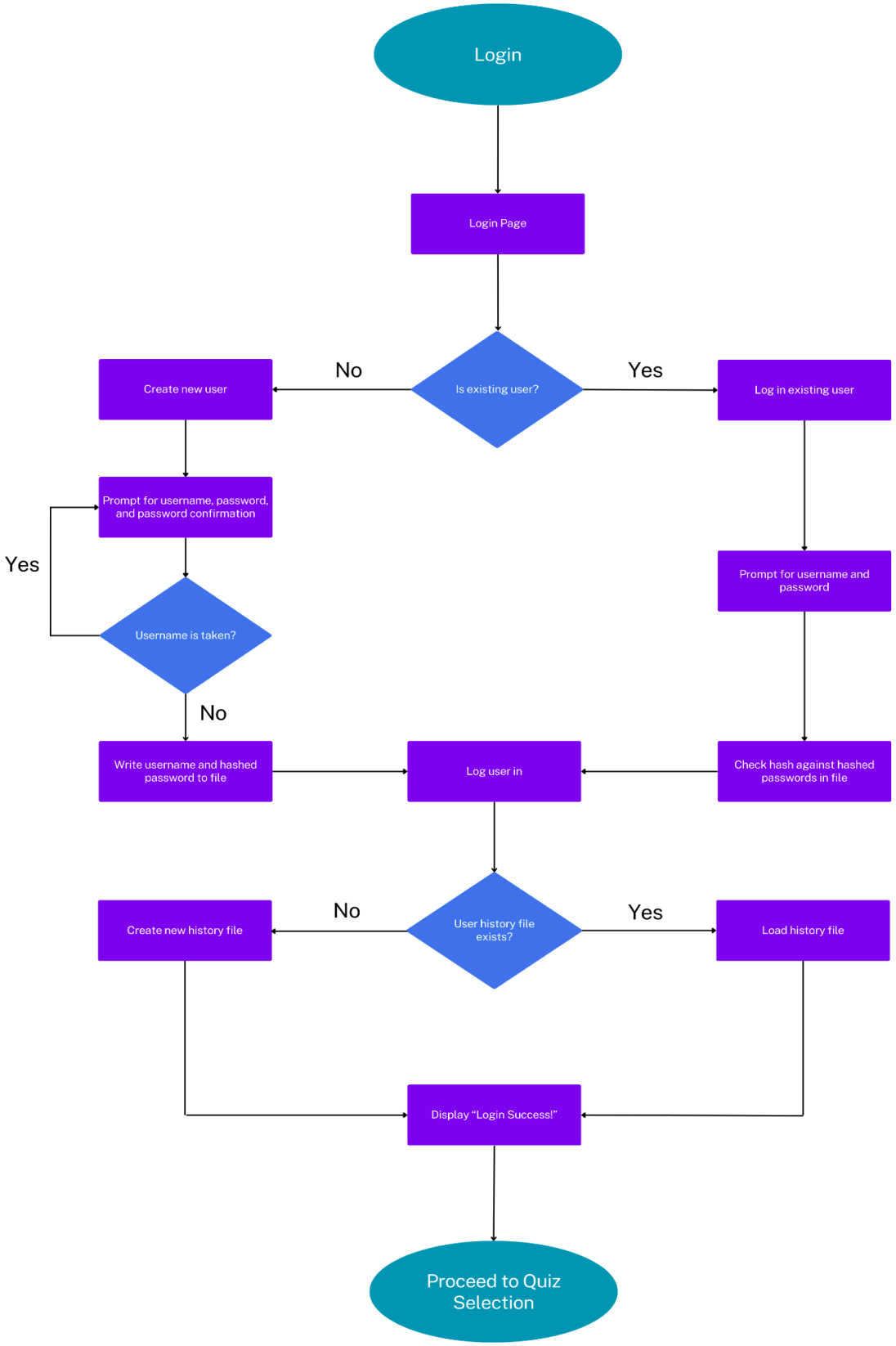
When the quiz type is selected by the user, one of three functions is called. askRandom() shuffles the list before asking the questions. askEscalation() sorts the list by difficulty. askRedemption uses the user's history to find their worst-answered questions. Each function then calls the same askQuestions() method, resulting in a uniform standard for adding game modes.

The program loops through all the questions in the now-sorted list, and displays them one by one. The possible answers are retrieved from the Question object and shuffled. The user then selects their choice, by keyboard on the CLI and by pressing the button on the GUI. The user's history is updated to reflect if they answered correctly or incorrectly. This repeats for all questions in the list. At the end, the user's score is displayed along with a percentage.

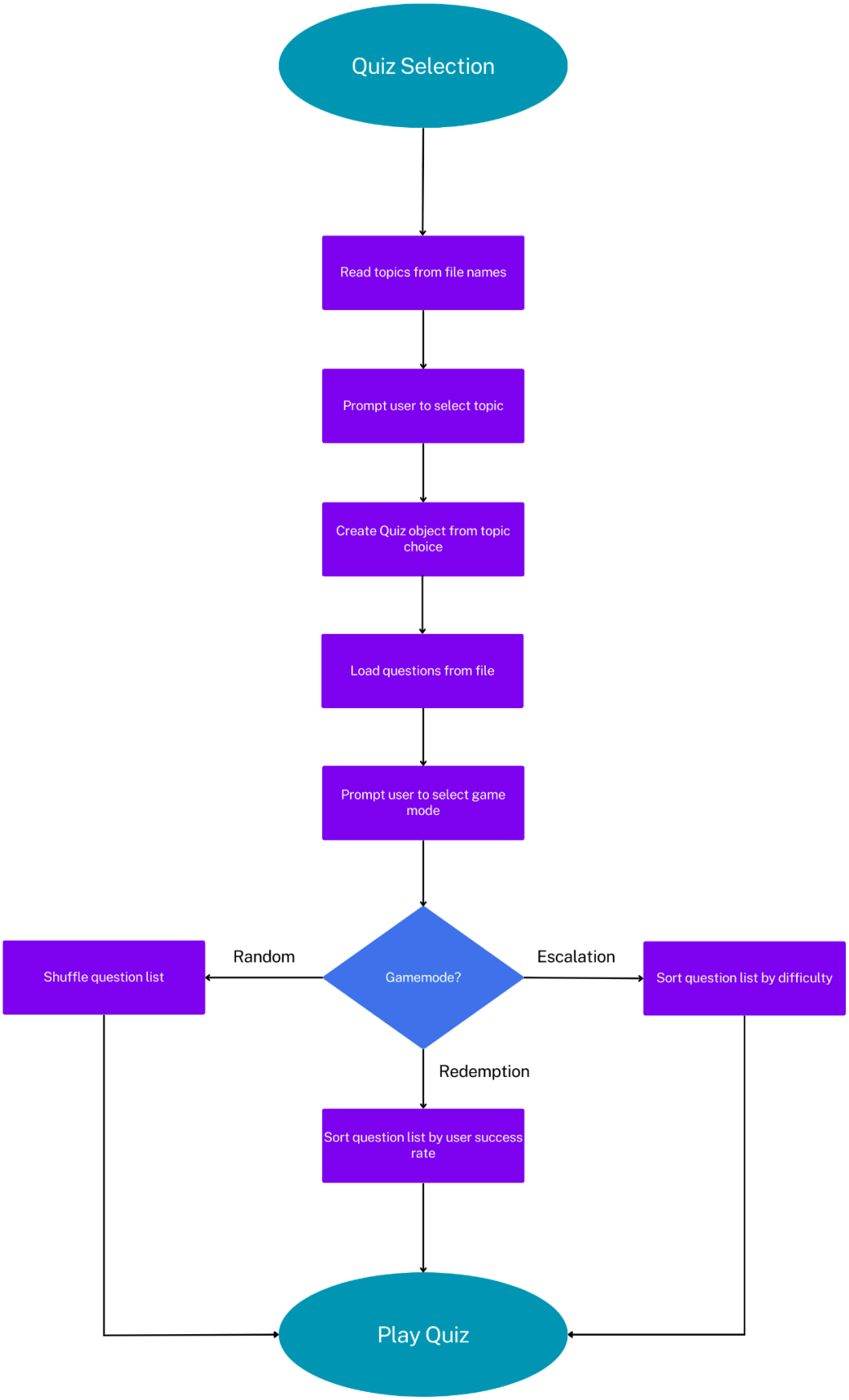
To calculate the statistics, heavy use was made of the functional programming paradigm. This was viable since the calculations involved performing operations on lists and arrays, which is suited to functional programming. An enum is used to declare the type of statistic desired, since there is a finite amount of valid inputs. Each calculation had to undergo unit testing to ensure that it was mathematically sound.

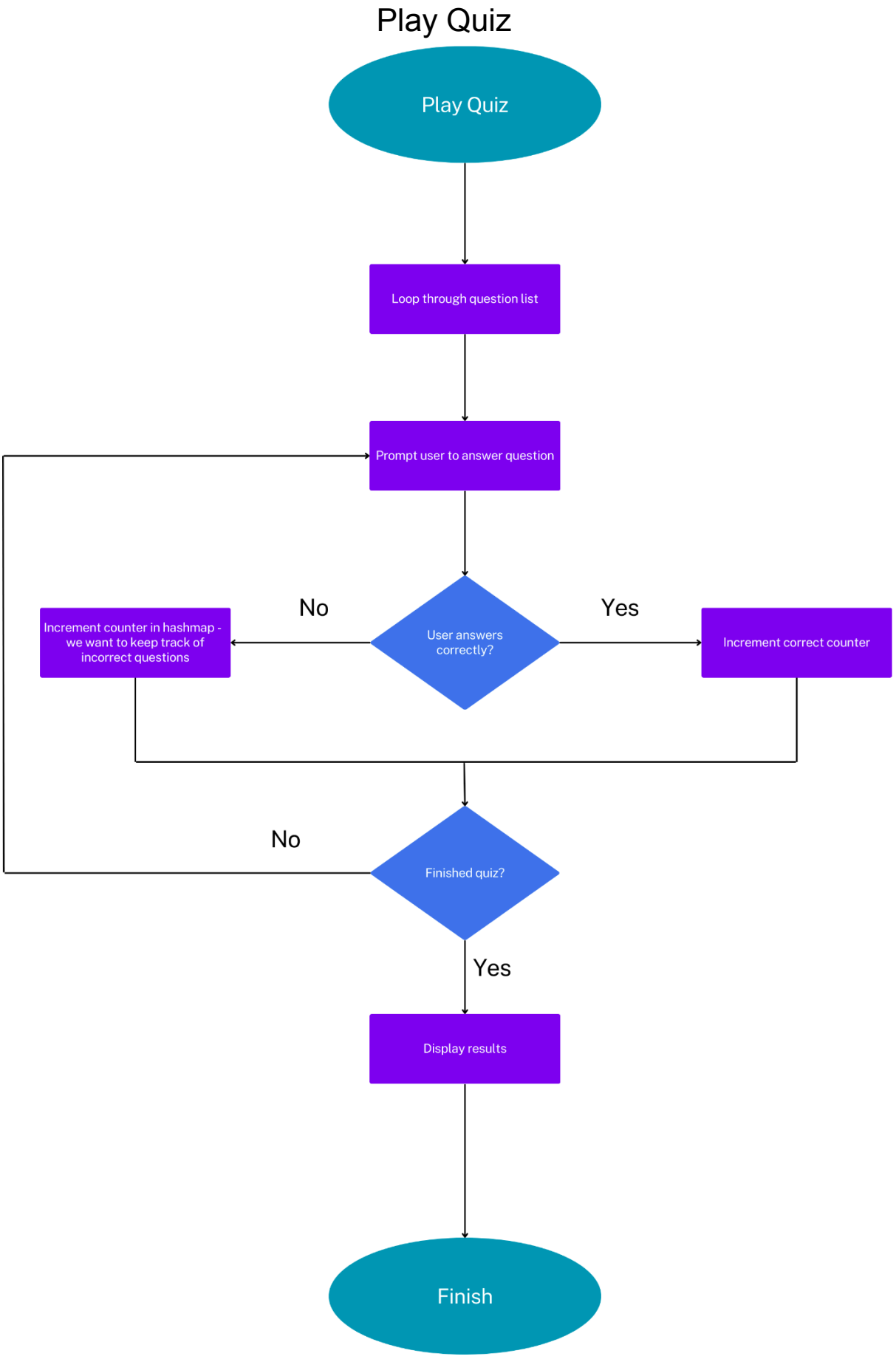
If the app closes at any point, the user is presented with a "Thanks for playing" message, which was implemented using a shutdown hook.

Login Page


Login
Login Page
Is existing user?
No
Yes
Create new user
Log in existing user
Prompt for username, password, and password confirmation
Yes
Username is taken?
No
Prompt for username and password
Write username and hashed password to file
Log user in
Check hash against hashed passwords in file
User history file exists?
No
Yes
Create new history file
Load history file
Display "Login Success!"
Proceed to Quiz Selection

Quiz Selection




Play Quiz
Play Quiz
Loop through question list
Prompt user to answer question
User answers correctly?
No
Yes
Increment counter in hashmap - we want to keep track of incorrect questions
Increment correct counter
Finished quiz?
No
Yes
Display results
Finish

Question Bank

DISCRETE MATHS:

Novice:

1. Which of the following is a basic operation in set theory?

A. Union
B. Differentiation
C. Multiplication
D. Factorization

2. Which of the following best describes a bijection?

A. A function that is both injective (one-to-one) and surjective (onto).
B. A function that has multiple outputs for one input.
C. A function where every element in the domain has a unique image in the codomain.
D. A function that is neither injective nor surjective.

Intermediate:

3. The function f(x) = 2f(x-1) is what type of sequence?

A. Recursive
B. Arithmetic
C. Geometric
D. Quadratic

4. In proof by contradiction, what is the negation of p -> q?

A. q ^ ¬p
B. ¬p -> ¬q
C. ¬p -> ¬q
D. p ^ ¬q

Expert:

5.If A, B, and C are sets, and $A \subseteq B$ and $B \subseteq C$, then which of the following is necessarily true?

a) $A \cap C = A$
b) $A = B$
c) $B \cup C = C$
d) $A - C = \varnothing$

6. Given the Boolean expression: $\neg(A \wedge B) \wedge \neg(C \vee D)$, which of the following is its equivalent after applying De Morgan's theorem?

a)$A' \wedge B' + C' \wedge D'$*
b)$A' + B' + C' \wedge D'$
c) $A' \wedge B' \wedge C' \wedge D'$
d) $A' \wedge B' \vee C' \wedge D'$

Computer Science:
Novice:

1. **Which of the following is true for any reflexive relation R on a set A?***

   a) For every a in A, (a, a) must be in R.*
   b) If (a, b) is in R, then (b, a) must be in R.
   c) R contains no pairs (a, b) where a≠b.
   d) If (a, b) is in R and (b, c) is in R, then (a, c) must be in R.

2. **What does the symbol "::=" imply?**

A. Declaration
B. End of Sentence
C. AND
D. Alternative

Intermediate:
3. What is the function {[(¬t_1)](β)}'s Semantic Definition?

A. neg([t_1](β))
B. conj (([t_1](β)), ([t_2](β))
C. disj (([t_1](β)), ([t_2](β))
D. None of the above

4.A lattice is a partially ordered set in which every two elements have a least upper bound and a greatest lower bound. Which of the following properties does a lattice NOT necessarily satisfy?

a) Distributivity
b) Antisymmetric
c) Reflexivity
d) Completeness

Expert:
5.Which of the following characterises a complete lattice?
a)Every subset has an infimum and a supremum
b)The lattice is finite
c)Every element has a unique complement
d)It is always distributive.

6.Which of the following statements is true for a distributive lattice?

a)Every pair of elements has a supremum
b) Every element has a unique predecessor
c) Every pair of elements has a greatest common divisor
d) Every element has a unique complement

Computer Organization:

***Novice*:**

What does CPU stand for?

a) Central Processing Unit
b) Control and Processing Utility
c) Computer Performance Usage
d) Core Power Usage

2. Who invented the finite state machine?

a) Alan Turing
b) Charles Babbage
c) Ada Lovelace
d) Konrad Zuse

Intermediate:

3. Which register is known as the link register?

a) r15
b) r1
c) r0
d) r31

4. Where was the transistor invented?

a) Bell Labs
b) Analog Devices
c) Intel
d) Apple

Expert:

5. Convert 0xFE to binary:

a) 11111110
b) 254
c) 11111101
d) 11101101

6. Convert 0b10101010 to octal

a) 252
b) 190
c) 354
d) AA

Accountability

Lines of Code

Developer	Lines Added	Lines Removed	Total
Fred	1514	853	661
Nanda	2044	1676	368

Diary

Iteration	Implementer	Contributions	Issues Arising
1	Fred	Worked together to create a plan for the project, defining features that each person would implement.	None
1	Nanda		None
2	Nanda	Created a question bank for the project, with 6 questions each, of Novice, Intermediate, and Expert difficulty for Computer Organization, Computer Science, and Discrete Mathematics.	None
3	Fred	Established a standard for storing questions on file. Created classes and methods for loading and representing questions.	Originally questions were simply stored as `String` arrays. This proved difficult to deal with, and was replaced with a dedicated `Question` class.
4	Fred	Created helper methods to make working with the CLI more efficient and more user friendly. These methods included taking a valid numerical input and holding the program until input was detected.	Did not know how to clear the screen in a terminal. Found a solution online that allowed a dedicated method to be created. *(see references)*
5	Nanda	Began working on a login system before external circumstances forced Fred to implement this feature.	External circumstances forced a deviation from the original iteration plan. Fred would have to implement some features by himself.

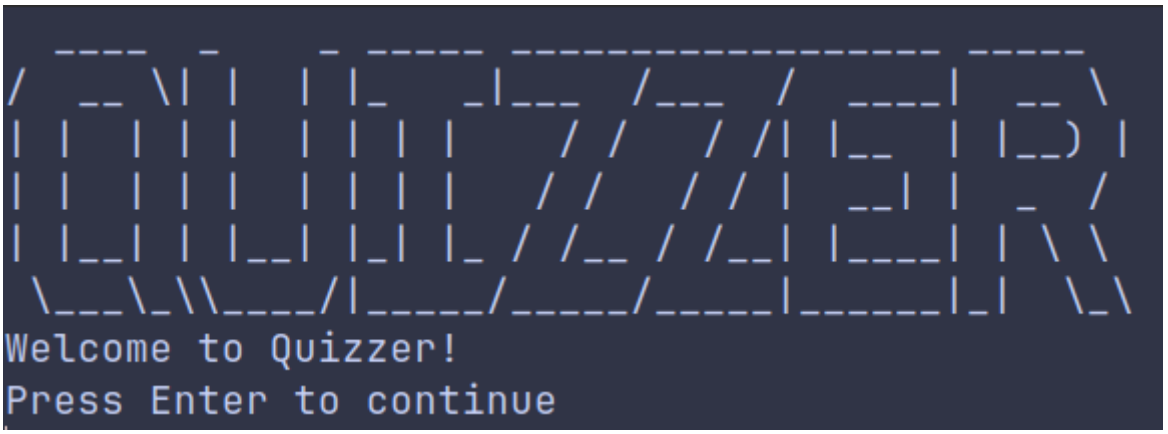
5	Fred	Created a login system that allows a user to create an account and later log in to that account.	Passwords were going to be stored in plaintext, but this was replaced with a hashing function for security.
6	Fred	Implement basic quiz types. These were very easy to implement due to how the Question and Quiz classes were laid out.	None
7	Fred	Implement the advanced quiz type - Redemption. This required additional data to be stored about users, namely their answer history. File IO featured heavily in this iteration.	Originally, user history was handled by the `Quiz` class. However, with the later implementation of user statistics, this was moved to a dedicated User class. This made code much more readable and optimised.
8	Nanda	Implement a GUI for the Login, Create Account, and Prompt pages using JavaFX. These pages must also be capable of taking inputs, processing them, and returning them to the functions they were called from.	Issues with the thread JavaFX uses being different from the “main” thread commonly used by Java, often leading to IllegalStateException. Resolved by setting IDEs to run on the JavaFX thread and using Platform.RunLater.
9	Fred	Created calculations for mean, median and standard deviation. Functional programming paradigms were used instead of loops to create readable, consistent code. This eliminates common errors such as “off-by-one” errors and indexing out of bounds.	Original implementation contained mistakes which were caught during the testing phase. Edge cases were not accounted for; calculating stats for a new user with no history caused a RuntimeException.

Code Listing

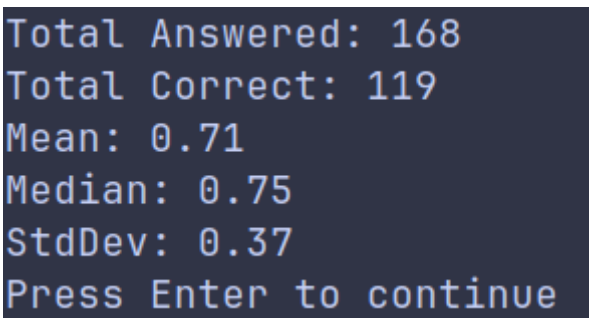
Source code may be found at https://github.com/Fred-Sheppard/Quizzer, and in the appendix at the end of this paper.
The CLI version is found in the CLI branch, and will run without any external libraries.
The GUI version is found in the GUI branch, and requires the JavaFX SDK to be installed to run.

Screenshots

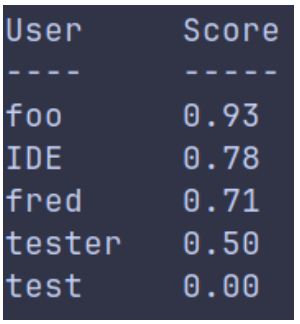
CLI Screenshots



Main welcome page



Statistics



Leaderboard

```
Which register is known as the link register?
(0) r1
(1) r0
(2) r15
(3) r31


(4) Exit
Choice:
```

Sample Quiz Question

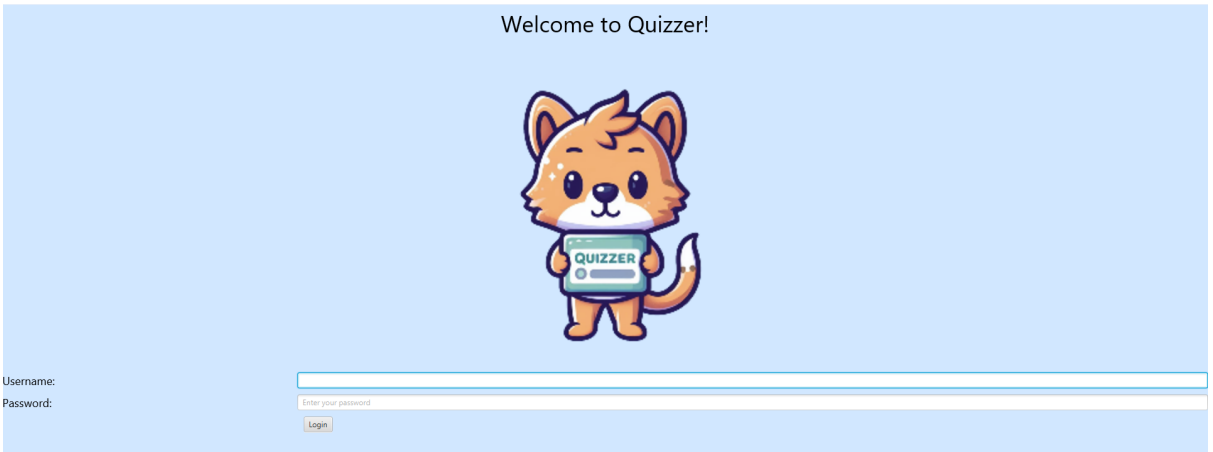
```
Enter username: fred
Enter password:
```

Login System

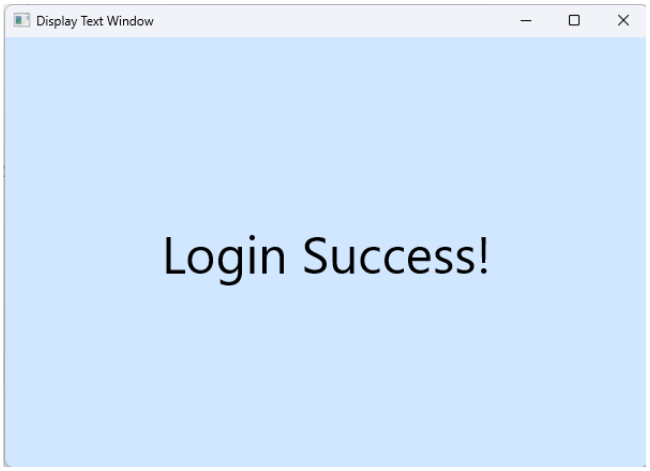
```
Choose a gamemode:
(0) Random
(1) Escalation
(2) Redemption
```

Game modes

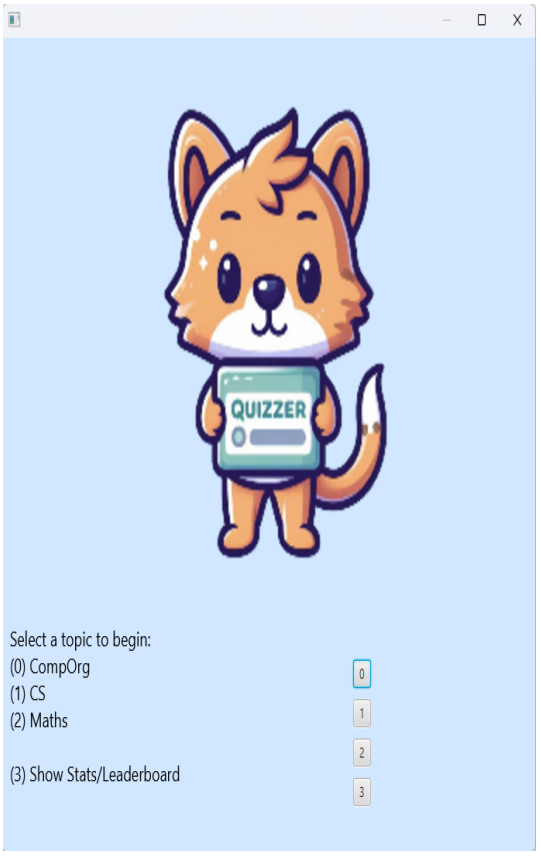
GUI Screenshots



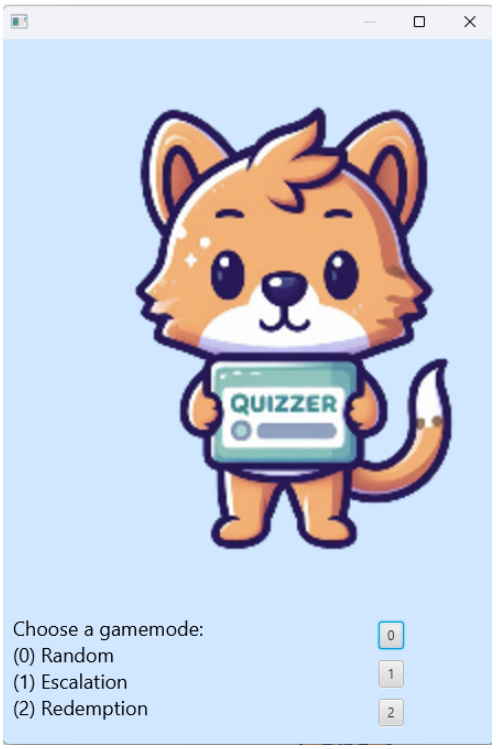
Welcome and Login Screen



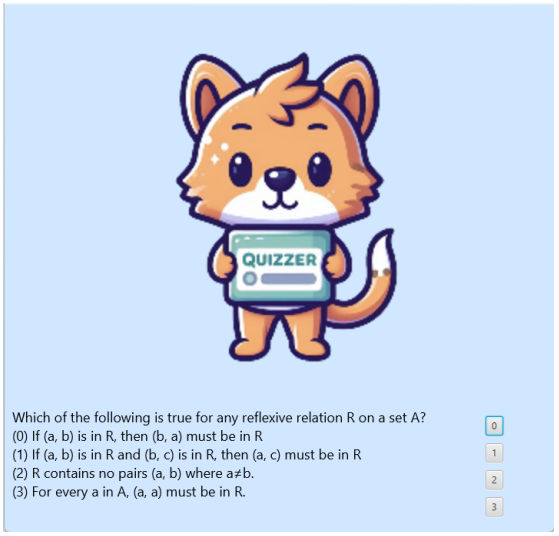
"Login Success!" Page



Topic Selection Screen



Gamemode Selection Screen



Sample Question Prompt

Innovation

For the innovation, we were interested in creating features that enhance the user experience. We approached this from two angles - File I/O and a Graphical User Interface.

File I/O

We make extensive use of file I/O across this project. When the user logs in, their credentials are checked against a file of hashed passwords. When playing a quiz, the user's history is written to a file to allow statistics to be calculated at a later date. Most importantly, file I/O allows the application to scale to the user's needs. More questions can be very easily added simply by updating the question files. This allows them to expand the app to fulfil their needs.

Reading and writing to files required research into best practices to inform us how to lay out the files. We decided to use the PSV (Pipe Separated Values) format for the question files. This is similar to CSV, but avoids the issue of the questions themselves containing commas. File contents are read using a `BufferedReader`, which makes it very efficient. The parsing of each line is handled by the `Question` class' constructor. This makes reading the file independent to the parsing, which is much more modular and scalable.

The Java documentation was used to gather much of the information required to make this run. The reference pages on `BufferedReader`, `File` and `ArrayList` were especially useful in implementing this functionality.

GUI

This project uses GUI extensively, as a cleaner, more appealing way of retrieving an input from the user. In various parts of the program, it will call JavaFX Applications, which will display a window presenting the user with the required screen. For example, when a program needs to display a piece of text, instead of using the terminal, the program will open a new window, with a chosen background colour and a logo loaded using the ImageView function.

Making JavaFX Applications work on the required thread and editing the layout to be compatible across different screen resolutions required deep research into the workings of Java, the Java Virtual Machine, and threads. It also made apparent the difficulty in using different IDEs across team members as we noticed both of us had to take different approaches to test the GUI applications. To research the program, I used online tutorials, which are linked in the References section of this Report.

Testing

Testing was used throughout the project to ensure that all code ran properly and executed as intended.

CLI Testing

Initial testing was manual, playing the quiz and ensuring each menu page displayed correctly. This was viable since each question has a limited number of possible answers to choose from, so the testing domain was small enough to test manually. Code coverage analysis was used to ensure that all methods and branches were tested thoroughly.

To test the statistics functions, unit testing was used. Each statistic type was given its own function, and both standard and edge cases were tested. Edge cases included:

- All questions answered correctly.
- No questions answered correctly.
- New user - has yet to answer any questions.

Testing proved vital for these functions, and helped to reveal several implementation errors. For example, initial implementations did not account for the fact that the user might not yet have played any quizzes, and this resulted in a runtime error.

```
public static void testMean() {
	User user = new User("test");
	var history = user.history;
	history.put("1", 1);
	history.put("2", 1);
	history.put("3", 0);
	history.put("4", 0);
	history.put("Rounds", 1);
	assert user.getStatistic(Statistic.MEAN) == 0.5;
	history.replace("1", 0);
	assert user.getStatistic(Statistic.MEAN) == 0.75;
	history.replace("2", 0);
	assert user.getStatistic(Statistic.MEAN) == 1.0;
	for (String key : history.keySet()) {
		history.replace(key, 1);
	}
	assert user.getStatistic(Statistic.MEAN) == 0.0;
	}
```

Example unit test for calculating the user's mean.

GUI Testing

Testing the GUI was done via calling the functions from a “test” class. It would create new instances of the functions and call them, to test if the Graphical User Interface displayed correctly and then to check if it was properly aligned. Multiple tests also had to be done to check the suitability of certain background colours, or to determine the best size Quizzer.png must be for a convenient and usable layout.

To test specific functions, each class had to be loaded and then called, to first display a text string, then to show the Login Page, testing the ability of the program to load an image given to it, and to contain a Username and Password field from the given JavaFX libraries, and to return those values to the original class, then printing it in the terminal to check if the value retrieved corresponds with the value sent. Then, a sample question is called, from which the answer is retrieved and returned to the original class.

```
DisplayTextWindow.launchWindow("TEST");
      promptLogin loginPage = new promptLogin();
      String[] credentials = loginPage.display();
      System.out.println("Username: " + credentials[0]);
      System.out.println("Password: " + credentials[1]);
      PromptInput prompt = new PromptInput();
      int choice = prompt.display("What does CPU stand for?\n1)Central
Processing Unit\n2)Control and Processing Utility\n3)Computer
Performance Usage\n4)Core Power Usage\nEASY", 3);  // 3 for four
choices: 0,1,2,3
      System.out.println("\nChoice is:"+choice);
      CreateAccount createAccountDialog = new CreateAccount();
      String[] accountDetails = createAccountDialog.display();
      System.out.println("Username: " + accountDetails[0]);
      System.out.println("Password: " + accountDetails[1]);
      System.out.println("Re-Entered Password: " + accountDetails[2]);
```

Example class used to test the GUI

Design Critique

The code is modularised, clean and scalable. From the beginning, methods were written with the goal of being scalable to various interfaces, including CLI, GUI and other APIs that may call it. Separate classes were created to handle user login, file I/O, and asking questions.

The code is well-documented. Each method has a document comment along with internal comments that illustrate the purpose of components. These comments make the code more readable, both for the writer and other members of the team.

Efforts were made to follow established design principles. Methods are kept short; if one grows too long, logic is extracted to another method. Classes and methods contain only one purpose; for example, when statistics were added, user logic was extracted from the `Quiz` class to a new `User` class.

Several tools were used to assess the quality of the code.

- IntelliJ coverage analysis - This tool monitors all classes, methods and lines during a code run and verifies that each was called. This pairs well with testing to ensure that all edge cases are accounted for. Coverage: 100%
- Qodana - Qodana is a state of the art static analysis tool developed by JetBrains. It applies a vast repository of lints to your entire project to catch bugs, logic errors and other issues. No issues were found by static analysis.

One area of improvement exists in the login system. Currently, a very basic hash function is used to encrypt passwords. This is more secure than storing in plaintext, but would not hold up long against a serious attack. In future, a dedicated library could be used to hash and salt the passwords. This would be much more secure and resistant to attackers.

Overall, the code is of good quality, but like all code can be improved.

Reflections

This has been a very enjoyable project with some intense challenges and opportunities for learning. There are many aspects and approaches we would change were we to do this again from the beginning.

A persistent issue we encountered was differences in workflow. We both used a different IDE, with a different project structure, with a different version of java, each working on a different branch. This caused serious issues when it came time to merge branches. For the next project, our first course of action will be to establish a standard that all members will use. This should make sharing code much easier.

Throughout this project, we learned that everything does not always go to plan. When events occur that are out of our control, it is important to be able to adapt our plans and keep moving forward. I feel like we handled ourselves very well, considering the circumstances. We were able to adjust our original iteration plan in a way that allowed the project to progress, while minimising any work shared unequally. While the final workload is not 100% shared evenly, it is the best that could be done given the timeframe.

Throughout this project, we both agreed that having the instructors present was a great benefit to our learning. We were able to come to them with issues and errors and learn how to fix them ourselves, as opposed to simply copy-pasting the answers from online. Similarly, having the ISE studio available was a huge resource for us. We found that we were far more productive working in person than at home. The social aspect also helped us to wind down in between coding sessions.

Overall this has been a great experience and we look forward to engaging with similar projects in the future.

References

E235 (2014) *How do I get "Press any key to continue" to work in my Java code?* available: https://stackoverflow.com/a/25095049 [accessed 12 Oct 2023]

Funkus (2020) *How do you clear terminal in Java? available:* https://replit.com/talk/ask/How-do-you-clear-terminal-in-Java/46341 [accessed 12 Oct 2023]

Bilash.saha (2011) Masking password input from the console : Java, available: https://stackoverflow.com/a/8138549 [accessed 16 Oct 2023]

Javatpoint Discrete Mathematics MCQ: https://www.javatpoint.com/discrete-mathematics-mcq

Maruthi Krishna (2020) How to display an image in JavaFX? https://www.tutorialspoint.com/how-to-display-an-image-in-javafx

Thenewboston (2018) JavaFX Java GUI Tutorial - 1 - Creating a Basic Window https://www.youtube.com/watch?v=FLkOX4Eez6o

Thenewboston (2018) JavaFX Java GUI Tutorial - 2 - Handle User Events https://www.youtube.com/watch?v=S_JN7zO12H4

Appendix

Main.java

```
import java.io.Console;
import java.io.File;
import java.util.Arrays;
import java.util.Scanner;

public class Main {

	/**
	* ASCII font of the Quizzer logo
	*/
	static final String QUIZZER = """
		 ____  _     _ _____ _____________________ _____
		/  __ \\| |   | |_   _|___  /___  /  ____|  __ \\
		| |  | | |   | | | |     / /   / /| |__  | |__) |
		| |  | | |   | | | |    / /   / / |  __| |  _  /
		| |__| | |__| |_| |_ / /__ / /__| |____| | \\ \\
		\\____\\_\\\\____/|_____/_____/_____|______|_|  \\_\\""";

	public static void main(String[] args) {
	// Print message when the program closes, even unexpectedly
	Runtime.getRuntime().addShutdownHook(new Thread(() -> {
		clearScreen();
		System.out.println("Thanks for playing!");
		System.out.println(QUIZZER);
	}));

	clearScreen();
	// Welcome messages
	System.out.println(QUIZZER);
	System.out.println("Welcome to Quizzer!");
	promptEnter();
	QuestionLoader loader = new QuestionLoader(new
File("res/questions/"));
	Scanner scanner = new Scanner(System.in);
	String[] topics = loader.listTopics();

	// Login
	Login login = new Login(new File("GameData/users.txt"));
	User user;
	String loginPrompt = """
			Are you a new or existing user?
```

```
                    (0) New
                    (1) Existing""";
        Console console = System.console();
        // If the program is run from within an IDE
        if (console == null) {
              System.out.println("""
                    You are running Quizzer from with an IDE.
                    Debugging mode enabled.
                    If you want the full experience, run Quizzer from a
terminal.""");
              promptEnter();
              user = new User("IDE");
        } else {
              boolean isExistingUser = promptInput(1, loginPrompt,
scanner) == 1;
              if (isExistingUser) {
                    user = promptLogin(login);
              } else {
                    user = promptCreateUser(login);
              }
        }

        // Continuously ask questions
        //noinspection InfiniteLoopStatement
        while (true) {
              int choice = chooseTopic(topics, scanner);
              if (choice == topics.length) {
                    showStats(user);
                    continue;
              }
              Quiz quiz = new Quiz(topics[choice], user, loader, scanner);
              int mode = promptInput(2, """
                    Choose a gamemode:
                    (0) Random
                    (1) Escalation
                    (2) Redemption""", scanner);
              switch (mode) {
                    case 0 -> quiz.askRandom();
                    case 1 -> quiz.askEscalation();
                    case 2 -> quiz.askRedemption();
                    default -> {
                    } //unreachable, since promptInput does not allow
invalid inputs
              }
        }
        }
```

```
	/**
	* Prompts the user to select a topic to be quizzed on
	*
	* @param topics  The list of available topics
	* @param scanner Scanner object to receive input
	* @return The user's choice
	*/
	public static int chooseTopic(String[] topics, Scanner scanner) {
	int options = topics.length;
	StringBuilder builder = new StringBuilder("Select a topic to
begin:\n");
	for (int i = 0; i < options; i++) {
		builder.append(String.format("(%d) %s%n", i, topics[i]));
	}
	builder.append(String.format("%n(%d) %s", options, "Show stats"));
	return promptInput(options, builder.toString(), scanner);
	}

	/**
	* Repeatedly prompts the user for a valid numerical input.
	* Valid inputs include any integer from 0 to `maxValid`,
inclusive.
	*
	* @param maxValid The largest number input that is valid
	* @param prompt   Prompt to display to the user displaying valid
options
	* @param input    Scanner object to receive input
	*/
	public static int promptInput(int maxValid, String prompt, Scanner
input) {
	while (true) {
		clearScreen();
		System.out.println(prompt);
		System.out.printf("%n(%d) Exit%n", maxValid + 1);
		System.out.print("Choice: ");
		try {
			// Parse this way to avoid infinite loops
			int choice = Integer.parseInt(input.next());
			// All prompts should include the option to exit
			if (choice == maxValid + 1) System.exit(0);
			// If the selection is valid
			if (0 <= choice && choice <= maxValid) return choice;
		} catch (NumberFormatException ignored) {
			// If the input was not a number at all, keep looping
		}
```

```
        }
        }

        /**
        * Clears the terminal
        */
        public static void clearScreen() {
        System.out.print("\033[H\033[2J");
        System.out.flush();
        }

        /**
        * Pauses the app until the user presses the enter key
        */
        public static void promptEnter() {
        System.out.println("Press Enter to continue");
        try {
              //noinspection ResultOfMethodCallIgnored
              System.in.read();
        } catch (Exception ignored) {
        }
        }

        /**
        * Prompts the user to enter their username and password.
        * Will loop until valid details are entered.
        *
        * @param login The login object to use
        */
        public static User promptLogin(Login login) {
        String user;
        clearScreen();
        Console console = System.console();
        while (true) {
              clearScreen();
              user = console.readLine("Enter username: ");
              var password = new String(console.readPassword("Enter
password: "));
              if (login.checkCredentials(user, password)) break;
        }
        System.out.println("Welcome back to Quizzer!");
        promptEnter();
        return new User(user);
        }

        /**
```

```
	* Prompts the user to create a new account.
	* Will loop until it succeeds.
	*
	* @param login The login object to use
	*/
	public static User promptCreateUser(Login login) {
	clearScreen();
	// Loop while the passwords don't match or the username is already
taken
	String name;
	String password;
	Console console = System.console();
	while (true) {
		String username = console.readLine("Enter your new username:
");
		char[] pass1 = console.readPassword("Enter your new
password: ");
		char[] pass2 = console.readPassword("Re-enter your new
password: ");
		// Assert the passwords are equal
		if (Arrays.equals(pass1, pass2)) {
			name = username;
			password = new String(pass1);
			// Loop if the user could not be created
			if (!login.createUser(name, password)) continue;
			break;
		}
		clearScreen();
		System.out.println("Passwords must be the same.");
	}
	System.out.println("Success! Welcome to Quizzer!");
	promptEnter();
	return new User(name);
	}

	public static void showStats(User user) {
	clearScreen();
	System.out.printf("Total Answered: %.0f%n",
user.getStatistic(Statistic.TOTAL_ANSWERED));
	System.out.printf("Total Correct: %.0f%n",
user.getStatistic(Statistic.TOTAL_CORRECT));
	System.out.printf("Mean: %.2f%n",
user.getStatistic(Statistic.MEAN));
	System.out.printf("Median: %.2f%n",
user.getStatistic(Statistic.MEDIAN));
	System.out.printf("StdDev: %.2f%n", User.stdDev());
```

```
		System.out.println();
		System.out.println(User.leaderboard());
		promptEnter();
		}
}
```

Login.java

```
import java.io.*;
import java.util.HashMap;


public class Login {

		private final File userFile;
		private final HashMap<String, String> map;

		/**
		* Creates a new Login object.
		*
		* @param userFile File in which credentials are to be saved
		*/
		@SuppressWarnings("ResultOfMethodCallIgnored")
		public Login(File userFile) {
		this.userFile = userFile;
		if (!userFile.exists()) {
			userFile.getParentFile().mkdirs();
			try {
				userFile.createNewFile();
			} catch (IOException e) {
				throw new RuntimeException(e);
			}
		}
		map = new HashMap<>();
		BufferedReader reader;
		try {
			reader = new BufferedReader(new FileReader(userFile));
		} catch (FileNotFoundException e) {
			throw new RuntimeException(e);
		}
		reader.lines().forEach(line -> {
			String[] split = line.split(",");
			map.put(split[0], split[1]);
		});
```

```
	}

	/**
	* Creates a user with the given username and password.
	*
	* @param user     The username of the user
	* @param password The password of the user
	* @return If the user was created successfully
	*/
	public boolean createUser(String user, String password) {
	if (map.containsKey(user)) {
		System.out.println("User already exists.");
		Main.promptEnter();
		return false;
	}
	int hashedPassword = password.hashCode();
	try (FileWriter writer = new FileWriter(userFile, true)) {
		writer.write(user + "," + hashedPassword + "\n");
	} catch (IOException e) {
		throw new RuntimeException(e);
	}
	return true;
	}

	/**
	* Checks a given username and password to see if they are equal
	*
	* @param user     The username of the user
	* @param password The password of the user
	* @return If the login was successful
	*/
	public boolean checkCredentials(String user, String password) {
	String hashedPass = String.valueOf(password.hashCode());
	if (!map.containsKey(user)) {
		System.out.println("Username not found");
		Main.promptEnter();
		return false;
	}
	if (!map.get(user).equals(hashedPass)) {
		System.out.println("Incorrect password");
		Main.promptEnter();
		return false;
	}
	return true;
	}
}
```

Question.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * A class representing a question, answer and set of wrong answers for
a topic,
 * to be loaded from a file.
 */
public class Question {

	public enum Difficulty {
	NOVICE,
	INTERMEDIATE,
	EXPERT
	}

	private final String question;
	private final String answer;
	private final String[] wrongs;
	private final Difficulty difficulty;

	public Question(String line) {
	String[] arr = line.split("\\|");
	question = arr[0];
	answer = arr[1];
	wrongs = Arrays.copyOfRange(arr, 2, 5);
	difficulty = Difficulty.valueOf(arr[5]);
	}

	public String toString() {
	return String.format("[Q: %s. A: %s. W: %s. D: %s]", question,
answer, Arrays.toString(wrongs), difficulty);
	}

	public String question() {
	return question;
	}

	public String answer() {
	return answer;
```

```
	}

	public String[] wrongs() {
	return wrongs;
	}

	public Difficulty difficulty() {
	return difficulty;
	}

	public ArrayList<String> possibilities() {
	ArrayList<String> list = new ArrayList<>(List.of(wrongs));
	list.add(answer);
	return list;
	}
}
```

Quiz.java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Scanner;

/**
 * Class representing a quiz.
 * <p>
 * A quiz is a set of questions asked in a specific order.
 * This order could be random, based on difficulty,
 * or on how successfully the user has answered each question in the
past.
 */
public class Quiz {

	private final String topic;
	private final ArrayList<Question> questions;
	private final Scanner scanner;
	private final User user;
```

```
	public Quiz(String topic, User user, QuestionLoader loader,
Scanner scanner) {
	this.topic = topic;
	this.user = user;
	questions = loader.getEntries(topic);
	this.scanner = scanner;
	}

	/**
	* Asks a single question, and verifies if the user entered the
correct answer.
	*
	* @param question The question to be asked
	* @return If the user selected the correct answer
	*/
	private boolean askQuestion(Question question) {
	Main.clearScreen();
	var possibilities = question.possibilities();
	Collections.shuffle(possibilities);
	int answer = possibilities.indexOf(question.answer());
	StringBuilder builder = new StringBuilder(question.question());
	builder.append("\n");
	for (int i = 0; i < 4; i++) {
		builder.append(String.format("(%d) %s%n", i,
possibilities.get(i)));
	}
	int choice = Main.promptInput(3, builder.toString(), scanner);
	if (choice == answer) {
		System.out.println("Correct! Well done.");
		user.history.putIfAbsent(question.question(), 0);
		return true;
	} else {
		System.out.println("Sorry. The correct answer was " +
answer);
		user.history.merge(question.question(), 1, Integer::sum);
		return false;
	}
	}

	/**
	* Asks all the questions in the given list.
	* <p>
	* This cannot be called by outside consumers,
	* who should instead call a helper method that specifies the order
to ask the questions in.
	*
```

```
	* @param questions The list of questions to be asked
	*/
	private void askQuestions(ArrayList<Question> questions) {
	Main.clearScreen();
	int numQuestions = questions.size();
	System.out.println("You have selected the " + topic + " topic.");
	System.out.printf("This topic contains %d questions.%n",
numQuestions);
	Main.promptEnter();
	int correct = 0;
	for (Question q : questions) {
		boolean userCorrect = askQuestion(q);
		if (userCorrect) correct++;
		Main.promptEnter();
	}
	Main.clearScreen();
	System.out.printf("Quiz complete! You got %d out of %d questions
correct! (%.0f%%)%n",
			correct, numQuestions, (float) correct / (float)
numQuestions * 100.0);
	user.history.merge("Rounds", 1, Integer::sum);
	PrintWriter writer;
	try {
		writer = new PrintWriter(new FileWriter(user.historyFile));
	} catch (IOException e) {
		throw new RuntimeException(e);
	}
	user.history.forEach((k, v) -> writer.println(k + "|" + v));
	writer.flush();
	writer.close();
	Main.promptEnter();
	}

	/**
	* Sorts and asks the questions, placing the user's worst-answered
questions first.
	*/
	public void askRedemption() {
	// Sort by the values in the map
	questions.sort((a, b) -> user.history.getOrDefault(b.question(),
0)
			.compareTo(user.history.getOrDefault(a.question(),
0)));
	askQuestions(questions);
	}
```

```
	/**
	* Asks the questions in a random order.
	*/
	public void askRandom() {
	Collections.shuffle(questions);
	askQuestions(questions);
	}

	/**
	* Asks the questions in order of difficulty.
	*/
	public void askEscalation() {
	// Sort the questions by difficulty
	questions.sort(Comparator.comparing(Question::difficulty));
	askQuestions(questions);
	}

}
```

Statistic.java

```
public enum Statistic {
	MEAN,
	MEDIAN,
	TOTAL_ANSWERED,
	TOTAL_CORRECT,
}
```

User.java

```
import java.io.*;
import java.util.*;

/**
 * A class representing a user of the quiz, created after logging in.
 * Contains methods for querying statistics of the user.
 */
public class User {

	/**
	* Username of the user.
	*/
```

```
	private final String name;
	/**
	* Map of all incorrectly answered questions,
	* along with how many times they have been answered incorrectly.
	* This is public to allow it to be edited directly by calling
code.
	*/
	public HashMap<String, Integer> history;
	/**
	* File containing the user's question history.
	* Public to allow it to be written to directly by calling code.
	*/
	public final File historyFile;

	/**
	* Creates a User with the given username.
	*
	* @param name The user's username
	*/
	@SuppressWarnings("ResultOfMethodCallIgnored")
	public User(String name) {
	this.name = name;
	historyFile = new File("GameData/UserHistory/" + name + ".txt");
	if (!historyFile.exists()) {
		historyFile.getParentFile().mkdirs();
		try {
			historyFile.createNewFile();
		} catch (IOException e) {
			throw new RuntimeException(e);
		}
	}
	BufferedReader reader;
	try {
		reader = new BufferedReader(new FileReader(historyFile));
	} catch (FileNotFoundException e) {
		throw new RuntimeException(e);
	}
	history = new HashMap<>();
	reader.lines().forEach(line -> {
		String[] split = line.split("\\|");
		history.put(split[0], Integer.parseInt(split[1]));
	});
	}

	public String name() {
	return name;
```

```
	}

	/**
	* Returns a given statistic for the user.
	* Valid statistics are:
	* <pre>{@code
	* MEAN
	* MODE
	* MEDIAN
	* TOTAL_CORRECT
	* TOTAL_ANSWERED}</pre>
	* <p>
	* To query standard deviation, use User.stdDev().
	*
	* @param stat The statistic to be queried
	* @return The value of the queried statistic
	*/
	public double getStatistic(Statistic stat) {
	// If user as never answered any questions
	if (history.isEmpty()) return 0;
	// How many rounds in total have been answered
	int rounds = history.getOrDefault("Rounds", 0);
	//noinspection unchecked
	var map = (HashMap<String, Integer>) history.clone();
	map.remove("Rounds");
	// List corresponding to each question, counting how many times
each was answered correctly
	List<Integer> corrects = map.values().stream().map(wrong -> rounds
- wrong).toList();
	// How many questions have ever been answered correctly
	int totalCorrect = corrects.stream().reduce(0, Integer::sum);
	return switch (stat) {
		// #correct questions divided by #total questions answered
		case MEAN -> (double) totalCorrect / (rounds * map.size());
		// The middle of the sorted list of correctly answered
questions
		case MEDIAN -> (double)
corrects.stream().sorted().toList().get(corrects.size() / 2) / rounds;
		case TOTAL_CORRECT -> totalCorrect;
		case TOTAL_ANSWERED -> rounds * map.size();
	};
	}

	/**
	* Calculates the standard deviation of all users' answer means.
	*
```

```
	* @return The standard deviation of all users
	*/
	public static double stdDev() {
	var means = userMeans();
	// Mean of the list of means (mu)
	// Same as means.sum() / means.size()
	double mu = means.stream().reduce(0.0, Double::sum) /
means.size();
	// SUM(x - mu)^2
	double xLessMuSquared = means.stream().reduce(0.0, (accum, x) ->
accum + (x - mu) * (x - mu));
	// stdDev = sqrt[ (x - mu)^2 / N ]
	int n = means.size();
	return Math.sqrt(xLessMuSquared / n);
	}

	/**
	* Returns a list of the means of all users.
	*
	* @return ArrayList of means
	*/
	public static ArrayList<Double> userMeans() {
	// Each user has a list of all the problems they've gotten wrong
	File userDir = new File("GameData/UserHistory/");
	// Each index will hold the mean of a user
	ArrayList<Double> means = new ArrayList<>();
	// Iterate through all user history files
	for (String name : Objects.requireNonNull(userDir.list())) {
		User user = new User(name.split("\\.")[0]);
		means.add(user.getStatistic(Statistic.MEAN));
	}
	return means;
	}


	/**
	* Returns a String representation of a leaderboard, comparing
users by their means.
	*
	* @return String leaderboard
	*/
	public static String leaderboard() {
	File userDir = new File("GameData/UserHistory/");
	TreeMap<Double, String> leaderboard = new TreeMap<>();
	for (String name : Objects.requireNonNull(userDir.list())) {
		User user = new User(name.split("\\.")[0]);
```

```
            leaderboard.put(user.getStatistic(Statistic.MEAN),
user.name());
      }
      StringBuilder builder = new StringBuilder("User \t Score\n");
      builder.append("---- \t -----\n");
      leaderboard.descendingMap().forEach((score, name) ->
builder.append(String.format("%s \t %.2f%n", name, score)));
      return builder.toString();
      }
}
```