Quiz game application

Created By:

1. **Armaan nakhunda – 02**
2. **Sushant Suresh Navle – 05**
3. **Nishal Poojary - 17**

(Dr. Sharmila Rajesh Ponnoran)
Supervisor

Department of Computer Engineering

K C College of Engineering and Management Studies and Research

Mith Bunder Road, Near Hume Pipe, Kopri, Thane(East)

University of Mumbai

(AY 2023-24)

CERTIFICATE

This is to certify that the Mini Project entitled **" Quiz game application"** is a bonafide work of **Armaan Nakhunda (02), Sushant Suresh Navle (05)**,Nishal poojary(17) submitted to the University of Mumbai in partial fulfilment of the requirement from the award of the degree of " Bachelor of Engineering" in "Computer Engineering".

(Dr. Sharmila Rajesh Ponnoran)
Supervisor

(Dr. Mahesh Maurya)
Head of Department

(Dr. Vilas Nitnavare)
Principal

_____

College Seal

Mini Project Approval

This Mini Project entitled " Quiz game application " by Armaan nakhunda (02); Sushant Suresh Navle(05); Nishal poojary (17) is approved for the degree of Bachelor of Engineering in Computer Engineering.

Examiners :

1. _____
   (Internal Examiner Name and Sign)

2. _____
   _
   (External Examiner Name and Sign)

Date:

Place:

Acknowledgement

We express our gratitude to Dr. Sharmila Ponnoran, our project supervisor, for her invaluable expertise and guidance, which steered us in the right direction throughout the project's development, and her insightful contributions significantly influenced the project's shape.

Content

Abstract

The Java Quiz Game application is a dedicated platform designed for Java enthusiasts, offering a wide range of interactive challenges within the realm of Java programming. With its competitive scoring system and user-friendly interface, this application seamlessly combines education with entertainment, making the process of mastering Java both enjoyable and engaging. Users can strategically navigate the game using lifelines, such as eliminating wrong options or relying on the computer's 80% accuracy for the right answer. The application prioritizes education, covering various aspects of Java programming and providing detailed progress tracking and statistics to support continuous improvement.

Code of project

```
// Import necessary packages
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;
import java.util.HashSet;
import java.util.Set;

public class QuizGameGUI extends JFrame {
    private JLabel questionLabel;// Label for displaying the current question
    private JRadioButton[] options;// Array of radio buttons for displaying answer choices
    private ButtonGroup optionGroup;// Button group to ensure only one answer choice is selected at a time
    private JButton nextButton;// Button for moving to the next question
    private JButton backButton;// Button for going back to the previous question
    private JButton pauseButton;// Button for pausing the game
    private JButton fiftyFiftyButton;// Button for using the 50/50 lifeline
    private JButton askFriendButton;// Button for using the ask-a-friend lifeline
    private boolean paused = false;// Flag for whether the game is currently paused
    private int currentQuestionIndex = 0;// Index of the current question being displayed
    private int score = 0;// Current score of the player
    private JTextArea summaryTextArea;// Text area for displaying a summary of the game after it ends
    private List<Question> allQuestions;// List of all available questions
    private List<Question> selectedQuestions;// List of questions selected for the current game
    private String[] userAnswers;// Array of the user's answers to each question
    private JFrame startupFrame;// Frame for selecting the number of questions to include in the game
    private JComboBox<Integer> questionCountComboBox;// Combo box for selecting the number of questions to include in
the game
    private JPopupMenu pauseMenu;// Popup menu for displaying options when the game is paused
    private Timer questionTimer;// Timer for each question
```

```
private int timerSeconds = 20; // Set the timer duration in seconds
private boolean timeUp = false;// Flag for whether the timer has run out for the current question
private Set<Integer> timedOutQuestions;// Set of questions for which the timer has run out
private StringBuilder summary;// StringBuilder for building the summary of the game
private int[] timeRemaining;// Array of time remaining for each question
private JLabel timerLabel;// Label for displaying the timer
private JFrame summaryFrame;// Frame for displaying the summary of the game

 // Constructor for the QuizGameGUI class
public QuizGameGUI() {
   // Create the startup frame
   createStartupFrame();

   // Initialize the JTextArea for the summary
   summaryTextArea = new JTextArea(20, 80);
   summaryTextArea.setEditable(false);

   // Initialize questions and userAnswers
   initializeQuestions();

   // Add key bindings to the content pane
   addKeyBindings();

   // Set up the main quiz frame
   setUpQuizFrame();

   // Show the startup frame
   startupFrame.setVisible(true);

   // Initialize the time remaining for each question
   timedOutQuestions = new HashSet<>();

   // Set the time remaining for each question to the timer duration
   timeRemaining = new int[allQuestions.size()];

   // Create the timer for each question
   Arrays.fill(timeRemaining, timerSeconds);
```

```
    // Timer for each question
    questionTimer = new Timer(1000, new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Decrement the time remaining for the current question
            timeRemaining[currentQuestionIndex]--;

            // Update the timer label
            if (timeRemaining[currentQuestionIndex] <= 6) {

                // Blink the timer label red at even seconds
                if (timeRemaining[currentQuestionIndex] % 2 == 0) {
                    timerLabel.setForeground(Color.RED);
                } else {
                    timerLabel.setForeground(Color.BLACK);
                }
            }
            else {
                // Reset the timer label color to black
                timerLabel.setForeground(Color.BLACK);
            }
            // Update the timer label
            timerLabel.setText("Timer: " + timeRemaining[currentQuestionIndex] + " seconds");

            // If the timer has run out
            if (timeRemaining[currentQuestionIndex] <= 0)
            {
                // Stop the timer
                questionTimer.stop();
                // Handle the timeout
                handleTimeout();
            }
        }
    });
}
```

```
// Method for creating the startup frame
private void createStartupFrame() {
    // Create the startup frame
    startupFrame = new JFrame("Quiz Startup");
    startupFrame.setSize(300, 150);
    startupFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    startupFrame.setLocationRelativeTo(null);

    // Panel for the startup frame
    JPanel startupPanel = new JPanel();
    // Set the layout to FlowLayout
    startupPanel.setLayout(new FlowLayout());

    // Label for the combo box
    JLabel label = new JLabel("Select the number of questions:");
    // Add the label to the panel
    startupPanel.add(label);

     // Create the combo box for selecting the number of questions
    Integer[] options = {5, 10, 15, 20};
    questionCountComboBox = new JComboBox<>(options);
    startupPanel.add(questionCountComboBox);

    // Create the start button
    JButton startButton = new JButton("Start Quiz");
    startupPanel.add(startButton);

    // Create Button for instructions
    JButton instructionsButton = new JButton("Instructions");
    startupPanel.add(instructionsButton);

    // Add the panel to the frame
    startupFrame.add(startupPanel);

    // Add key bindings to the content pane
    addKeyBindings();
```

```
    // Action listener for the start button
    startButton.addActionListener(new ActionListener()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            // Get the selected number of questions
            int selectedQuestionCount = (int) questionCountComboBox.getSelectedItem();

            // Select random questions based on the user's choice
            selectRandomQuestions(selectedQuestionCount);

            // Load the first question
            loadQuestion(currentQuestionIndex);

            // Hide the startup frame and show the quiz frame
            startupFrame.setVisible(false);
            setVisible(true);
        }
    });

    // Action listener for the instructions button
    instructionsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // Open a new frame for instructions
            showInstructionsFrame();
        }
    });
}

// Method for setting up the quiz frame
private void setUpQuizFrame()
{
    // Set the title of the frame
    setTitle("Quiz Game");
```

```
// Set the size of the frame
setSize(1100, 300);

// Set the default close operation
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Set the location of the frame to the center of the screen
setLocationRelativeTo(null);

// Create a new panel
JPanel panel = new JPanel();

// Set the layout to BorderLayout
panel.setLayout(new BorderLayout());

// Create a new label for the question
questionLabel = new JLabel();

// Add the label to the panel
panel.add(questionLabel, BorderLayout.NORTH);

// Create a new panel for the answer choices
JPanel optionsPanel = new JPanel();

// Set the layout to GridLayout with 4 rows and 1 column
optionsPanel.setLayout(new GridLayout(4, 1));

// Create a new array of radio buttons for the answer choices
options = new JRadioButton[4];
// Create a new button group to ensure only one answer choice is selected at a time
optionGroup = new ButtonGroup();
// Loop 4 times
for (int i = 0; i < 4; i++)
{
   // Create a new radio button
   options[i] = new JRadioButton();
   // Add the radio button to the panel
```

```
    optionsPanel.add(options[i]);
    // Add the radio button to the button group
    optionGroup.add(options[i]);
}

// Add the options panel to the center of the main panel
panel.add(optionsPanel, BorderLayout.CENTER);

// Panel for the buttons
JPanel buttonPanel = new JPanel();
// Set the layout to FlowLayout
buttonPanel.setLayout(new FlowLayout());

// Label for displaying the timer
 timerLabel = new JLabel("Timer: " + timerSeconds + " seconds");
 // Center the text in the label
 timerLabel.setHorizontalAlignment(JLabel.CENTER);

  // Add the timer label to the panel
  JPanel timerPanel = new JPanel(new BorderLayout());
  // Add the timer label to the panel
  timerPanel.add(timerLabel, BorderLayout.CENTER);
  // Add the timer panel to the main panel
   add(timerPanel, BorderLayout.SOUTH);


   // Button for going back to the previous question
   backButton = new JButton("Back");
   // Add the button to the panel
   buttonPanel.add(backButton);

   // Button for moving to the next question
   nextButton = new JButton("Next");
   // Add the button to the panel
   buttonPanel.add(nextButton);

   // Button for pausing the game
```

```
pauseButton = new JButton("Pause");
// Add the button to the panel
buttonPanel.add(pauseButton);

// Button for using the 50/50 lifeline
fiftyFiftyButton = new JButton("50-50");
// Add the button to the panel
buttonPanel.add(fiftyFiftyButton);

// Button for using the ask-a-friend lifeline
askFriendButton = new JButton("Ask the Computer");
// Add the button to the panel
buttonPanel.add(askFriendButton);

// Add the button panel to the main panel
panel.add(buttonPanel, BorderLayout.SOUTH);

// Add the main panel to the frame
add(panel);

// Add key bindings to the content pane
addKeyBindings();

nextButton.addActionListener(new ActionListener()// Action listener for the next button
{
 @Override
 public void actionPerformed(ActionEvent e)// Method for handling the next button
 {
    // Check the user's answer
    checkAnswer();
    // Increment the current question index
    currentQuestionIndex++;
    // Clear the selected answer choice
    optionGroup.clearSelection();

    if (currentQuestionIndex < selectedQuestions.size())// If there are more questions
    {
```

```
            // Load the next question
            loadQuestion(currentQuestionIndex);
        }
        else
        {
            // Otherwise, show the result
            showResult();
        }
    }
});

backButton.addActionListener(new ActionListener()// Action listener for the back button
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // If the timer has expired, mark the current question as timed out
        if (timerSeconds <= 0 && !timedOutQuestions.contains(currentQuestionIndex))
        {
            // Add the current question index to the set of timed out questions
            timedOutQuestions.add(currentQuestionIndex);
        }

        if (currentQuestionIndex > 0)
        {
            // Decrement the current question index
            currentQuestionIndex--;
            // Load the previous question
            loadQuestion(currentQuestionIndex);
        }

        // Enable the back button for all questions before the current one
        // Loop through all questions before the current one
        for (int i = 0; i < currentQuestionIndex; i++)
        {
            // If the question has not timed out
            if (!timedOutQuestions.contains(i))
```

```
            {
                // Enable the back button
                backButton.setEnabled(true);
                break;
            }
        }
    }
});

// Action listener for the pause button
pauseButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Show the pause menu
        showPauseMenu();
    }
});

// Action listener for the 50/50 button
fiftyFiftyButton.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Use the 50/50 lifeline
        useFiftyFiftyLifeline();
    }
});

askFriendButton.addActionListener(new ActionListener()// Action listener for the ask-a-friend button
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Use the ask-a-friend lifeline
```

```
            useAskFriendLifeline();
        }
    });
}

// Method for initializing the questions
private void initializeQuestions()
{
    // Initialize the list of all questions
    allQuestions = new ArrayList<>();

    // Adding the questions to the list, in the format of question, option1, option2, option3, option4, correct answer.
    allQuestions.add(new Question("What does JVM stand for?", "Java Virtual Machine", "Just Very Much", "Jungle Virtual Mouse", "Java Virtual Method", "Java Virtual Machine"));
    allQuestions.add(new Question("What is a variable in Java?", "A reserved keyword", "A data type", "A storage location", "An operator", "A storage location"));
    allQuestions.add(new Question("Which data type is used for whole numbers in Java?", "float", "double", "int", "String", "int"));
    allQuestions.add(new Question("How do you declare a constant variable in Java?", "Using the 'var' keyword", "Using the 'let' keyword", "Using the 'final' keyword", "Using the 'const' keyword", "Using the 'final' keyword"));
    allQuestions.add(new Question("What is the main purpose of the 'public static void main(String[] args)' method?", "To declare variables", "To print output", "To initialize objects", "To start the program", "To start the program"));
    allQuestions.add(new Question("Which Java keyword is used to create a new instance of a class?", "new", "class", "instance", "this", "new"));
    allQuestions.add(new Question("What is the output of 'System.out.println(5 + 3 * 2)'?", "11", "16", "56", "26", "11"));
    allQuestions.add(new Question("Which operator is used for equality comparison in Java?", "==", "=", "!=", "===","=="));
    allQuestions.add(new Question("What is the keyword used to create a new class in Java?", "new", "class", "instance", "this", "class"));
    allQuestions.add(new Question("Which loop is used for iterating over elements of an array or collection in Java?", "for loop", "while loop", "if-else loop", "do-while loop", "for loop"));
    allQuestions.add(new Question("What is the correct syntax for a single-line comment in Java?", "// This is a comment", "/* This is a comment */", "# This is a comment", "<!-- This is a comment -->", "// This is a comment"));
    allQuestions.add(new Question("Which access modifier makes a class or method accessible only within the same package?", "public", "protected", "private", "default", "default"));
    allQuestions.add(new Question("What is the purpose of the 'this' keyword in Java?", "To create a new instance of a class", "To call a method of the superclass", "To refer to the current instance of a class", "To declare a constant", "To refer to the current instance of a class"));
    allQuestions.add(new Question("Which Java data type is used to store text?", "int", "char", "String", "float", "String"));
    allQuestions.add(new Question("What is the result of '10 % 3' in Java?", "1", "2", "3", "0", "1"));
    allQuestions.add(new Question("Which statement is used to exit a loop prematurely in Java?", "break", "continue", "return", "exit", "break"));
    allQuestions.add(new Question("What is the term for a function defined within a class in Java?", "Procedure", "Function", "Method", "Routine", "Method"));
```

```
allQuestions.add(new Question("What is the correct syntax to create a new object of a class in Java?", "new Object();", "create Object();", "Object.create();", "Object.new();", "new Object();"));

allQuestions.add(new Question("What is the default value of a boolean variable in Java?", "0", "1", "false", "true", "false"));

allQuestions.add(new Question("Which Java keyword is used to declare a constant variable?", "constant", "const", "final", "static", "final"));

allQuestions.add(new Question("What is the term for a class that cannot be instantiated and may have abstract methods?", "Interface", "Abstract class", "Concrete class", "Final class", "Abstract class"));

allQuestions.add(new Question("Which keyword is used to implement multiple inheritance in Java?", "inherit", "extends", "implements", "multiextends", "extends"));

allQuestions.add(new Question("What is the output of 'System.out.println(\"Hello\" + \"World\");'?", "Hello World", "Hello\nWorld", "Hello + World", "HelloWorld", "HelloWorld"));

allQuestions.add(new Question("In Java, a switch statement can be used with which data types?", "int", "float", "String", "All of the above", "int"));

allQuestions.add(new Question("What is the purpose of the 'default' case in a switch statement?", "To specify the default value", "To define the default behavior when no case matches", "To indicate an error", "To break out of the switch statement", "To define the default behavior when no case matches"));

allQuestions.add(new Question("Which operator is used for logical AND in Java?", "&", "&&", "||", "!", "&&"));

allQuestions.add(new Question("What is the term for a class that inherits properties and behaviors from another class in Java?", "Derived class", "Superclass", "Parent class", "Child class", "Child class"));

allQuestions.add(new Question("Which exception is thrown when an array index is out of bounds?", "IndexOutOfRangeException", "ArrayIndexException", "OutOfBoundsException", "ArrayIndexOutOfBoundsException", "ArrayIndexOutOfBoundsException"));

allQuestions.add(new Question("What is the purpose of the 'finally' block in a try-catch-finally statement?", "To specify the catch block", "To handle exceptions", "To ensure code is executed regardless of exceptions", "To skip the try block", "To ensure code is executed regardless of exceptions"));

allQuestions.add(new Question("What is the difference between '==' and '.equals()' when comparing strings in Java?", "'==' compares object references, '.equals()' compares string contents", "'==' compares string contents, '.equals()' compares object references", "There is no difference", "Both are used to compare object references", "'==' compares object references, '.equals()' compares string contents"));

allQuestions.add(new Question("Which Java keyword is used to explicitly call a superclass constructor?", "superclass", "base", "super", "parent", "super"));

allQuestions.add(new Question("In Java, which keyword is used to create an array?", "new", "array", "create", "make", "new"));

allQuestions.add(new Question("What is the result of '5 / 2' in Java?", "2.5", "2", "2.0", "2.25", "2"));

allQuestions.add(new Question("What is the purpose of the 'volatile' keyword in Java?", "To make a variable thread-safe", "To declare a constant", "To define a final variable", "To prevent variable modification", "To make a variable thread-safe"));

allQuestions.add(new Question("Which Java data type is used to represent a single 16-bit Unicode character?", "char", "byte", "int", "short", "char"));

allQuestions.add(new Question("What is the term for a method that has the same name as the class and is used to initialize objects?", "Constructor", "Initializer", "Destructor", "Accessor", "Constructor"));

allQuestions.add(new Question("What is the Java keyword used to create a subclass that inherits from a superclass?", "inherits", "extends", "implements", "inheritsfrom", "extends"));

allQuestions.add(new Question("What is the Java keyword used to refer to the current instance of a class within that class's methods?", "this", "self", "current", "instance", "this"));

allQuestions.add(new Question("Which access modifier allows a class or method to be accessible only within the same package or by subclasses?", "private", "public", "protected", "default", "protected"));
```

```
        allQuestions.add(new Question("What is the result of '5 + 5.0' in Java?", "10.0", "10", "5.0", "5", "10.0"));

        allQuestions.add(new Question("Which Java data type is used to store characters in Unicode format?", "char", "byte",
"int", "string", "char"));

        allQuestions.add(new Question("What is the purpose of the 'super' keyword in Java?", "To call a superclass method",
"To call a static method", "To create a new instance of a class", "To declare a constant", "To call a superclass method"));

        allQuestions.add(new Question("In Java, what is the term for hiding a class's implementation details and exposing only
necessary functionalities?", "Abstraction", "Encapsulation", "Inheritance", "Polymorphism", "Abstraction"));

        allQuestions.add(new Question("Which loop in Java is used for iterating a block of code repeatedly while a condition is
true?", "for loop", "while loop", "do-while loop", "if-else loop", "while loop"));

        allQuestions.add(new Question("What is the purpose of the 'break' statement in a loop?", "To exit the loop
prematurely", "To continue to the next iteration of the loop", "To skip the loop entirely", "To create a nested loop", "To exit
the loop prematurely"));

        allQuestions.add(new Question("What is the term for defining more than one method with the same name in a class,
but with different parameters?", "Method overloading", "Method overriding", "Method hiding", "Method chaining",
"Method overloading"));

        allQuestions.add(new Question("In Java, what keyword is used to declare a method that does not return a value?",
"void", "null", "return", "empty", "void"));

        allQuestions.add(new Question("What is the output of 'System.out.println(10 > 5 && 5 < 3);' in Java?", "true", "false",
"compile error", "runtime error", "false"));

        allQuestions.add(new Question("Which exception is thrown when an arithmetic operation results in a value that is too
large or too small to be represented in the data type?", "ArithmeticException", "OverflowException",
"NumberFormatException", "InvalidValueException", "ArithmeticException"));

        allQuestions.add(new Question("What is the term for a method that is defined in a subclass and provides a specific
implementation for a method declared in its superclass?", "Overloading", "Overriding", "Hiding", "Polymorphism",
"Overriding"));

        allQuestions.add(new Question("What is the result of '5 / 0' in Java?", "5", "0", "Infinity", "Runtime error", "Runtime
error"));

        allQuestions.add(new Question("In Java, which operator is used for bitwise AND?", "&", "&&", "|", "!", "&"));

        allQuestions.add(new Question("What is the purpose of the 'try' and 'catch' blocks in exception handling?", "To handle
exceptions", "To throw exceptions", "To declare variables", "To define methods", "To handle exceptions"));

        allQuestions.add(new Question("What is the term for the process of converting an object into a stream of bytes for
storage or transmission?", "Serialization", "Deserialization", "Encoding", "Decoding", "Serialization"));

        allQuestions.add(new Question("In Java, which keyword is used to create an interface?", "interface", "create", "new",
"implements", "interface"));

    }


    // Method for selecting random questions

    private void selectRandomQuestions(int count)

    {

        // If the user selects more questions than are available

        if (count >= allQuestions.size())

        {

            // Select all questions

            selectedQuestions = allQuestions;
```

```
    }
    else
    {
        // Create a new list of all questions
        List<Question> shuffledQuestions = new ArrayList<>(allQuestions);
        // Shuffle the questions
        Collections.shuffle(shuffledQuestions, new Random());
        // Select the first count questions
        selectedQuestions = shuffledQuestions.subList(0, count);
    }

    // Initialize userAnswers based on the selected question count
    userAnswers = new String[selectedQuestions.size()];
}

private void loadQuestion(int index)
{
    // Reset the timer for the current question to the timer duration
    timerSeconds = timeRemaining[index];
    questionTimer.restart();

    // Reset timeUp
    timeUp = false;

    // Get the current question
    Question currentQuestion = selectedQuestions.get(index);
    // Include question number
    questionLabel.setText("Question " + (index + 1) + ": " + currentQuestion.getQuestion());

    // Get the answer choices for the current question
    String[] answerChoices = currentQuestion.getAnswerChoices();
    // Loop 4 times
    for (int i = 0; i < 4; i++)
    {
        // Set the text for each radio button
        options[i].setText(answerChoices[i]);
        // Set enabled or disabled based on time remaining
```

```
        options[i].setEnabled(timeRemaining[index] > 0);
        // Clear the selected answer choice
        options[i].setSelected(false);
    }

    if (userAnswers[index] != null)// If the user has answered the current question
    {
        // Loop 4 times
        for (int i = 0; i < 4; i++)
        {
            // If the answer choice matches the user's answer
            if (options[i].getText().equals(userAnswers[index]))
            {
                // Select the answer choice
                options[i].setSelected(true);
                break;
            }
        }
    }
}



private void checkAnswer()
{
    // Loop 4 times
    for (int i = 0; i < 4; i++)
    {
        // If the answer choice is selected
        if (options[i].isSelected())
        {
            // Store the user's answer
            userAnswers[currentQuestionIndex] = options[i].getText();
            // If the user's answer is correct
            if (userAnswers[currentQuestionIndex].equals(selectedQuestions.get(currentQuestionIndex).getCorrectAnswer()))
            {
                // Increment the score
                score++;
```

```
            }
            break;
        }
    }

    // Variable for storing the selected answer choice
    String selectedAnswer = null;
    // Loop 4 times
    for (int i = 0; i < 4; i++)
    {
        // If the answer choice is selected
        if (options[i].isSelected())
        {
            // Store the selected answer choice
            selectedAnswer = options[i].getText();
            userAnswers[currentQuestionIndex] = selectedAnswer;
            break;
        }
    }
}

// Method for showing the result
private void showResult()
{
    // Stop the timer
    questionTimer.stop();

    // Display the summary screen
    StringBuilder summary = new StringBuilder();
    // Include the score
    summary.append("Quiz Complete!\nYour Score: ").append(score).append(" out of
").append(selectedQuestions.size()).append("\n\n");

    // Iterate through selected questions and display information
    for (int i = 0; i < selectedQuestions.size(); i++)
    {
        // Get the current question
```

```
    Question question = selectedQuestions.get(i);
    // Include the question number and question text
    summary.append("Question ").append(i + 1).append(": ").append(question.getQuestion()).append("\n");
    // Include the correct answer
    summary.append("Correct Answer: ").append(question.getCorrectAnswer()).append("\n");
    // Include the user's answer
    summary.append("Your Answer: ").append(userAnswers[i]).append("\n");

    // Check if the user's answer is correct
    boolean answeredCorrectly = userAnswers[i] != null && userAnswers[i].equals(question.getCorrectAnswer());

    // Indicate if the user answered correctly or not and print if the answer is correct or not
    if (answeredCorrectly)
    {
        summary.append("Result: Correct\n");
    }
    else
    {
        summary.append("Result: Incorrect\n");
    }

    // Include the remaining time for each question
    summary.append("Time Remaining: ").append(timeRemaining[i]).append(" seconds\n\n");
}

// Set the summary text to the JTextArea
summaryTextArea.setText(summary.toString());

// Remove the Next button and adjust the window size
// Hide the "Next" button
nextButton.setVisible(false);
// Hide the "Back" button
backButton.setVisible(false);
// Hide the "Pause" button
pauseButton.setVisible(false);
// Hide the "50-50" button
fiftyFiftyButton.setVisible(false);
```

```
// Hide the "Ask the Computer" button
askFriendButton.setVisible(false);

// Add an "Exit" button
// Create a new button
JButton exitButton = new JButton("Exit");
// Action listener for the exit button
exitButton.addActionListener(new ActionListener()
{
   @Override
   public void actionPerformed(ActionEvent e)
   {
      // Exit the entire program
      System.exit(0);
   }
});

// Add a "New Game" button
// Create a new button
JButton newgameButton = new JButton("New Game");
// Action listener for the new game button
newgameButton.addActionListener(new ActionListener()
{
   @Override
   public void actionPerformed(ActionEvent e)
   {
      // Start a new game
      startNewGame();
   }
});

// Create a new panel for the summary and exit button
// Create a new panel
JPanel summaryPanel = new JPanel();
// Set the layout to BorderLayout
summaryPanel.setLayout(new BorderLayout());
// Add the summary text area to the panel
```

```
    summaryPanel.add(new JScrollPane(summaryTextArea), BorderLayout.CENTER);

    // Create a new panel for the exit button
    // Create a new panel
    JPanel buttonPanel = new JPanel();
    // Add the exit button to the panel
    buttonPanel.add(exitButton);
    // Add the new game button to the panel
    buttonPanel.add(newgameButton);

    // Add the panels to the summary frame
    // Create a new frame
    summaryFrame = new JFrame("Quiz Summary");
    // Set the size of the frame
     summaryFrame.setSize(900, 600);
     // Set the default close operation
     summaryFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     // Set the layout to BorderLayout
     summaryFrame.setLayout(new BorderLayout());
     // Add the summary panel to the frame
     summaryFrame.add(summaryPanel, BorderLayout.CENTER);
     // Add the button panel to the frame
     summaryFrame.add(buttonPanel, BorderLayout.SOUTH);
     // Set the location of the frame to the center of the screen
     summaryFrame.setLocationRelativeTo(null);
     // Show the frame
     summaryFrame.setVisible(true);
     // Set the caret position to the top of the text area
     summaryTextArea.setCaretPosition(0);

     // Close the quiz game window
      dispose();
}

// Method for showing the pause menu
private void showPauseMenu()
{
```

```
// If the game is paused
if (!paused)
{
   // Stop the timer
   questionTimer.stop();
   // Set paused to true
   paused = true;
   // Create the pause menu
   createPauseMenu();
   // Show the pause menu
   pauseMenu.show(pauseButton, 0, pauseButton.getHeight());
   //hide next button
   nextButton.setVisible(false);
   //hide back button
   backButton.setVisible(false);
   //hide fifty-fifty button
   fiftyFiftyButton.setVisible(false);
   //hide ask the computer button
   askFriendButton.setVisible(false);
   //hide question
   questionLabel.setVisible(false);
   // Loop 4 times
   for (int i = 0; i < options.length; i++)
   {
      //hide radio buttons
      options[i].setVisible(false);
   }

}
else
{
   // Set paused to false
   paused = false;
   // Hide the pause menu
   pauseMenu.setVisible(false);
   // Enable the next button
   nextButton.setEnabled(true);
```

```
        // Enable the back button
        backButton.setEnabled(true);
    }
}

// Method for creating the pause menu
private void createPauseMenu()
{
    // If the pause menu has not been created yet
    if (pauseMenu == null)
    {
        // Create a new popup menu
        pauseMenu = new JPopupMenu();

        // Create a new menu item for resuming the game
        JMenuItem resumeItem = new JMenuItem("Resume");

        // Action listener for the resume item
        resumeItem.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
                // Resume the game
                resumegame();
            }
        });

        // Create a new menu item for starting a new game
        JMenuItem newGameItem = new JMenuItem("New Game");

        // Action listener for the new game item
        newGameItem.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent e)
            {
```

```
        // Set paused to false
        paused = false;
        // Hide the pause menu
        pauseMenu.setVisible(false);
        // Enable the next button
        nextButton.setEnabled(true);
        // Enable the back button
        backButton.setEnabled(true);
        // Restart the game
        restartGame();
    }
});

// Create a new menu item for showing the credits
JMenuItem creditsItem = new JMenuItem("Credits");

// Action listener for the credits item
creditsItem.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Show the credits
        showCredits();
    }
});

// Create a new menu item for exiting the game
JMenuItem exitItem = new JMenuItem("Exit");

// Action listener for the exit item
exitItem.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Exit the entire program
```

```
            System.exit(0);
        }
    });

    // Add the resume item to the pause menu
    pauseMenu.add(resumeItem);
    // Add the new game item to the pause menu
    pauseMenu.add(newGameItem);
    // Add the credits item to the pause menu
    pauseMenu.add(creditsItem);
    // Add the exit item to the pause menu
    pauseMenu.add(exitItem);
  }
}

// Method for resuming the game
private void resumegame()
{
  // Set paused to false
  paused = false;
  // Hide the pause menu
  pauseMenu.setVisible(false);
  //show next button
  nextButton.setVisible(true);
  //show back button
  backButton.setVisible(true);
  //show fifty-fifty button
  fiftyFiftyButton.setVisible(true);
  //show ask the computer button
  askFriendButton.setVisible(true);
  // Start the timer
  questionTimer.start();
  //show question
  questionLabel.setVisible(true);
  //show radio buttons
  for (int i = 0; i < options.length; i++)// Loop 4 times
  {
```

```
            options[i].setVisible(true);//show radio buttons
        }
    }

    // Method for restarting the game
    private void restartGame()
    {
        // Stop the timer
        questionTimer.stop();

        // Reset the game state (e.g., score, currentQuestionIndex)
        score = 0;
        currentQuestionIndex = 0;

        // Show the startup frame to allow the user to choose new options
        startupFrame.setVisible(true);

        // Hide the current quiz frame
        setVisible(false);

        // Clear the set of timed out questions
        timedOutQuestions.clear();
        // Reset the time remaining for each question
        Arrays.fill(timeRemaining, timerSeconds);
        // Reset the user's answers for each question
        Arrays.fill(userAnswers, null);

        // Set paused to false
        paused = false;
        // Hide the pause menu
        pauseMenu.setVisible(false);
        //show next button
        nextButton.setVisible(true);
        //show back button
        backButton.setVisible(true);
        //show fifty-fifty button
        fiftyFiftyButton.setVisible(true);
```

```
        //show ask the computer button
        askFriendButton.setVisible(true);
        // Enable the 50-50 lifeline button
        fiftyFiftyButton.setEnabled(true);
        // Enable the Ask the computer lifeline button
        askFriendButton.setEnabled(true);
        //show the question
        questionLabel.setVisible(true);
        // Loop 4 times
        for (int i = 0; i < options.length; i++)
        {
            //show radio buttons
            options[i].setVisible(true);
        }
    }

    // Method for showing the credits
    private void showCredits()
    {
        //show credits
        JOptionPane.showMessageDialog(this, "Credits: \n Armaan Nakhuda B-02 \n  Sushant Navle B-05 \n Nishal Poojary B-
17 \n \n");
    }

    // Method for using the 50-50 lifeline
    private void useFiftyFiftyLifeline()
    {
        // Get the current question
        Question currentQuestion = selectedQuestions.get(currentQuestionIndex);
        // Get the correct answer
        String correctAnswer = currentQuestion.getCorrectAnswer();

         // Disable two incorrect options
         // Variable for counting the number of disabled options
            int disabledCount = 0;
            // Loop 4 times
            for (int i = 0; i < options.length; i++)
```

```
        {
            // If the option is incorrect
            if (!options[i].getText().equals(correctAnswer))
            {
                // Disable the option
                options[i].setEnabled(false);
                // Increment the disabled count
                disabledCount++;

                // If two options have been disabled
                if (disabledCount == 2)
                {
                    break;
                }
            }
        }


    // Disable the 50-50 lifeline button after using it
    //Disable for testing by putting // in front of the line
    fiftyFiftyButton.setEnabled(false);
}

// Method for using the Ask a Friend lifeline
private void useAskFriendLifeline()
{
    // Get the current question
    Question currentQuestion = selectedQuestions.get(currentQuestionIndex);

    // Generate a random number to simulate friend's response
    // 0 to 100
    int responsePercentage = new Random().nextInt(101);

    // Friend has an 80% chance of giving the correct answer
    // If the friend gives the correct answer
    if (responsePercentage <= 80)
    {
```

```
        // Select the correct answer
        String correctAnswer = currentQuestion.getCorrectAnswer();

        // Loop 4 times
        for (int i = 0; i < options.length; i++)
        {
            // If the option is the correct answer
            if (options[i].getText().equals(correctAnswer))
            {
                //select the correct answer
                options[i].setSelected(true);
                break;
            }
        }
    }
    else
    {
        // Find the index of the correct answer
        int correctIndex = findCorrectAnswerIndex(currentQuestion.getAnswerChoices());
        // Generate a random wrong index
        int wrongIndex = generateRandomWrongIndex(currentQuestion.getAnswerChoices().length, correctIndex);
        //select the wrong answer
        options[wrongIndex].setSelected(true);
    }

    // Disable the Ask a Friend lifeline button after using it
    //Disable for testing by putting // in front of the line
    askFriendButton.setEnabled(false);
}

//generate random wrong index
private int generateRandomWrongIndex(int totalOptions, int correctIndex)
{
    // Generate a random number
    int wrongIndex = new Random().nextInt(totalOptions);

    // While the random number is the same as the correct index
```

```
    while (wrongIndex == correctIndex)
    {
        // Generate a new random number
        wrongIndex = new Random().nextInt(totalOptions);
    }

    // If the wrong index is less than 0
    if (wrongIndex < 0)
    {
        // Set the wrong index to 0
        wrongIndex = 0;
    }
    // If the wrong index is greater than 3
    else if (wrongIndex > 3)
    {
        // Set the wrong index to 3
        wrongIndex = 3;
    }

    // Return the wrong index
    return wrongIndex;
}

//find the correct answer index
private int findCorrectAnswerIndex(String[] answerChoices)
{
    // Loop 4 times
    for (int i = 0; i < answerChoices.length; i++)
    {
        // If the answer choice is the correct answer
        if (answerChoices[i].equals(selectedQuestions.get(currentQuestionIndex).getCorrectAnswer()))
        {
            // Return the index
            return i;
        }
    }
```

```
    // Not found
    return -1;
}

// Method for handling the timeout
private void handleTimeout()
{
    // If the timer runs out, show a message to the user
    int choice = JOptionPane.showOptionDialog
    // Show OK and Cancel buttons
    (
     this,
     "Time's up! Click OK to move to the next question.",
     "Timeout",
     JOptionPane.OK_CANCEL_OPTION,
     JOptionPane.INFORMATION_MESSAGE,
     null,
     null,
     null
    );

        // Update and disable radio buttons (whether the user clicks OK or Cancel)
        disableRadioButtonsForTimedOutQuestion(currentQuestionIndex);

    // If the user clicks OK, move to the next question
    if (choice == JOptionPane.OK_OPTION)
    {
        // Increment the current question index
        currentQuestionIndex++;
        // If there are more questions remaining
        if (currentQuestionIndex < selectedQuestions.size())
        {
            // Load the next question
            loadQuestion(currentQuestionIndex);
            //load the previously selected answer by the user
            // Loop 4 times
            for (int i = 0; i < 4; i++)
```

```
        {
            // If the answer choice matches the user's answer
            if (options[i].getText().equals(userAnswers[currentQuestionIndex]))
            {
                //select the answer choice
                options[i].setSelected(true);
                break;
            }
        }
    }
    else
    {
        // Show the result
        showResult();
    }
  }

  // If the user clicks Cancel, do nothing (stay on the current question)
}

// Add a new method to disable radio buttons for the timed out question
private void disableRadioButtonsForTimedOutQuestion(int questionIndex)
{
  // Loop 4 times
  for (int i = 0; i < options.length; i++)
  {
    //disable radio buttons
    options[i].setEnabled(false);
  }
}

//method to show the instructions frame
private void showInstructionsFrame()
{
  // Create a new frame
  JFrame instructionsFrame = new JFrame("Instructions");
  // Set the size of the frame
```

```
instructionsFrame.setSize(800, 550);
// Set the default close operation
instructionsFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
// Set the location of the frame to the center of the screen
instructionsFrame.setLocationRelativeTo(null);

// Create a new text area
JTextArea instructionsTextArea = new JTextArea();
// Make the text area non-editable
instructionsTextArea.setEditable(false);

// Set the text for the text area
instructionsTextArea.setText
(   "Instructions: \n\n" +
    "1. Select the number of questions you want to answer from the drop-down menu.\n" +
    "2. Click the 'Start Quiz' button to begin the quiz.\n" +
    "3. Click the 'Next' button to move to the next question.\n" +
    "4. Click the 'Back' button to move to the previous question.\n" +
    "5. Click the 'Pause' button to pause the quiz and access the pause menu.\n" +
    "6. Click the '50-50' button to use the 50-50 lifeline.\n" +
    "7. Click the 'Ask the Computer' button to use the Ask a Friend lifeline.\n" +
    "8. You have 20 seconds to answer each question.\n" +
    "9. The timer will start as soon as the question is loaded.\n" +
    "10. Once the timer is complete the answer buttons will be disabled after which it wont be possible to answer the question/change your answer.\n" +
    "11. The timer will stop when you click the 'Next' button or when you run out of time.\n" +
    "12. Click the 'Exit' button to exit the quiz.\n\n" +
    "Note: You can also use the physical keyboard keys to interact with the quiz:\n\n" +
    "a) 1 to 4 number keys- option 1 to 4 for answers.\n "+
    "b) P-pause the quiz.\n"+
    "c) R-resume the quiz.\n"+
    "d) Enter-next question.\n"+
    "e) Back Space- previous question.\n"+
    "f) F-50-50 lifeline.\n"+
    "g) A-ask the computer lifeline.\n"+
    "h) E-exit the quiz.\n"+
    "i) I-Instructions.\n\n"+
```

```
            "Good luck!"
        );

        // Create a new button
        JButton closeButton = new JButton("Close");

        // Action listener for the close button
        closeButton.addActionListener(new ActionListener()
        {
            @Override
            // When the button is clicked
            public void actionPerformed(ActionEvent e)
            {
                // Close the instructions frame
                instructionsFrame.dispose();
            }
        });

        // Create a new panel
        JPanel buttonPanel = new JPanel();
        // Add the close button to the panel
        buttonPanel.add(closeButton);
        // Add the text area to the frame
        instructionsFrame.add(new JScrollPane(instructionsTextArea), BorderLayout.CENTER);
        // Add the panel to the frame
        instructionsFrame.add(buttonPanel, BorderLayout.SOUTH);
        // Show the frame
        instructionsFrame.setVisible(true);
    }

    // Method for starting a new game
    private void startNewGame()
    {
        // Stop the timer
        questionTimer.stop();

        // Reset the game state (e.g., score, currentQuestionIndex)
```

```
score = 0;
currentQuestionIndex = 0;

// Show the startup frame to allow the user to choose new options
startupFrame.setVisible(true);

// Hide the current quiz frame
setVisible(false);

// Clear the set of timed out questions
timedOutQuestions.clear();
// Reset the time remaining for each question
Arrays.fill(timeRemaining, timerSeconds);
// Reset the user's answers for each question
Arrays.fill(userAnswers, null);

// Set paused to false
paused = false;
//show pause button
pauseButton.setVisible(true);
//show next button
nextButton.setVisible(true);
//show back button
backButton.setVisible(true);
//show fifty-fifty button
fiftyFiftyButton.setVisible(true);
//show ask the computer button
askFriendButton.setVisible(true);
// Enable the 50-50 lifeline button
fiftyFiftyButton.setEnabled(true);
// Enable the Ask the computer lifeline button
askFriendButton.setEnabled(true);
//show the question
questionLabel.setVisible(true);
// Loop 4 times
for (int i = 0; i < options.length; i++)
{
```

```
        //show radio buttons
        options[i].setVisible(true);
    }

    //close the summary frame
    summaryFrame.setVisible(false);
}

//method to add key bindings
private void addKeyBindings()
{
    // Input map
    InputMap inputMap = this.getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
    // Action map
    ActionMap actionMap = this.getRootPane().getActionMap();

    // Key bindings
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_P, 0), "pause");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_1, 0), "answer1");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_2, 0), "answer2");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_3, 0), "answer3");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_4, 0), "answer4");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0), "next");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_BACK_SPACE, 0), "back");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_F, 0), "fiftyFifty");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_A, 0), "askFriend");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_S, 0), "startGame");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_I, 0), "openInstructions");
    inputMap.put(KeyStroke.getKeyStroke(KeyEvent.VK_E, 0), "exit");

    // Pause the game
    actionMap.put("pause", new AbstractAction()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            // Handle P key press (pause)
```

```
        pauseButton.doClick();
    }
});

// Select answer choices
actionMap.put("answer1", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle 1 key press (answer 1)
        options[0].doClick();
    }
});

// Select answer choices
actionMap.put("answer2", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle 2 key press (answer 2)
        options[1].doClick();
    }
});

// Select answer choices
actionMap.put("answer3", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle 3 key press (answer 3)
        options[2].doClick();
    }
});
```

```
// Select answer choices
actionMap.put("answer4", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle 4 key press (answer 4)
        options[3].doClick();
    }
});

// Move to the next question
actionMap.put("next", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle Enter key press (next question)
        nextButton.doClick();
    }
});

// Move to the previous question
actionMap.put("back", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle Backspace key press (previous question)
        backButton.doClick();
    }
});

// Use the 50-50 lifeline
actionMap.put("fiftyFifty", new AbstractAction()
{
    @Override
```

```
    public void actionPerformed(ActionEvent e)
    {
        // Handle F key press (50-50 lifeline)
        useFiftyFiftyLifeline();
    }
});

// Use the Ask the computer lifeline
actionMap.put("askFriend", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle A key press (Ask the computer lifeline)
        useAskFriendLifeline();
    }
});

// Start a new game
actionMap.put("startGame", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle S key press (start game)
        startNewGamekey();
    }
});

// Open the instructions
actionMap.put("openInstructions", new AbstractAction()
{
    @Override
    public void actionPerformed(ActionEvent e)
    {
        // Handle I key press (open instructions)
        showInstructionsFrame();
```

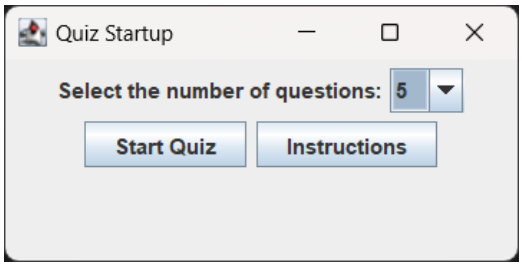
```
        }
    });

    actionMap.put("exit", new AbstractAction()
    {
        @Override
        public void actionPerformed(ActionEvent e)
        {
            // Handle E key press (exit)
            System.exit(0);
        }
    });
}

//method to start a new game
private void startNewGamekey()
{
    // Stop the timer
    questionTimer.stop();

    // Reset the game state (e.g., score, currentQuestionIndex)
    score = 0;
    currentQuestionIndex = 0;

    // Show the startup frame to allow the user to choose new options
    startupFrame.setVisible(true);

    // Hide the current quiz frame
    setVisible(false);

    // Clear the set of timed out questions
    timedOutQuestions.clear();
    // Reset the time remaining for each question
    Arrays.fill(timeRemaining, timerSeconds);
    // Reset the user's answers for each question
    Arrays.fill(userAnswers, null);
```

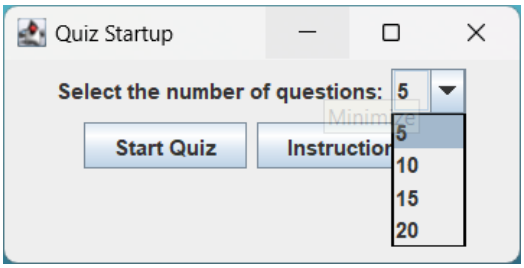
```
    // Set paused to false
    paused = false;
    //show pause button
    pauseButton.setVisible(true);
    //show next button
    nextButton.setVisible(true);
    //show back button
    backButton.setVisible(true);
    //show fifty-fifty button
    fiftyFiftyButton.setVisible(true);
    //show ask the computer button
    askFriendButton.setVisible(true);
    // Enable the 50-50 lifeline button
    fiftyFiftyButton.setEnabled(true);
    // Enable the Ask the computer lifeline button
    askFriendButton.setEnabled(true);

    //show the question
    questionLabel.setVisible(true);

    // Loop 4 times
    for (int i = 0; i < options.length; i++)
    {
        //show the radio buttons
        options[i].setVisible(true);
    }
}

// Main method
public static void main(String[] args)
{
    // Create a new thread
    SwingUtilities.invokeLater(new Runnable()
    {
        // Run the thread
        @Override
        public void run()
```

```
        {
            // Create a new QuizGameGUI object
            new QuizGameGUI();
        }
      });
   }
}

//class for question
class Question
{
   //question
   private String question;
   //answer choices
   private String[] answerChoices;
   //correct answer
   private String correctAnswer;

   // Constructor for the Question class
   public Question(String question, String... answerChoices)
   {
      //question
      this.question = question;
      //answer choices
      this.answerChoices = answerChoices;
      //correct answer
      this.correctAnswer = answerChoices[answerChoices.length - 1];
   }

   //method to get question
   public String getQuestion()
   {
      //return question
      return question;
   }

   //method to get answer choices
```

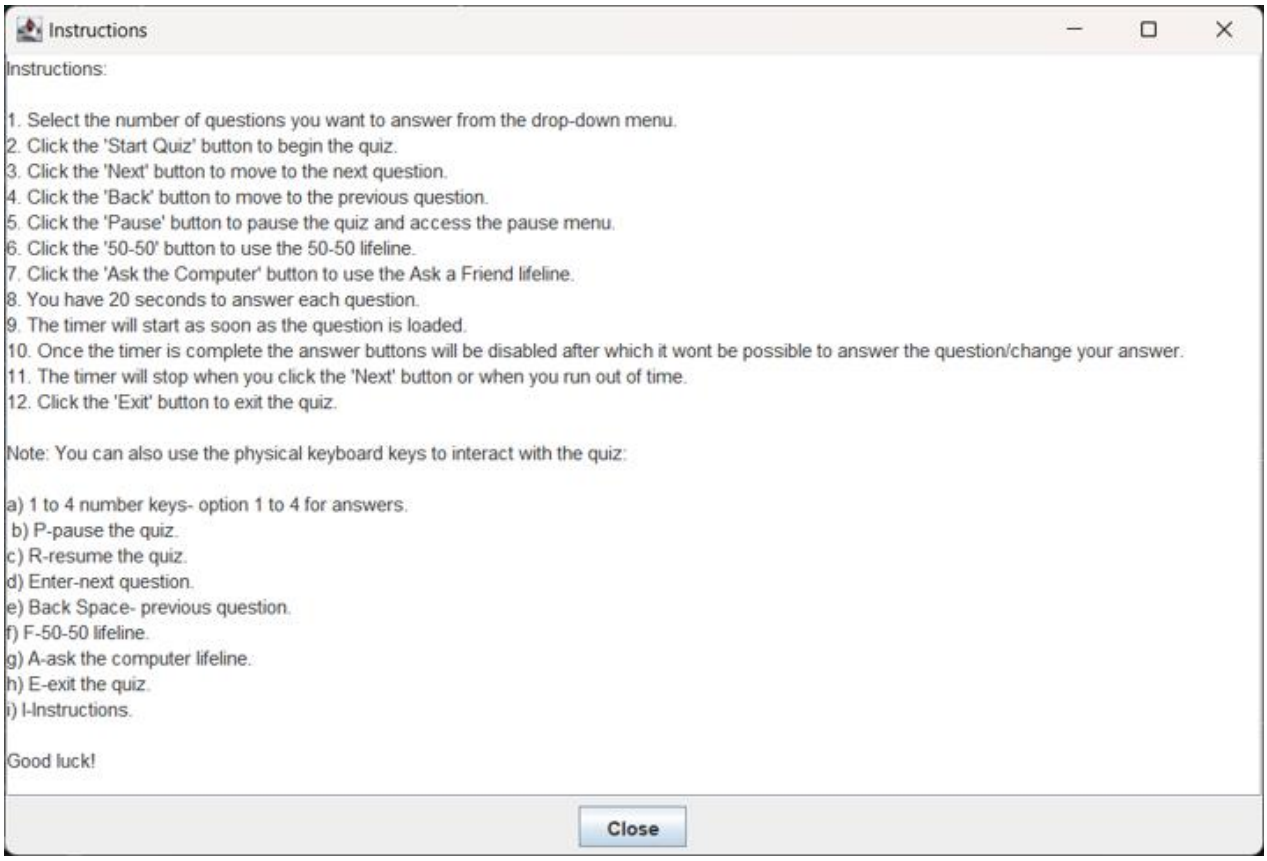
```
    public String[] getAnswerChoices()
    {
        //return answer choices
        return answerChoices;
    }

    //method to get correct answer
    public String getCorrectAnswer()
    {
        //return correct answer
        return correctAnswer;
    }
}
```

Screenshots Of The Project

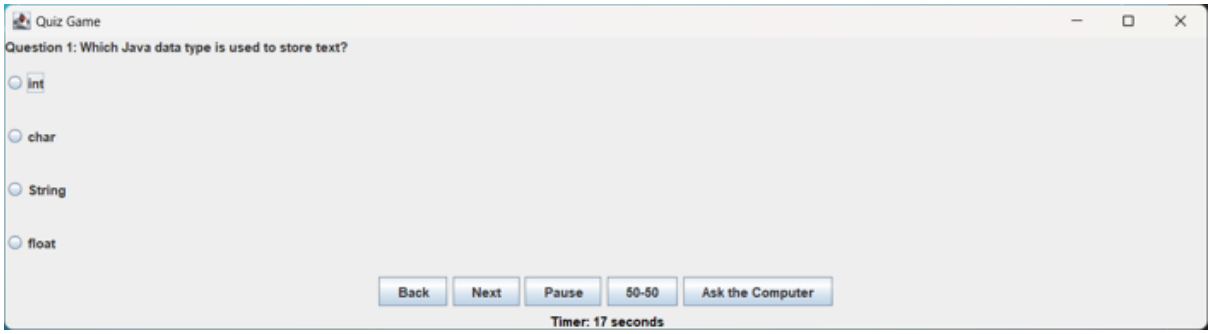




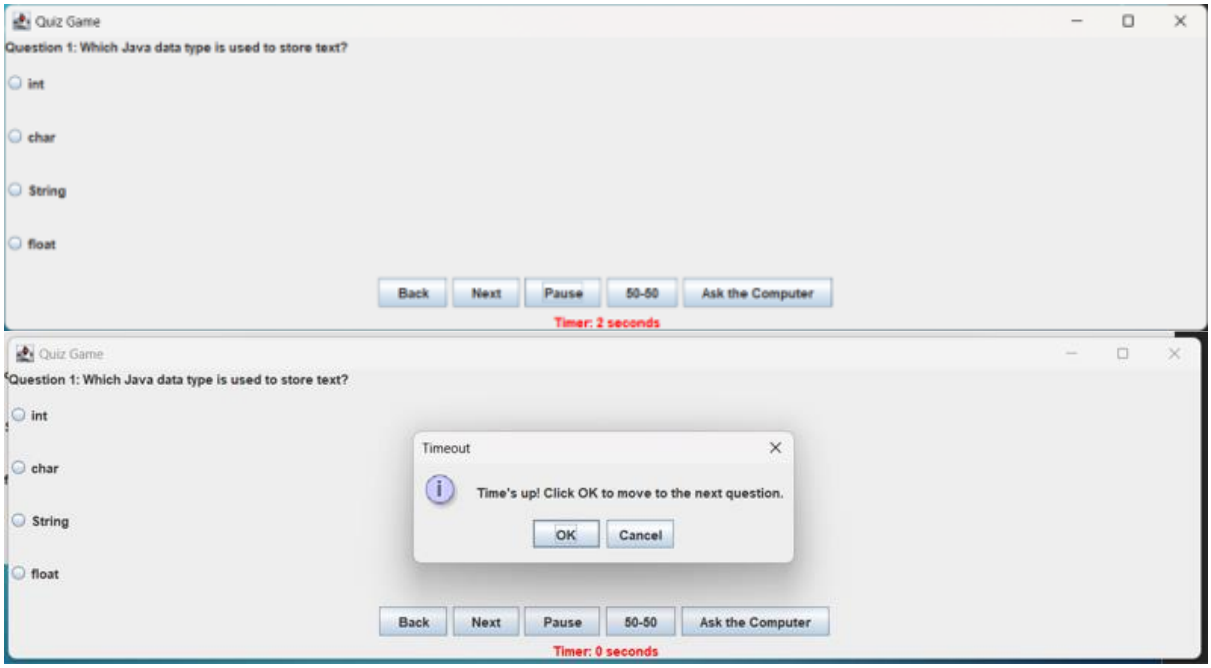
Startup Screen to give the user the option to choose how many questions they would like while also giving the instructions options they can see the full functionality of the app



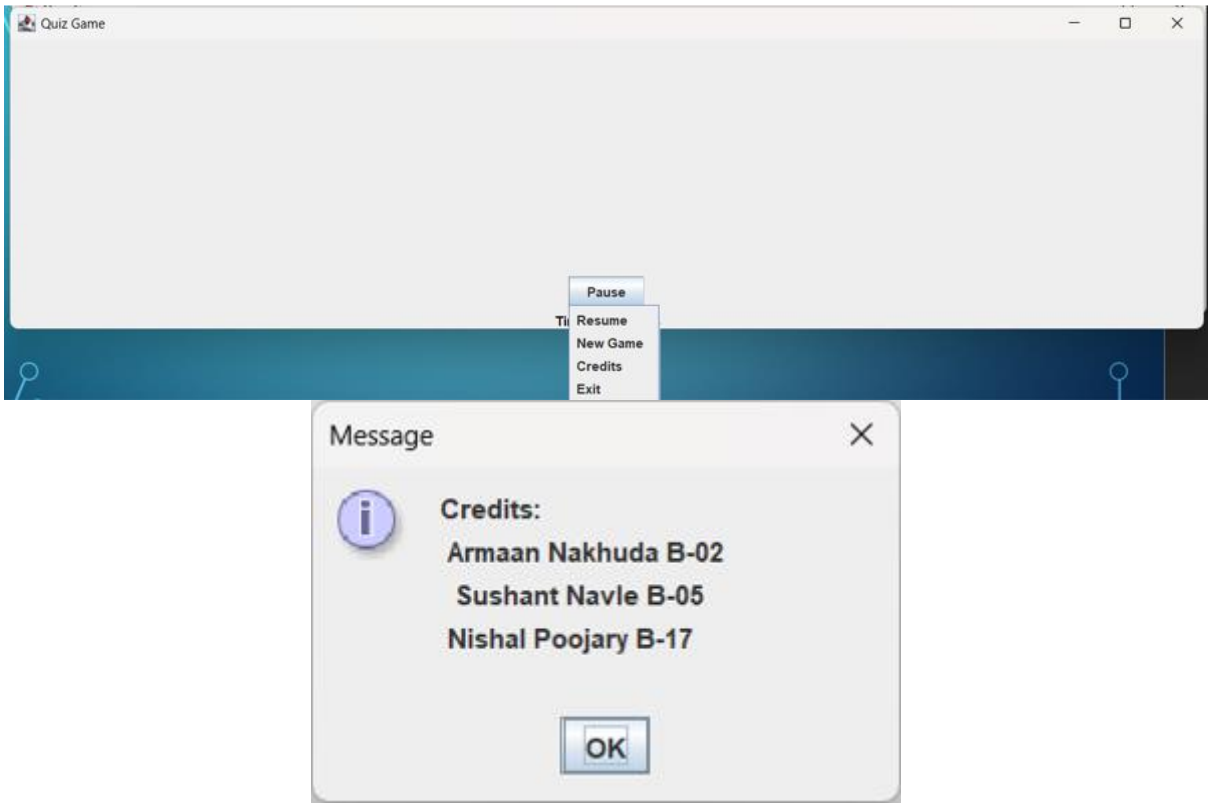
Instructions panel for the user to understand the functionality and a close button to return back to the quiz startup screen



The 1st question along with the game options after the user starts the quiz



As the timer is heading towards 0 the timer flashes red so warn the user after which a time's up pop up comes up and disables the radio buttons so the user cannot change or add their answer.



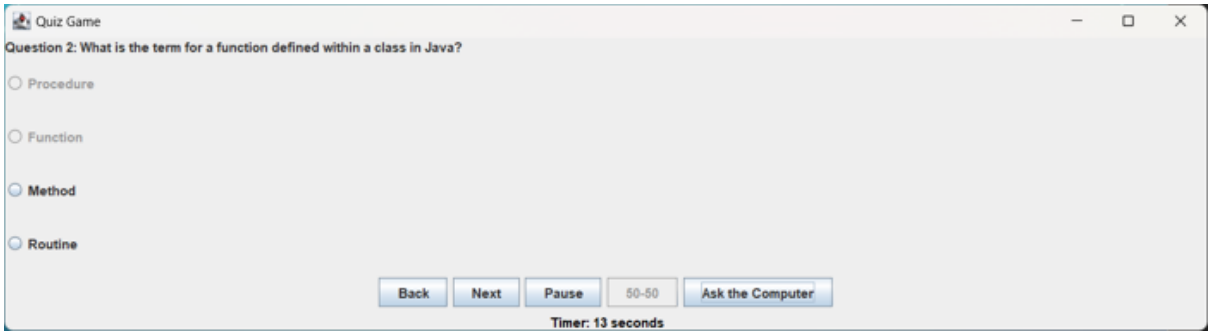
The popup menu the user gets after clicking the pause button which also hides the questions and options to reduce cheating and also pauses the timer.

The resume option resumes the quiz from the point it was paused.

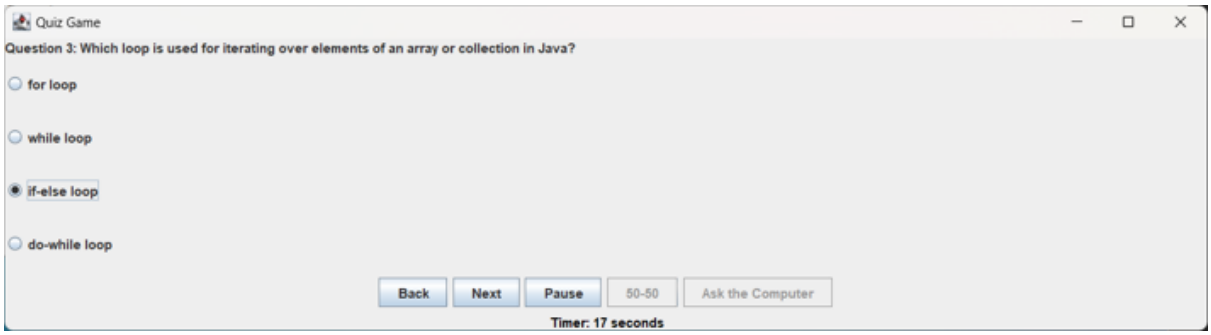
The new game option takes the user back to the quiz startup screen and resets everything.

The Credits button brings a separate popup to show the names of the team members.

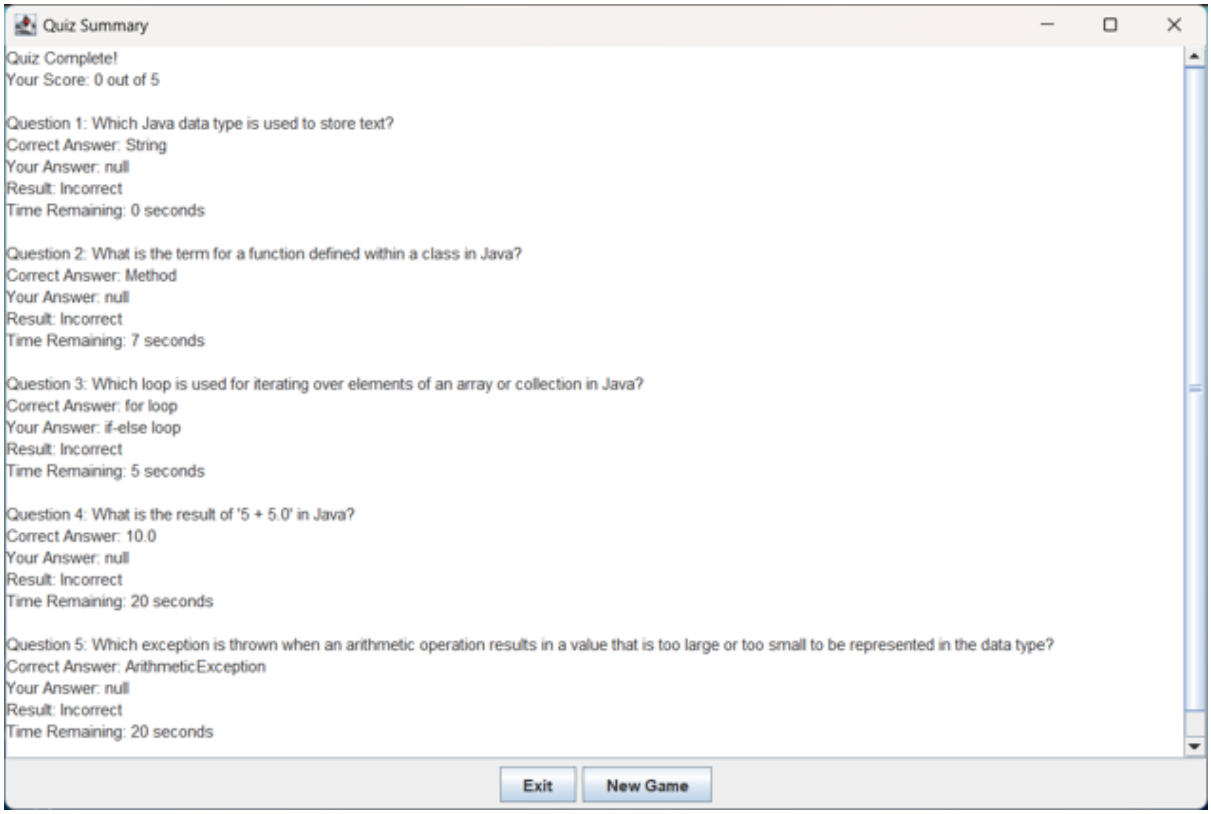
The exit button quits the whole quiz.



The 50-50 lifeline button disables 2 random wrong options giving the user only 2 options to choose from, this is an one time use button after which it will be disabled for rest of the quiz.



The Ask the computer Lifeline has a 80% chance of giving the user the right answer and a 20% chance of the wrong answer, this is also an one time use button after which it will be disabled for rest of the quiz.



The Summary/Results screen which shows the overall right answers, time taken by the user for each question, the option chosen by the user, the current answer and if the user choose the right answer or not.

Conclusion

The Quiz Application is an innovative educational tool with a user-friendly interface and interactive features, offering a dynamic learning experience. Its future scope includes AI personalization, mobile support, collaborative learning, integration with learning management systems, advanced analytics, global expansion, and enhanced accessibility features. These developments aim to make the application more engaging, inclusive, and adaptable to evolving educational needs, ensuring its continued growth and impact in the field of education.

Future scope

1. AI-Powered Personalization: Utilize artificial intelligence to tailor quiz content to individual user preferences and skill levels, enhancing the learning experience.
2. Mobile and Offline Modes: Develop mobile applications for convenient access to quizzes on various devices, with offline capabilities to increase accessibility.
3. Collaborative Learning: Expand community-driven features to enable collaborative quiz creation and group competitions, promoting knowledge sharing and teamwork.
4. Integration with Learning Management Systems: Partner with educational institutions to integrate the Quiz Application into their systems, enhancing the educational experience for students.
5. Advanced Analytics: Improve performance analytics to provide deeper insights into user strengths and weaknesses, offering personalized recommendations for improvement.

These future developments aim to make the Quiz Application more engaging, inclusive, and adaptable to evolving educational needs, solidifying its role as a valuable tool in modern education.

Reference

1. Used Chatgpt to give a better Grammatical format for the wording.
2. Used javapoint's website and knowledge for learning about some concepts of java to implement the particular system.
3. Used Chatgpt for the extensive list of questions used in the project.

Github Repository Link for progress and commit history related to this project:

https://github.com/Armaan4477/Quiz-Game/