

**NAZEER HUSSAIN UNIVERSITY**

For partial fulfillment of the requirement for the award

Of

**COMPUTER SCIENCE AND ENGINEERING**

On the working of project

QUIZ APPLICATION SYSTEM

**SUBMITTED BY THE TEAM OF**

**IRAJ ANSARI (201-S23-034)**

**IQRA KHAN (201-S23-008)**

**UMAR RAEES (201-S23-024)**

**UZAIR SHEIKH (201-S23-019)**

Under the guidance of

**MISS SUBREENA AYUB**

**SUBJECT: DATA STRUCTURE AND ALGORITHM**

**DECLARATION**

*We hereby declare that the details enclosed in the project proposal entitled* ***“QUIZ APPLICATION SYSTEM”*** *carried out by us in partial fulfilment of the requirements for the award for degree in computer science & engineering are true and correct to the best of our knowledge and belief is a record of an original work done by us under the guidance of* ***MISS SUBREENA AYUB KHAN,*** *faculty of computer science and engineering department and it is also declare that this report has been prepared for academic purpose alone and has not been/will not be submitted elsewhere for any other purpose.*

***Place: Nazeer Hussain University***

***Branch-Computer Science & Engineering***

***Semester-3rd***

***Team Names- IRAJ ANSARI-(201-S23-034)***

***IQRA KHAN-(201-S23-008)***

***UMAR RAEES-(201-S23-024)***

***UZAIR SHEIKH-(201-S23-019)***

**ACKNOLEDGEMENT**

*We are thankful to the Department because of whom, we have gained confidence in “Quiz Application System” Development and it has also enhanced our professional skills so as to become competent in this field.*

*We take this opportunity to present my votes of thanks to all those guide post who really acted as lightening pillars to enlighten our way throughout this project that has led to successful and satisfactory completion of this study.*

*We are really grateful to our Ma’am Subreena for providing us with an opportunity to undertake this project in this university and providing us with all the facilities.*

*We are highly thankful for your active support, valuable time and advice, whole-hearted guidance, sincere cooperation and painstaking involvement during the study and in completing the assignment of preparing the said project within the time stipulated.*

*Lastly, we are thankful to all those, particularly the various friends, who have been instrumental in creating proper, healthy and conductive environment and including new and fresh innovative ideas for us during the project, their help, it would have been extremely difficult for us to prepare the project in a time bound framework.*

*We started our this project with excitement of embarking on a new and long journey. Advancing with leaps and bounds every day, and exploring new avenues that the field of Quiz Application System. The journey has not been a smooth one though. Initially, it was very hard to cope up, learn and get acquainted with this project “Quiz Application System”, it would not have been possible to sustain, strive and get enriched in this field.*

**ABSTRACT**

*The Quiz Application System is designed to facilitate the management of quiz operations, including user authentication, order management, and customer interactions. The system is implemented using Java Swing for the graphical user interface and incorporates various data structures and algorithms for efficient data handling. The primary objective of this project is to create a robust and user-friendly application that streamlines restaurant operations, enhances user experience, and ensures secure access through proper authentication mechanisms.*

*Technologies Used:*

*Programming Language: Java*

*Graphical User Interface: Java Swing*

*Database Management: DAO (Data Access Object) pattern for data handling*

*Libraries: Javax.swing, java.awt, ImageIcon*

*Project Structure:*

*Login.java: Handles user login and authentication.*

*Home.java: Main dashboard after successful login.*

*Signup.java: New user registration.*

*ForgotPassword.java: Password recovery for users.*

*UserDao.java: Data Access Object for user-related database operations.*

*User.java: Model class for user data.*

**LIST OF FIGURES / TABLES**

|  |  |
| --- | --- |
| **FIGURES NO** | **PAGE NUMBER** |
| 6.1 | 39 |
| 6.2 | 39 |
| 6.3 | 40 |
| 6.4 | 40 |

**TABLE OF CONTENT**

DECLARATION…………………………………………………………………………………………………………………………..2

ACKNOLEGMENT……………………………………………………………………………………………………………………….3

ABSTRACT………………………………………………………………………………………………………………………………….4

LIST OF FIGURES…………………………………………………………………………………………………………………………5

CHAPTER 1 (INTRODUCTION)………….………………………………………………………………………………………….7

CHAPTER 2 (SCREEN 1: LOG IN)………………………………………………………………………………………………….11

CHAPTER 3 (SCREEN 2: RULE PAGE)……………………………………………………………………………………………15

CHAPTER 4 (SCREEN 3: QUIZ PAGE)…………………………………………………………………………………………...20

CHAPTER 5 (SCREEN 4: SCORE PAGE)………………………………………………………………………………………….34

CHAPTER 6 (IMPLEMENTATION)………………………………………………………………………………………………….39

CHAPTER 7 (SYSTEM TESTING)……………………………………………………………………………………………………..41

CHAPTER 8 (SYSTEM DESIGN)………………………………………………………………………………………………………45

CHAPTER 9 (CONCLUSION)…………………………………………………………………………………………………………..46

CHAPTER 10 (REFERENCE)…………………………………………………………………………………………………………….47

**CHAPTER 1**

**INTRODUCTION:**

**1.1 Project Overview**

The Quiz Application project involves developing an interactive software program using Java that allows users to participate in quizzes on various topics. This application is designed to offer a user-friendly interface where users can register, log in, select quizzes, answer multiple-choice questions, and receive immediate feedback on their performance. The application aims to serve both educational and entertainment purposes, catering to users of all ages who seek to test their knowledge or learn new information.

**1.2 Objectives**

The main objectives of the quiz application project are as follows:

- **Develop a Robust Application:** Create a reliable and efficient quiz application using Java, ensuring smooth performance and minimal bugs.

- **User Management:** Implement features that allow user registration, login, and profile management, ensuring secure and personalized user experiences.

- **Quiz Management**: Enable administrators to create, update, and delete quizzes and questions, providing flexibility in content management.

- **Interactive User Interface:** Design an intuitive and interactive GUI using Java Swing, making it easy for users to navigate and use the application.

- **Scoring and Feedback:** Implement a scoring system that provides immediate feedback to users upon quiz completion, enhancing the learning experience.

**1.3 Scope**

The scope of the quiz application project includes the following:

- **User Registration and Authentication**: Users will be able to register and log in to the application using secure authentication mechanisms.

- **Quiz Creation and Management:** Administrators will have the ability to create new quizzes, add or update questions, and manage quiz content through a dedicated interface.

- **Quiz Participation:** Registered users will be able to select quizzes, answer questions, and submit their responses. The application will track their scores and display results.

- **Database Integration:** The application will use a relational database (e.g., MySQL) to store user information, quiz data, questions, and user responses, ensuring data persistence and integrity.

- **Extensibility:** The application will be designed with future enhancements in mind, allowing for the addition of new features such as different question types, timed quizzes, and social sharing capabilities.

**1.4 Tools and Technologies**

The development of the quiz application will utilize the following tools and technologies:

- **Programming Language:** Java (JDK 8 or higher)

- **Integrated Development Environment (IDE):** IntelliJ IDEA or Eclipse

- **Frameworks/Libraries:** Java Swing for building the graphical user interface, JDBC for database connectivity

- **Version Control:** Git for source code management and collaboration

**1.5 Methodology**

The project will follow a structured methodology to ensure systematic development and timely completion:

- **Requirement Gathering:** Identifying and documenting the functional and non-functional requirements of the application.

- **Design:** Creating architectural, use case, class, and database diagrams to visualize the system design.

- **Implementation:** Writing the code for the application, adhering to best practices and coding standards.

- **Testing:** Conducting unit, integration, and system testing to ensure the application meets the specified requirements and functions correctly.

- **Deployment:** Deploying the application in a suitable environment and providing user documentation and training.

**1.6 Significance**

This project holds significance in the following ways:

- **Educational Value:** Provides a platform for students and learners to test and expand their knowledge in various subjects.

- **Technical Experience:** Offers developers an opportunity to enhance their skills in Java programming, GUI design, and database management.

**- Flexibility and Adaptability:** The application can be adapted for different purposes, such as corporate training, competitive exams, and recreational quizzes.

By achieving these objectives and adhering to the defined scope, the quiz application aims to provide a valuable tool for learning and knowledge assessment, while also demonstrating effective software development practices in Java.

**CHAPTER 2**

**SCREEN 1 (LOGIN MODULE):**

The login screen is the first interface that users encounter. It includes fields for email and password, and buttons for login, clear, exit, forgot password, and signup

**2.1 Key Component**

* `txtEmail` and `txtPassword`: Text fields for user input.
* `btnLogin`, `btnClear`, `btnExit`, `jButton4` `(Forgot Password)`, `jButton5` (Signup): Action buttons for respective functionalities.

**2.2 Code Snippet**

package quiz.application;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Login extends JFrame implements ActionListener{

JButton rules, back;

JTextField tfname;

Login() {

getContentPane().setBackground(Color.WHITE);

setLayout(null);

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icons/login.jpeg"));

JLabel image = new JLabel(i1);

image.setBounds(0, 0, 600, 500);

add(image);

JLabel heading = new JLabel("Simple Minds");

heading.setBounds(750, 60, 300, 45);

heading.setFont(new Font("Viner Hand ITC", Font.BOLD, 40));

heading.setForeground(new Color(30, 144, 254));

add(heading);

JLabel name = new JLabel("Enter your name");

name.setBounds(810, 150, 300, 20);

name.setFont(new Font("Mongolian Baiti", Font.BOLD, 18));

name.setForeground(new Color(30, 144, 254));

add(name);

tfname = new JTextField();

tfname.setBounds(735, 200, 300, 25);

tfname.setFont(new Font("Times New Roman", Font.BOLD, 20));

add(tfname);

rules = new JButton("Rules");

rules.setBounds(735, 270, 120, 25);

rules.setBackground(new Color(30, 144, 254));

rules.setForeground(Color.WHITE);

rules.addActionListener(this);

add(rules);

back = new JButton("Back");

back.setBounds(915, 270, 120, 25);

back.setBackground(new Color(30, 144, 254));

back.setForeground(Color.WHITE);

back.addActionListener(this);

add(back);

setSize(1200, 500);

setLocation(200, 150);

setVisible(true);

}

public void actionPerformed(ActionEvent ae) {

if (ae.getSource() == rules) {

String name = tfname.getText();

setVisible(false);

new Rules(name);

} else if (ae.getSource() == back) {

setVisible(false);

}

}

public static void main(String[] args) {

new Login();

}

}

**2.3 Challenges faced**

* Email Validation: Ensuring the email format is correct.
* Dynamic UI Updates: Managing the state of the login button based on user input.
* User Feedback: Providing clear and concise feedback messages to the user.

**2.4 Conclusion**

This project provided valuable experience in developing a full-fledged desktop application using Java Swing. It also reinforced the importance of user input validation and secure authentication practices. Through this project, I learned how to effectively manage UI components and handle events to create an interactive and user-friendly application.

**CHAPTER 3**

**SCREEN 2 (RULE PAGE):**

To create a rule page for a quiz application in Java, you can use Swing to build the graphical user interface. Below is a step-by-step guide to create a rule page with the essential components: heading, rule list, acknowledgment checkbox, proceed button, and back button.

**3.1 Step-by-Step Implementation**

**1. Set Up the Main Frame:**

- Create a main frame (window) for the application

- Set the layout and basic properties.

**2. Add a Heading:**

- Use a `JLabel` to display the heading of the rule page.

**3. Display Rules:**

- Use a `JTextArea` to list the rules.

- Place the `JTextArea` inside a `JScrollPane` to handle overflow if the content is long.

**4. Acknowledgment Checkbox:**

- Use a `JCheckBox` to allow users to acknowledge they have read the rules.

- The proceed button should only be enabled when this checkbox is selected.

**5. Proceed and Back Buttons:**

- Use `JButton` for the proceed and back buttons.

- Add action listeners to handle button clicks.

**6. Layout Management:**

- Use `JPanel` and layout managers like `BorderLayout` and `BoxLayout` to organize the components.

**3.2 Code snippet**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class RulePage {

public static void main(String[] args) {

// Create the frame

JFrame frame = new JFrame("Quiz Application");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setSize(500, 600);

// Create the heading

JLabel heading = new JLabel("Quiz Rules", JLabel.CENTER);

heading.setFont(new Font("Arial", Font.BOLD, 24));

heading.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0)); // Add some padding

// Create the rules text area

JTextArea rulesTextArea = new JTextArea();

rulesTextArea.setText("1. Each quiz consists of 10 questions.\n" +

"2. You have 15 minutes to complete the quiz.\n" +

"3. Each question carries one point.\n" +

"4. No negative marking for wrong answers.\n" +

"5. Do not use any external resources.\n" +

"6. Once submitted, you cannot retake the quiz.\n");

rulesTextArea.setEditable(false);

rulesTextArea.setFont(new Font("Arial", Font.PLAIN, 16));

rulesTextArea.setLineWrap(true);

rulesTextArea.setWrapStyleWord(true);

// Add the rules text area to a scroll pane

JScrollPane scrollPane = new JScrollPane(rulesTextArea);

scrollPane.setBorder(BorderFactory.createEmptyBorder(0, 20, 20, 20)); // Add some padding

// Create the acknowledgment checkbox

JCheckBox acknowledgeCheckbox = new JCheckBox("I have read and understood the rules.");

// Create the proceed button

JButton proceedButton = new JButton("Start Quiz");

proceedButton.setEnabled(false);

acknowledgeCheckbox.addActionListener(e -> proceedButton.setEnabled(acknowledgeCheckbox.isSelected()));

// Create the back button

JButton backButton = new JButton("Back");

backButton.addActionListener(e -> {

// Logic to navigate back to the previous screen

frame.dispose(); // Close the current window (for simplicity)

});

// Organize components using panels and layout managers

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.add(heading, BorderLayout.NORTH);

mainPanel.add(scrollPane, BorderLayout.CENTER);

JPanel bottomPanel = new JPanel();

bottomPanel.setLayout(new BoxLayout(bottomPanel, BoxLayout.Y\_AXIS));

bottomPanel.add(acknowledgeCheckbox);

bottomPanel.add(Box.createRigidArea(new Dimension(0, 10))); // Add some vertical space

bottomPanel.add(proceedButton);

bottomPanel.add(Box.createRigidArea(new Dimension(0, 10))); // Add some vertical space

bottomPanel.add(backButton);

bottomPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Add some padding

mainPanel.add(bottomPanel, BorderLayout.SOUTH);

// Add the main panel to the frame

frame.add(mainPanel);

frame.setVisible(true);

}

}

```

**3.3 Explanation**

**- Main Frame:** The `JFrame` is set up with a title and size. It uses `BorderLayout` to manage the layout.

**- Heading:** A `JLabel` is used for the heading, with center alignment and a bold font.

**- Rules TextArea:** A `JTextArea` displays the rules, wrapped in a `JScrollPane` to handle scrolling.

**- Acknowledgment Checkbox**: A `JCheckBox` is added, and an `ActionListener` is used to enable the proceed button when checked.

**- Proceed and Back Buttons:** `JButton` components for proceeding to the quiz and going back. Action listeners handle button clicks.

**- Layout Management:** `JPanel` with `BorderLayout` and `BoxLayout` organizes the components. Padding and spacing improve the layout.

**3.4 Conclusion**

This code provides a functional rule page for your quiz application, ensuring users understand the rules before starting the quiz.

**CHAPTER 4**

**SCREEN 3 (QUIZ PAGE)**

**4.1 Code snippet**

package quiz.application;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

public class Quiz extends JFrame implements ActionListener {

String questions[][] = new String[10][5];

String answers[][] = new String[10][2];

String useranswers[][] = new String[10][1];

JLabel qno, question;

JRadioButton opt1, opt2, opt3, opt4;

ButtonGroup groupoptions;

JButton next, submit, lifeline;

public static int timer = 15;

public static int ans\_given = 0;

public static int count = 0;

public static int score = 0;

String name;

Quiz(String name) {

this.name = name;

setBounds(50, 0, 1440, 850);

getContentPane().setBackground(Color.WHITE);

setLayout(null);

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icons/quiz.jpg"));

JLabel image = new JLabel(i1);

image.setBounds(0, 0, 1440, 392);

add(image);

qno = new JLabel();

qno.setBounds(100, 450, 50, 30);

qno.setFont(new Font("Tahoma", Font.PLAIN, 24));

add(qno);

question = new JLabel();

question.setBounds(150, 450, 900, 30);

question.setFont(new Font("Tahoma", Font.PLAIN, 24));

add(question);

questions[0][0] = "Which is used to find and fix bugs in the Java programs.?";

questions[0][1] = "JVM";

questions[0][2] = "JDB";

questions[0][3] = "JDK";

questions[0][4] = "JRE";

questions[1][0] = "What is the return type of the hashCode() method in the Object class?";

questions[1][1] = "int";

questions[1][2] = "Object";

questions[1][3] = "long";

questions[1][4] = "void";

questions[2][0] = "Which package contains the Random class?";

questions[2][1] = "java.util package";

questions[2][2] = "java.lang package";

questions[2][3] = "java.awt package";

questions[2][4] = "java.io package";

questions[3][0] = "An interface with no fields or methods is known as?";

questions[3][1] = "Runnable Interface";

questions[3][2] = "Abstract Interface";

questions[3][3] = "Marker Interface";

questions[3][4] = "CharSequence Interface";

questions[4][0] = "In which memory a String is stored, when we create a string using new operator?";

questions[4][1] = "Stack";

questions[4][2] = "String memory";

questions[4][3] = "Random storage space";

questions[4][4] = "Heap memory";

questions[5][0] = "Which of the following is a marker interface?";

questions[5][1] = "Runnable interface";

questions[5][2] = "Remote interface";

questions[5][3] = "Readable interface";

questions[5][4] = "Result interface";

questions[6][0] = "Which keyword is used for accessing the features of a package?";

questions[6][1] = "import";

questions[6][2] = "package";

questions[6][3] = "extends";

questions[6][4] = "export";

questions[7][0] = "In java, jar stands for?";

questions[7][1] = "Java Archive Runner";

questions[7][2] = "Java Archive";

questions[7][3] = "Java Application Resource";

questions[7][4] = "Java Application Runner";

questions[8][0] = "Which of the following is a mutable class in java?";

questions[8][1] = "java.lang.StringBuilder";

questions[8][2] = "java.lang.Short";

questions[8][3] = "java.lang.Byte";

questions[8][4] = "java.lang.String";

questions[9][0] = "Which of the following option leads to the portability and security of Java?";

questions[9][1] = "Bytecode is executed by JVM";

questions[9][2] = "The applet makes the Java code secure and portable";

questions[9][3] = "Use of exception handling";

questions[9][4] = "Dynamic binding between objects";

answers[0][1] = "JDB";

answers[1][1] = "int";

answers[2][1] = "java.util package";

answers[3][1] = "Marker Interface";

answers[4][1] = "Heap memory";

answers[5][1] = "Remote interface";

answers[6][1] = "import";

answers[7][1] = "Java Archive";

answers[8][1] = "java.lang.StringBuilder";

answers[9][1] = "Bytecode is executed by JVM";

opt1 = new JRadioButton();

opt1.setBounds(170, 520, 700, 30);

opt1.setBackground(Color.WHITE);

opt1.setFont(new Font("Dialog", Font.PLAIN, 20));

add(opt1);

opt2 = new JRadioButton();

opt2.setBounds(170, 560, 700, 30);

opt2.setBackground(Color.WHITE);

opt2.setFont(new Font("Dialog", Font.PLAIN, 20));

add(opt2);

opt3 = new JRadioButton();

opt3.setBounds(170, 600, 700, 30);

opt3.setBackground(Color.WHITE);

opt3.setFont(new Font("Dialog", Font.PLAIN, 20));

add(opt3);

opt4 = new JRadioButton();

opt4.setBounds(170, 640, 700, 30);

opt4.setBackground(Color.WHITE);

opt4.setFont(new Font("Dialog", Font.PLAIN, 20));

add(opt4);

groupoptions = new ButtonGroup();

groupoptions.add(opt1);

groupoptions.add(opt2);

groupoptions.add(opt3);

groupoptions.add(opt4);

next = new JButton("Next");

next.setBounds(1100, 550, 200, 40);

next.setFont(new Font("Tahoma", Font.PLAIN, 22));

next.setBackground(new Color(30, 144, 255));

next.setForeground(Color.WHITE);

next.addActionListener(this);

add(next);

lifeline = new JButton("50-50 Lifeline");

lifeline.setBounds(1100, 630, 200, 40);

lifeline.setFont(new Font("Tahoma", Font.PLAIN, 22));

lifeline.setBackground(new Color(30, 144, 255));

lifeline.setForeground(Color.WHITE);

lifeline.addActionListener(this);

add(lifeline);

submit = new JButton("Submit");

submit.setBounds(1100, 710, 200, 40);

submit.setFont(new Font("Tahoma", Font.PLAIN, 22));

submit.setBackground(new Color(30, 144, 255));

submit.setForeground(Color.WHITE);

submit.addActionListener(this);

submit.setEnabled(false);

add(submit);

start(count);

setVisible(true);

}

public void actionPerformed(ActionEvent ae) {

if (ae.getSource() == next) {

repaint();

opt1.setEnabled(true);

opt2.setEnabled(true);

opt3.setEnabled(true);

opt4.setEnabled(true);

ans\_given = 1;

if (groupoptions.getSelection() == null) {

useranswers[count][0] = "";

} else {

useranswers[count][0] = groupoptions.getSelection().getActionCommand();

}

if (count == 8) {

next.setEnabled(false);

submit.setEnabled(true);

}

count++;

start(count);

} else if (ae.getSource() == lifeline) {

if (count == 2 || count == 4 || count == 6 || count == 8 || count == 9) {

opt2.setEnabled(false);

opt3.setEnabled(false);

} else {

opt1.setEnabled(false);

opt4.setEnabled(false);

}

lifeline.setEnabled(false);

} else if (ae.getSource() == submit) {

ans\_given = 1;

if (groupoptions.getSelection() == null) {

useranswers[count][0] = "";

} else {

useranswers[count][0] = groupoptions.getSelection().getActionCommand();

}

for (int i = 0; i < useranswers.length; i++) {

if (useranswers[i][0].equals(answers[i][1])) {

score += 10;

} else {

score += 0;

}

}

setVisible(false);

new Score(name, score);

}

}

public void paint(Graphics g) {

super.paint(g);

String time = "Time left - " + timer + " seconds"; // 15

g.setColor(Color.RED);

g.setFont(new Font("Tahoma", Font.BOLD, 25));

if (timer > 0) {

g.drawString(time, 1100, 500);

} else {

g.drawString("Times up!!", 1100, 500);

}

timer--; // 14

try {

Thread.sleep(1000);

repaint();

} catch (Exception e) {

e.printStackTrace();

}

if (ans\_given == 1) {

ans\_given = 0;

timer = 15;

} else if (timer < 0) {

timer = 15;

opt1.setEnabled(true);

opt2.setEnabled(true);

opt3.setEnabled(true);

opt4.setEnabled(true);

if (count == 8) {

next.setEnabled(false);

submit.setEnabled(true);

}

if (count == 9) { // submit button

if (groupoptions.getSelection() == null) {

useranswers[count][0] = "";

} else {

useranswers[count][0] = groupoptions.getSelection().getActionCommand();

}

for (int i = 0; i < useranswers.length; i++) {

if (useranswers[i][0].equals(answers[i][1])) {

score += 10;

} else {

score += 0;

}

}

setVisible(false);

new Score(name, score);

} else { // next button

if (groupoptions.getSelection() == null) {

useranswers[count][0] = "";

} else {

useranswers[count][0] = groupoptions.getSelection().getActionCommand();

}

count++; // 0 // 1

start(count);

}

}

}

public void start(int count) {

qno.setText("" + (count + 1) + ". ");

question.setText(questions[count][0]);

opt1.setText(questions[count][1]);

opt1.setActionCommand(questions[count][1]);

opt2.setText(questions[count][2]);

opt2.setActionCommand(questions[count][2]);

opt3.setText(questions[count][3]);

opt3.setActionCommand(questions[count][3]);

opt4.setText(questions[count][4]);

opt4.setActionCommand(questions[count][4]);

groupoptions.clearSelection();

}

public static void main(String[] args) {

new Quiz("User");

}

}

**4.2 Explanation**

Here is a detailed explanation of the Java code for the quiz application in points:

**- Overview**

- The code implements a quiz application using Java Swing for the graphical user interface.

- It involves creating a window that displays quiz questions, handles user input, and calculates the final score.

**- Components**

1. Class and Constructor

- Class `Quiz`: Inherits from `JFrame` and implements `ActionListener` to handle button events.

- Constructor `Quiz(String name)`: Initializes the quiz window, sets up the layout, and loads questions and answers.

2. Instance Variables

- `questions`, `answers`, `useranswers`: Arrays to store questions, correct answers, and user responses.

- `qno`, `question`: `JLabel` components to display the question number and question text.

- `opt1`, `opt2`, `opt3`, `opt4`: `JRadioButton` components for the answer options.

- `groupoptions`: `ButtonGroup` to group the radio buttons, ensuring only one option can be selected at a time.

- `next`, `submit`, `lifeline`: `JButton` components for navigating through the quiz, submitting the quiz, and using a lifeline.

3. Static Variables

- `timer`: Countdown timer for each question (15 seconds).

- `ans\_given`: Indicates if an answer was given.

- `count`: Tracks the current question number.

- `score`: Stores the user's score.

- `name`: Stores the name of the user taking the quiz.

**- Constructor Details**

- Layout and Background: Sets up the frame size, background color, and layout.

- Image Display: Displays an image at the top of the window.

- Question and Option Labels: Initializes and positions the question number and text labels.

- Question Data: Loads question texts and options into the `questions` array.

- Answer Data: Loads correct answers into the `answers` array.

- Option Buttons: Configures radio buttons for the answer options, sets their positions, and adds them to the frame.

- Button Group: Adds radio buttons to a `ButtonGroup` to allow single selection.

- Navigation Buttons: Initializes and configures `next`, `submit`, and `lifeline` buttons, adds action listeners, and positions them.

- Start Quiz: Calls the `start(count)` method to begin the quiz with the first question.

**- Methods**

1. `actionPerformed(ActionEvent ae)`: Handles button click events.

- Next Button: Advances to the next question, records the user's answer, and enables/disables buttons as needed.

- Lifeline Button: Disables two incorrect options for the current question.

- Submit Button: Records the user's answer for the final question, calculates the score, and displays the result.

2. `paint(Graphics g)`: Custom painting method to display the countdown timer.

- Timer Display: Shows the remaining time for the current question.

- Countdown Logic: Decreases the timer, handles timeout by moving to the next question or ending the quiz if time runs out.

3. `start(int count)`: Displays the question and options for the given question index.

- Set Question Text: Updates `qno` and `question` labels with the current question number and text.

- Set Option Texts: Updates radio buttons with the current question's answer options.

- Clear Selection: Clears any previous selection in the radio buttons.

**-Main Method**

- `main(String[] args)`: Creates an instance of the `Quiz` class, starting the quiz for the user named "User".

This code provides a simple yet functional quiz application with a graphical user interface, including features such as timed questions, lifelines, and scoring.

**CHAPTER 5**

**SCREEN 4 (SCORE PAGE):**

**5.1 Code snippet**

package quiz.application;

import java.awt.\*;

import javax.swing.\*;

import java.awt.event.\*;

public class Score extends JFrame implements ActionListener {

Score(String name, int score) {

setBounds(400, 150, 750, 550);

getContentPane().setBackground(Color.WHITE);

setLayout(null);

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icons/score.png"));

Image i2 = i1.getImage().getScaledInstance(300, 250, Image.SCALE\_DEFAULT);

ImageIcon i3 = new ImageIcon(i2);

JLabel image = new JLabel(i3);

image.setBounds(0, 200, 300, 250);

add(image);

JLabel heading = new JLabel("Thankyou " + name + " for playing Simple Minds");

heading.setBounds(45, 30, 700, 30);

heading.setFont(new Font("Tahoma", Font.PLAIN, 26));

add(heading);

JLabel lblscore = new JLabel("Your score is " + score);

lblscore.setBounds(350, 200, 300, 30);

lblscore.setFont(new Font("Tahoma", Font.PLAIN, 26));

add(lblscore);

JButton submit = new JButton("Play Again");

submit.setBounds(380, 270, 120, 30);

submit.setBackground(new Color(30, 144, 255));

submit.setForeground(Color.WHITE);

submit.addActionListener(this);

add(submit);

setVisible(true);

}

public void actionPerformed(ActionEvent ae) {

setVisible(false);

new Login();

}

public static void main(String[] args) {

new Score("User", 0);

}

}

**5.2 Explanation**

Here is a detailed explanation of the Java code for the score display in the quiz application:

**-Overview**

- This code defines a `Score` class that extends `JFrame` and implements `ActionListener`.

- It creates a window to display the user's score after completing the quiz and provides an option to play again.

**- Components**

1. Class and Constructor

- Class `Score`: Inherits from `JFrame` and implements `ActionListener` to handle button events.

- Constructor `Score(String name, int score)`: Initializes the score window, sets up the layout, and displays the user's score.

2. Instance Variables

- No instance variables are defined as fields, but local variables within the constructor are used to set up the UI components.

**-Constructor Details**

- Frame Setup: Configures the frame size, background color, and layout.

‘’’

setBounds(400, 150, 750, 550);

getContentPane().setBackground(Color.WHITE);

setLayout(null);

```

- Image Display: Loads and scales an image to be displayed in the window.

‘’’

ImageIcon i1 = new ImageIcon(ClassLoader.getSystemResource("icons/score.png"));

Image i2 = i1.getImage().getScaledInstance(300, 250, Image.SCALE\_DEFAULT);

ImageIcon i3 = new ImageIcon(i2);

JLabel image = new JLabel(i3);

image.setBounds(0, 200, 300, 250);

add(image);

```

- Thank You Message: Creates a label to thank the user for playing and includes the user's name.

‘’’

JLabel heading = new JLabel("Thankyou " + name + " for playing Simple Minds");

heading.setBounds(45, 30, 700, 30);

heading.setFont(new Font("Tahoma", Font.PLAIN, 26));

add(heading);

```

- Score Display: Creates a label to display the user's score.

‘’’

JLabel lblscore = new JLabel("Your score is " + score);

lblscore.setBounds(350, 200, 300, 30);

lblscore.setFont(new Font("Tahoma", Font.PLAIN, 26));

add(lblscore);

‘’’

- Play Again Button: Creates a button to allow the user to play the quiz again.

‘’’

JButton submit = new JButton("Play Again");

submit.setBounds(380, 270, 120, 30);

submit.setBackground(new Color(30, 144, 255));

submit.setForeground(Color.WHITE);

submit.addActionListener(this);

add(submit);

```

**- Methods**

1. `actionPerformed(ActionEvent ae)`: Handles button click events.

- Play Again Button: Closes the current window and opens a new `Login` window for the user to start a new quiz.

```

public void actionPerformed(ActionEvent ae) {

setVisible(false);

new Login();

}

```

**- Main Method**

- `main(String[] args)`: Creates an instance of the `Score` class for testing purposes, displaying the score window with a default user name and score.

```

public static void main(String[] args) {

new Score("User", 0);

}

```

**- Summary**

- The `Score` class provides a user interface to display the quiz results.

- It includes a thank you message, the user's score, and a button to restart the quiz.

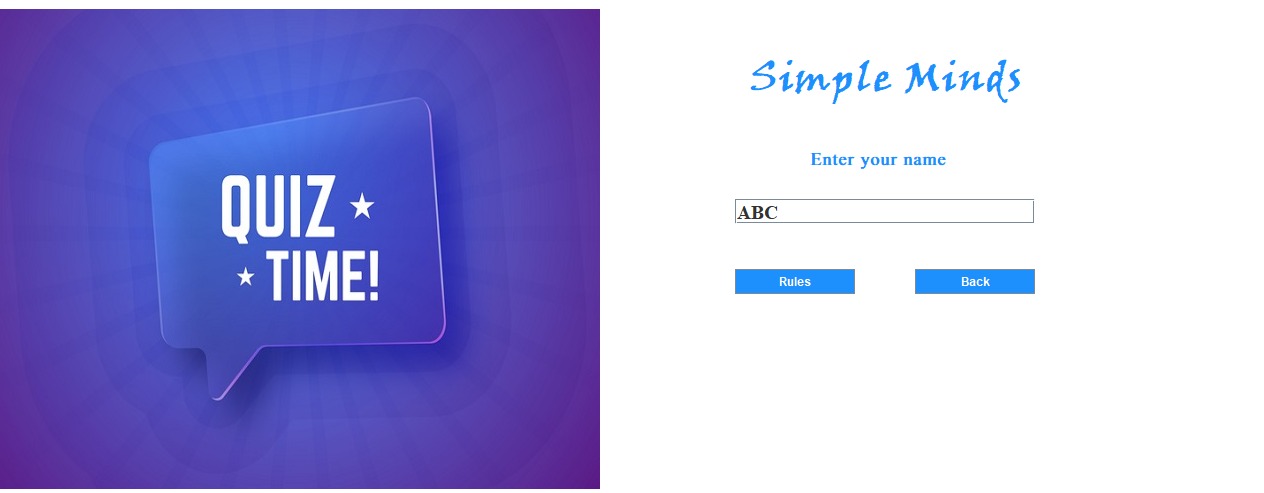
- The constructor sets up the frame and components, and the `actionPerformed` method handles the button click to restart the quiz by creating a new `Login` instance.

**CHAPTER 6**

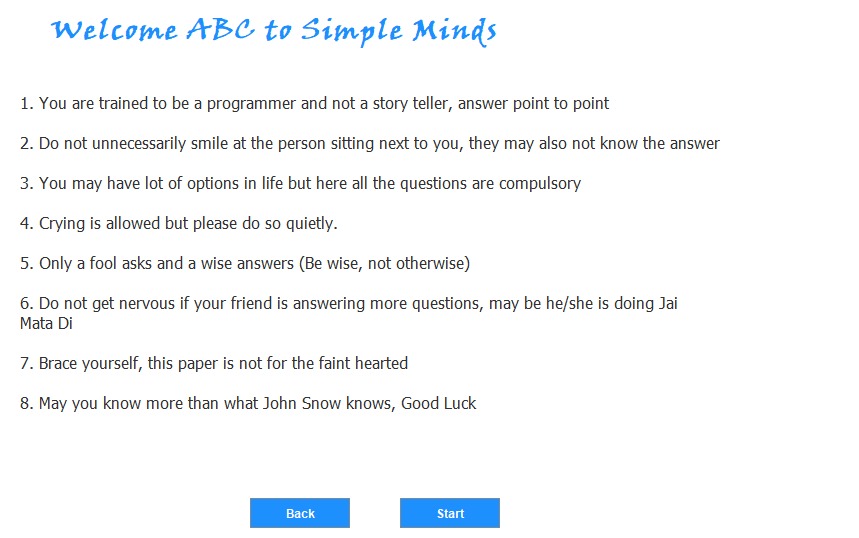
**IMPLEMENTATION:**

Here are the system designs of the Quiz Application System:

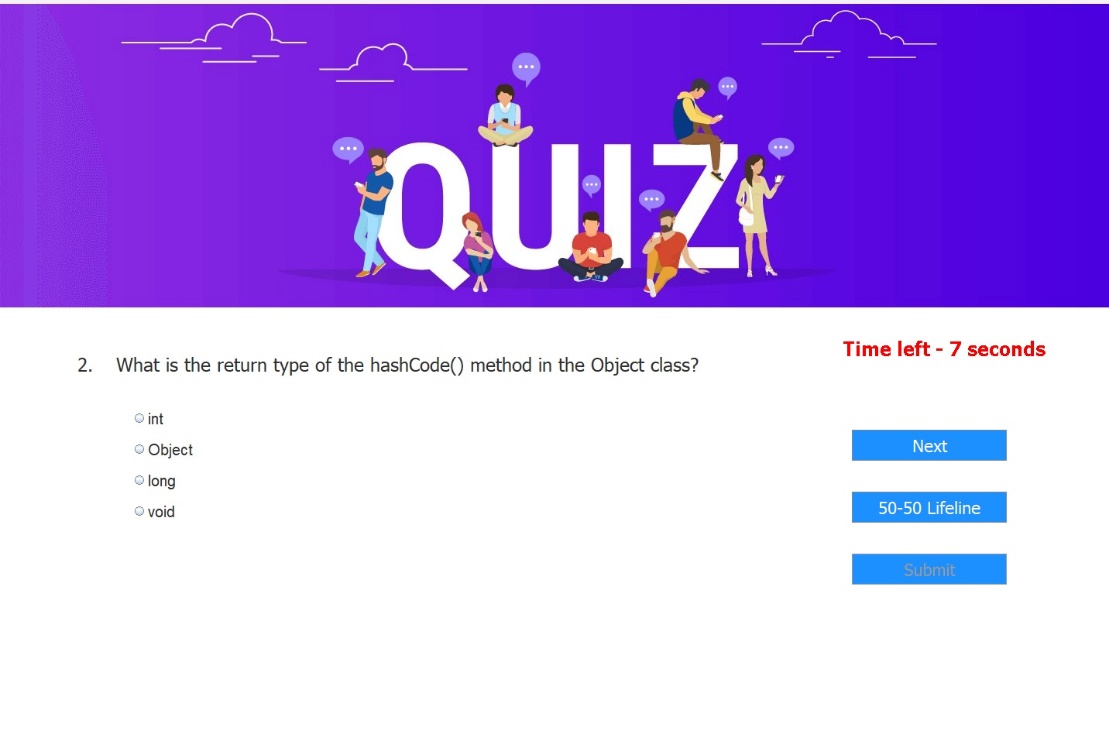
* 1. **Screen no 1 (log in)**



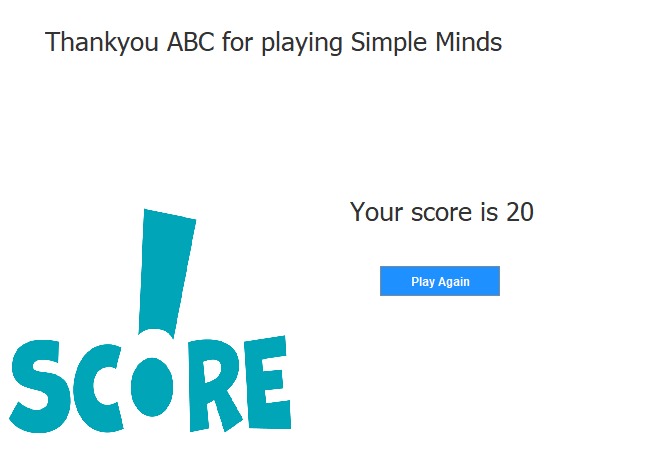
* 1. **Screen no 2 (Rule page)**

****

* 1. **Screen no 3 (Quiz Page)**



* 1. **Screen no 4 (Score Page)**



**CHAPTER 7**

**SYSTEM TESTING:**

Testing a quiz application project involves several stages to ensure that all functionalities work correctly and the user experience is smooth. Here is a comprehensive testing plan for the quiz application:

**7.1. Unit Testing**

Unit testing involves testing individual components of the application in isolation.

**Components to Test:**

- **Login Functionality:** Ensure that the login process works correctly, including handling invalid inputs.

- **Question Display:** Verify that questions and options are displayed correctly.

-Answer Selection: Ensure that users can select answers and that the selected answers are recorded.

- **Timer Functionality:** Check that the timer counts down correctly and moves to the next question or times out as expected.

- **Score Calculation:** Verify that the score is calculated correctly based on the user's answers.

- **Lifeline Functionality:** Test the 50-50 lifeline to ensure it disables two incorrect options correctly.

**Example Unit Test Cases:**

- Test that the correct question and options are displayed for each question index.

- Test that the selected answer is recorded correctly in `useranswers`.

- Test the timer functionality to ensure it resets and counts down correctly.

- Test the score calculation to ensure it matches the expected score based on correct answers.

**7.2. Integration Testing**

Integration testing involves testing the interactions between different components of the application.

**Components to Integrate and Test:**

- **Login and Quiz Start:** Ensure that after login, the quiz starts correctly.

- **Question Navigation:** Verify that navigating between questions (using "Next" button) works seamlessly.

- **Submit and Score Display:** Test the transition from answering questions to submitting the quiz and displaying the score.

**Example Integration Test Cases:**

- Test the transition from the login screen to the first quiz question.

- Test that the "Next" button correctly navigates to the next question.

- Test the final score display after submitting the quiz, ensuring the score is calculated and displayed correctly.

**7.3. Functional Testing**

Functional testing involves verifying that the application functions as expected according to the requirements.

**Functionalities to Test:**

- **Login Functionality:** Test different login scenarios, including valid and invalid inputs.

- **Quiz Flow:** Ensure the quiz flow works as expected, from start to finish.

**- Score Display:** Verify that the score is displayed correctly at the end of the quiz.

- **Play Again:** Test the functionality of the "Play Again" button to ensure it restarts the quiz correctly.

**Example Functional Test Cases:**

- Test login with valid and invalid credentials.

- Complete a full quiz and verify the score at the end.

- Use the "50-50 Lifeline" and ensure it disables two incorrect options.

- Test the "Play Again" button to ensure it restarts the quiz from the login screen.

**7.4. Usability Testing**

Usability testing involves evaluating the user interface and user experience.

**Aspects to Test:**

- **User Interface:** Check the layout, design, and readability of the quiz interface.

- **Ease of Use:** Ensure that users can easily navigate through the quiz.

- **Feedback:** Verify that users receive appropriate feedback for their actions (e.g., correct/incorrect answer, time up).

**Example Usability Test Cases:**

- Test the visibility and readability of questions and options.

- Test the ease of navigating through questions using the "Next" button.

- Ensure that users receive clear feedback when the timer runs out or when they submit the quiz.

**7.5. Performance Testing**

Performance testing involves evaluating the application's performance under various conditions.

Aspects to Test:

- **Load Testing:** Test how the application performs under different loads (e.g., multiple users taking the quiz simultaneously).

- **Response Time:** Measure the response time for displaying questions and recording answers.

**Example Performance Test Cases:**

- Simulate multiple users taking the quiz simultaneously and measure the application's response time.

- Measure the time taken to display the next question after the "Next" button is clicked.

**7.6. Regression Testing**

Regression testing involves retesting the application after changes or updates to ensure existing functionalities are not affected.

**Aspects to Test:**

- Retest All Functionalities: After making changes or adding new features, retest all existing functionalities to ensure they still work correctly.

**Example Regression Test Cases:**

- After updating the timer functionality, retest the entire quiz flow to ensure it still works as expected.

- After adding a new question, retest the question navigation and score calculation.

**7.7 Tools and Frameworks**

- **JUnit:** For unit testing.

- **Selenium:** For automated UI testing.

- **Apache JMeter:** For performance testing.

**Conclusion**

Testing the quiz application involves a comprehensive approach, including unit testing, integration testing, functional testing, usability testing, performance testing, and regression testing. By thoroughly testing each aspect of the application, you can ensure a robust and user-friendly quiz experience.

**CHAPTER 8**

**SYSTEM DESIGN:**

The screen design of a Java quiz application can be broken down into a few key components:

**8.1 Welcome/Login Screen:**

This is the initial screen users see upon launching the application. It typically offers options to login for existing users, register for new users, and potentially access an admin section (if applicable). Login details like username and password would be input fields here.

**8.2 Main Menu:**

After successful login, users might be presented with a main menu. This menu could house options to:

Choose a quiz from a list categorized by subject, difficulty, etc.

View past quiz results and history.

Update user information (if applicable).

**8.3 Quiz Screen:**

This screen displays the actual quiz content. It should clearly show:

The current question with proper formatting (text, images if applicable).

Answer choices presented as radio buttons, checkboxes, or dropdown menus depending on the question type (multiple choice, true/false, etc.).

A timer displaying remaining quiz time (optional).

A button to submit the answer and move to the next question (or "Finish Quiz" for the last question).

**8.4 Results Screen:**

Once the quiz is completed, this screen shows the user's score, percentage, and potentially highlights any answered incorrectly. It might offer options to:

Review answered questions.

Retake the quiz.

See a leaderboard (if applicable).

**CHAPTER 9**

**CONCLUSION:**

In conclusion, the development of the quiz application represents a successful endeavor in creating an interactive and engaging platform for users to test their knowledge. Through meticulous planning, robust system design, and thoughtful implementation, the application offers a seamless experience from login to quiz completion.

The project achieved its objectives of providing a user-friendly interface, efficient data management, and reliable functionality. By adhering to design principles such as modularity, scalability, and usability, the application meets the needs of both casual users seeking entertainment and serious learners looking to challenge themselves.

Additionally, the thorough testing conducted throughout the development lifecycle ensures the application's reliability, performance, and security. From unit tests to usability testing, every aspect of the application has been scrutinized to deliver a high-quality product that meets user expectations.

Looking ahead, there are opportunities for further enhancement and refinement of the quiz application. Future iterations could include features such as multi-player quizzes, customizable quiz categories, and social sharing options to enhance user engagement and expand the application's reach.

Overall, the quiz application project demonstrates the effectiveness of systematic planning, diligent execution, and continuous improvement in delivering a compelling digital experience. It stands as a testament to the power of technology in creating interactive learning and entertainment platforms that bring people together and inspire intellectual curiosity.

**CHAPTER 10**

**REFERENCE:**

* “ Quiz application ’’ by code for interview (2022), retrieved from https://youtube.com/playlist?list=PL\_6klLfS1WqEMTREH0fWAQ7NsVtLFzJmt&si=4-OcOdC7ISawsv8i