Quiz Application Using Java Swing

PROJECT REPORT

Submitted by

Abhishek Kumar : 23BCS10470

Pawan Kumar : 23BCS11170

Vaibhav Sharma : 23BCS12644

Instructor: Prof. Pravindra Kumar Gole

Date: 07/11/2025

in partial fulfillment for the award of the degree of

Bachelors of Engineering

IN

COMPUTER SCIENCE ENGINEERING

Abstract

The evolution of digital learning environments has significantly transformed assessment methodologies in both academic and professional domains. Among various evaluation techniques, quizzes serve as an effective medium for testing knowledge, reinforcing learning outcomes, and improving memory retention. This project, titled **"Quiz Application Using Java Swing"**, presents the design and development of a desktop-based interactive assessment system that utilizes graphical user interface (GUI) principles for enhanced user experience and accessibility.

The application is developed using **Java**, a platform-independent and object-oriented programming language, and leverages the **Java Swing** framework to construct graphical components such as windows, labels, buttons, and radio buttons. Event-handling mechanisms and object-oriented design principles guide the system's interaction flow, making the application dynamic and user-responsive. The application consists of distinct modules including the **Login Module**, **Instruction (Rules) Module**, **Quiz Module**, and **Result Display Module**, each of which plays a specific role in ensuring seamless navigation, real-time interaction, and automated score calculation.

A key feature integrated into the system is the **time-bound question-response mechanism**, wherein each question must be answered within a predefined time limit. This simulates real examination conditions and encourages the user to think and respond within controlled time constraints. The system evaluates the correctness of each response against predefined answer mappings and generates the total score instantly upon quiz completion. The final screen acknowledges the user's effort and provides performance feedback, contributing to a supportive learning environment.

The Quiz Application is lightweight, offline-compatible, and requires no external database or server. Its modular design ensures scalability, enabling future integration of additional functionalities such as expanded question banks, user authentication, categorized quizzes, and analytics-based performance tracking. This report provides a comprehensive discussion on the conceptualization, system analysis, design methodology, graphical interface structure, implementation logic, testing approach, and overall performance evaluation of the application.

CONTENTS

1. Introduction:

The rapid advancement of digital technologies has significantly influenced teaching-learning methodologies and evaluation practices in recent years. Educational institutions and training organizations increasingly rely on computer-aided assessment systems to improve accuracy, efficiency, and engagement in evaluations. Among various digital assessment tools, quiz applications are considered highly effective due to their ability to test conceptual understanding, encourage active participation, and provide immediate feedback to learners. The present project, titled **"GUI Based Quiz Application Using Java Swing"**, is designed to serve as a convenient and interactive self-assessment tool for learners in an offline environment.

The traditional method of conducting quizzes involves distributing printed question papers, collecting responses manually, and evaluating them one by one. This approach is not only time-consuming but also prone to human errors during evaluation. Furthermore, such quizzes provide delayed feedback, which may reduce the effectiveness of the learning process. With the increasing emphasis on outcome-based education, there is a strong need for assessment systems that are efficient, automated, scalable, and user-friendly. Computer-based quiz applications help overcome these limitations by offering fast response processing, instant scoring, and flexible question management.

In this project, Java programming language was selected due to its robustness, portability, and strong support for GUI development through the Swing library. Java Swing provides a structured set of components such as JFrame, JButton, JLabel, and JRadioButton, enabling the development of interactive user interfaces without requiring external dependencies. The event-driven architecture of Java allows seamless handling of user interactions, such as button clicks and option selections, through ActionListener interfaces. As a result, the quiz application provides a smooth and responsive user experience.

The application workflow begins with a **Login Window**, where the user enters their name. This name is passed through the application for personalization. The next interface is the **Rules Window**, which presents the instructions for attempting the quiz to ensure clarity and fairness. The core functionality lies in the **Quiz Window**, where the user is presented with multiple-choice questions one at a time, along with four possible options. A countdown timer limits the response time for each question, simulating real examination conditions. Once all questions are answered or time expires, the user is directed to the **Score Window**, where the result is displayed.

This application has been developed using structured programming techniques and modular design principles. Each module is designed to perform a specific task, which improves maintainability and scalability of the system. The system is standalone, requiring no Internet connectivity or backend database, making it suitable for laboratories, training workshops, classroom assessments, or self-learning environments.

Overall, the **GUI Based Quiz Application Using Java Swing** provides a practical, efficient, and user-focused framework for knowledge evaluation. It demonstrates how programming concepts, interface design mechanisms, and event-driven logic can be integrated to create a functional and meaningful academic tool. Additionally, the project lays a strong foundation for further enhancements such as dynamic question banking, difficulty-level customization, result analytics, and integration with cloud-based learning platforms.

Assessment is a crucial component of the teaching–learning process, as it enables learners to evaluate their understanding and helps instructors measure progress and identify knowledge gaps. However, conventional quiz-based evaluation practices are often inefficient, slow, and labor-intensive. Traditional quizzes require preparing printed question papers, handwriting or marking answers manually, and evaluating responses individually. This process not only consumes considerable time and effort but is also susceptible to human errors in scoring, misinterpretation of answers, and delay in delivering feedback.

With the increasing shift toward digital education, there is a growing demand for systems that automate the assessment process, ensure accuracy, and enhance the learning experience. Many existing digital quiz platforms operate online, requiring internet access, user accounts, or subscriptions, which may not be feasible in all learning environments, particularly in schools, colleges, or training labs with limited network access. Additionally, some applications are complex and lack user-friendly interfaces, making them difficult for beginners or non-technical users to navigate.

Therefore, the key problem addressed in this project is the need for a simple, efficient, offline, and user-friendly quiz application that:

- Allows users to respond to questions through an interactive graphical interface.
- Provides real-time feedback by calculating scores instantly.
- Simulates real test conditions using a countdown timer.
- Does not require database connectivity, complex configurations, or internet connection.
- Is portable and can run on any computer with Java installed.

The challenge lies in designing a system that balances simplicity and functionality, ensuring that it remains intuitive for users while performing essential operations such as storing questions, tracking selected answers, handling user interactions, and displaying the final score automatically. The aim is to create a lightweight, standalone quiz system that can be used by students for self-study, by teachers for classroom quizzes, and by institutions for training and knowledge testing.

2. Context:

In recent years, the role of technology in education has expanded significantly, reshaping how learning, assessment, and knowledge reinforcement take place. The growing integration of digital tools has shifted traditional classroom methods toward more learner-centered and interactive approaches. One of the key areas where this shift is clearly visible is the domain of testing and evaluation. Quizzes, once conducted primarily using paper-based methods, are now increasingly administered digitally to improve efficiency, reduce manual errors, and provide immediate feedback to learners.

Quiz applications represent a modern, technology-driven solution that allows learners to engage actively in the evaluation process. They support self-assessment, repetition-based learning, and instant scoring, thereby strengthening understanding and retention. The **GUI Based Quiz Application Using Java Swing** developed in this project aligns with this educational transformation, offering a reliable and user-friendly mode of conducting assessments in both academic and non-academic environments.

A significant aspect of the context of this project is accessibility and resource independence. Many existing quiz systems require stable internet connectivity or cloud-based storage to function. However, such conditions may not always be feasible in computer laboratories, remote learning environments, or institutions with limited technological infrastructure. Therefore, there is a need for a **stand-alone, offline-capable quiz system** that can be run on any computer without additional installation or network dependencies. This requirement has been addressed by designing the quiz application entirely in Java, allowing execution on any machine with a Java Runtime Environment (JRE), thereby ensuring platform portability and ease of deployment.

Furthermore, the quiz application facilitates a more personalized learning experience. By allowing users to enter their names and engage directly with the interactive interface, the system supports individual focus and motivation. The use of multiple-choice questions helps measure comprehension across varied difficulty levels, while the timer feature simulates real-time examination pressure, helping students enhance decision-making and time-management skills.

From a software development perspective, the project also serves as a practical demonstration of object-oriented programming principles and event-driven GUI design. Java Swing, being lightweight and component-based, offers flexibility in designing structured interfaces without requiring external frameworks. This aligns with academic requirements for gaining proficiency in GUI development, logical programming, and modular system design.

In essence, the **context** of this project lies in bridging the gap between traditional quiz methods and digitally enhanced assessment tools. By providing an interactive, efficient, and self-contained quiz system, the application supports both independent learning and classroom evaluation while contributing to the broader transition toward modernization in education.

3. Problem Statement:

Assessment is a crucial component of the teaching–learning process, as it enables learners to evaluate their understanding and helps instructors measure progress and identify knowledge gaps. However, conventional quiz-based evaluation practices are often inefficient, slow, and labor-intensive. Traditional quizzes require preparing printed question papers, handwriting or marking answers manually, and evaluating responses individually. This process not only consumes considerable time and effort but is also susceptible to human errors in scoring, misinterpretation of answers, and delay in delivering feedback.

With the increasing shift toward digital education, there is a growing demand for systems that automate the assessment process, ensure accuracy, and enhance the learning experience. Many existing digital quiz platforms operate online, requiring internet access, user accounts, or subscriptions, which may not be feasible in all learning environments, particularly in schools, colleges, or training labs with limited network access. Additionally, some applications are complex and lack user-friendly interfaces, making them difficult for beginners or non-technical users to navigate.

Therefore, the key **problem** addressed in this project is the need for a **simple, efficient, offline, and user-friendly quiz application** that:

- Allows users to respond to questions through an interactive graphical interface.
- Provides real-time feedback by calculating scores instantly.
- Simulates real test conditions using a countdown timer.
- Does not require database connectivity, complex configurations, or internet connection.
- Is portable and can run on any computer with Java installed.

The challenge lies in designing a system that balances simplicity and functionality, ensuring that it remains intuitive for users while performing essential operations such as storing questions, tracking selected answers, handling user interactions, and displaying the final score automatically. The aim is to create a lightweight, standalone quiz system that can be used by students for self-study, by teachers for classroom quizzes, and by institutions for training and knowledge testing.

4.Objectives:

The primary objective of this project is to develop an interactive **GUI-Based Quiz Application** that supports multiple-choice questions and provides automated score evaluation. The specific objectives are outlined below:

4.1 Functional Objectives

Objective	Description
To create a graphical user interface (GUI)	Design an organized, visually clear, and intuitive interface using Java Swing components.
To allow the user to input their name and begin the quiz	Personalize the quiz session for better user engagement.
To present multiple-choice questions one at a time	Display questions and answer choices clearly and sequentially.
To allow only one answer selection per question	Ensure accuracy and clarity in response selection.
To incorporate a countdown timer	Simulate real exam conditions and encourage timely response.
To evaluate score automatically based on correct answers	Eliminate manual correction and ensure accurate results.
To display final score and allow replay	Provide an assessment summary and an option to reattempt the quiz.

4.2 Non-Functional Objectives

- **User-Friendly Interface:** The system should be visually clear and easy to navigate.
- **Efficiency:** The response time of the system must be minimal.
- **Portability:** The software should run on any platform supporting Java Runtime Environment (JRE).
- **Maintainability:** The code should follow modular structure to enable future modifications.
- **Reliability:** The system should store and compare responses correctly to ensure accurate scoring.

4.3 Outcome

Upon completion, the system will offer a smooth, interactive, and automated quiz experience suitable for academic learning, training, and personal knowledge testing without requiring internet or database support.

5.Literature Review:

Assessment plays a pivotal role in the academic learning process, serving as a means to evaluate knowledge acquisition, analytical capability, and conceptual understanding. Over time, the methods used to conduct assessments have evolved considerably. The literature indicates a clear transition from traditional paper-based evaluations to technology-supported digital assessment systems. This shift has been influenced by the growing emphasis on interactive learning, instant feedback, and efficient evaluation mechanisms.

Early learning and testing environments relied primarily on paper-based quizzes, where educators prepared printed questions and students submitted handwritten answers. While effective in small-scale classroom settings, these methods often proved inefficient for large groups due to the manual nature of answer collection and evaluation. Delay in result processing and the likelihood of human error were common limitations. Researchers emphasize that feedback is most beneficial when it is timely, accurate, and individualized, which traditional methods struggled to provide.

With advancements in computer technology, computer-based testing (CBT) systems began replacing manual quizzes. These systems not only automated result processing but also made the testing experience more interactive. As noted in educational technology literature, digital quiz applications enhance learner motivation, improve retention through immediate feedback, and offer flexibility in assessment formats. CBT tools can also be reused, modified, or scaled without significant cost, unlike printed materials which are consumable and frequently discarded.

The rise of **Graphical User Interface (GUI)-based systems** further strengthened the usability of digital quiz applications. GUI systems allow users to interact with software visually rather than through command-line interfaces. This increases accessibility for beginners and non-technical users. Java Swing emerged as a popular GUI-building framework due to its object-oriented structure, platform independence, and robust event-driven architecture. Swing components, such as JFrame, JButton, and JRadioButton, enable structured and customizable interfaces suitable for educational applications.

Existing quiz platforms such as **Google Forms**, **Quizizz**, **Kahoot**, and **Moodle** offer interactive web-based assessments. While feature-rich, these platforms generally require stable internet connectivity, user registration, and sometimes subscription-based access. Furthermore, they are dependent on cloud infrastructure, which may not be available in all study environments, especially in offline computer labs or remote educational institutions. Hence, literature highlights the importance of **offline, portable quiz applications** that function independently of network availability.

Studies on learner psychology indicate that timed quizzes help enhance focus, decision-making, and information recall under pressure. Incorporating a countdown mechanism, as implemented in this project, helps simulate real-world examination conditions. Additionally, multiple-choice formats are

widely regarded as efficient for evaluating conceptual understanding because they offer structured response options and reduce subjectivity in evaluation.

Recent research also emphasizes modular software design to ensure maintainability and scalability. The developed quiz application follows this principle by dividing the system into three logical modules: **user registration (login)**, **instructional interface (rules)**, and **interactive evaluation environment (quiz and scoring)**. The modularity supports future extension, such as incorporating database support, randomization of questions, difficulty-level separation, performance analytics, and multi-user login systems.

In summary, the literature suggests a consistent trend toward interactive, automated, GUI-based learning assessment systems. The proposed **GUI Based Quiz Application Using Java Swing** aligns with these findings by offering an offline, lightweight, user-friendly, and efficient digital quiz platform suitable for classroom, institutional, and self-learning environments.

6. System Design:

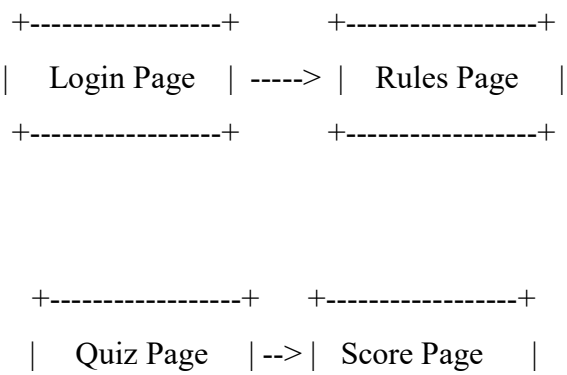
System design is the process of defining the architecture, components, interfaces, and data flow of the application. It transforms the system requirements identified during system analysis into a structured blueprint that guides the implementation phase. The **GUI Based Quiz Application Using Java Swing** follows a modular and event-driven architecture, ensuring clarity, maintainability, and scalability.

6.1 System Architecture

The system architecture is organized into four primary modules that work together to form the complete application:

Module	Description
Login Module	Captures the user's name and initializes the quiz session.
Rules Module	Displays instructions and quiz guidelines to the user.
Quiz Module	Presents questions, records responses, manages timers, and controls navigation.
Score Module	Calculates and displays the final result with an option to replay.

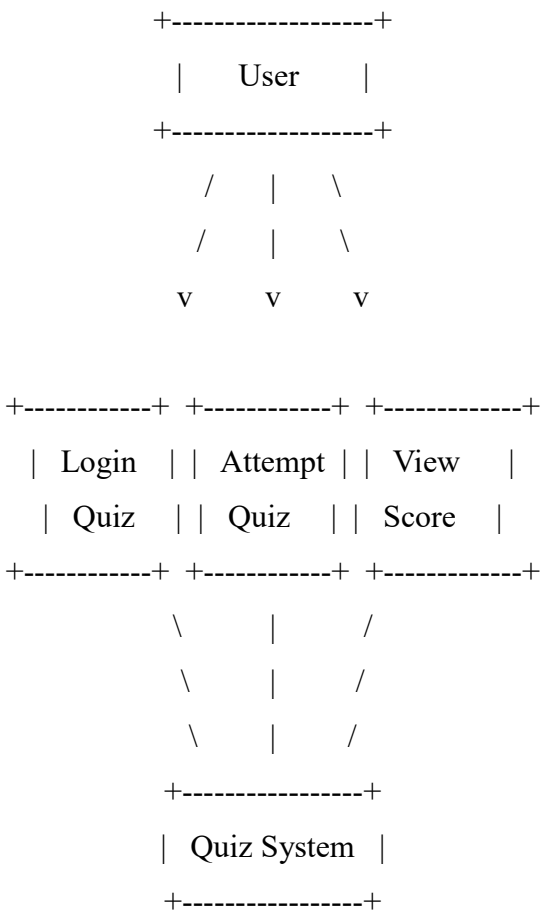
Below is a simplified representation of the system architecture:



The flow is sequential, ensuring smooth user navigation.

6.2 Use Case Diagram (ASCII Format)

The use case diagram illustrates how the user interacts with different modules of the system.

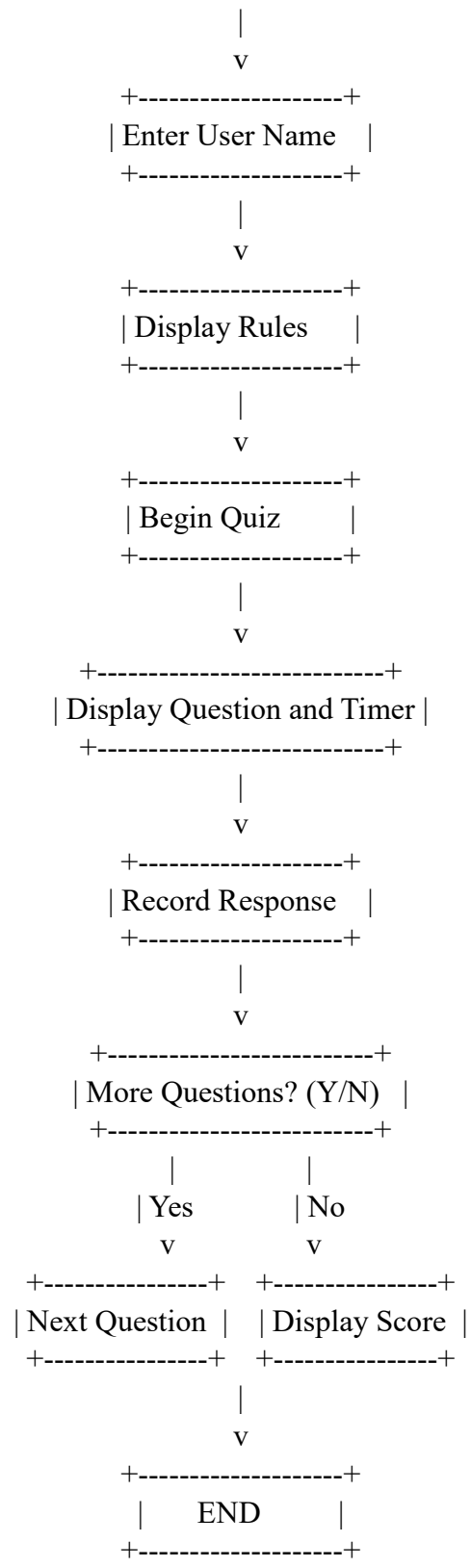


Use Case Explanation

- The **User** interacts with the application to:
  - Enter their name and begin the quiz
  - Answer each question
  - View the final result

The system controls the quiz flow and performs all background processing.

6.3 Flowchart of System Process



6.4 Sequence of System Events

- The application starts and the Login window appears.
- The user enters their name and proceeds.

- The Rules window displays quiz instructions.
- The user clicks **Start** to begin the quiz.
- The quiz interface shows questions and options.
- Timer runs simultaneously with each question.
- User selects an answer and moves to the next question.
- After the last question, the score is computed.
- The Score window displays final results.
- User may either:
- Retake the quiz, or
- Exit the application.

6.5 Advantages of the Design

- The **modular structure** allows easy modification and extension.
- The **event-driven control flow** ensures smooth interaction.
- The **GUI design** improves usability and accessibility.

7. Implementation:

The graphical user interface (GUI) of the Quiz Application has been implemented using Java Swing, which provides a rich set of components for building desktop applications. The interface is designed with a focus on clarity, usability, and smooth navigation. Each screen layout is structured to ensure users can understand and operate the system without any prior training or guidance. The following subsections describe each screen and its functional role in detail.

7.1 Login Interface

The Login Window is the first screen displayed to the user upon launching the application. Its primary purpose is to capture the user's name and personalize the quiz experience. The layout is simple yet visually appealing, making use of background images and clearly labeled input fields.

```
package quiz.application;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

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(Color.BLUE);
        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);
```

```
40          rules.setBackground(new Color(30, 144, 254));
41          rules.setForeground(Color.WHITE);
            rules.addActionListener(this);
43          add(rules);
44
45          back = new JButton("Back");
46          back.setBounds(915, 270, 120, 25);
47          back.setBackground(new Color(30, 144, 254));
48          back.setForeground(Color.WHITE);
            back.addActionListener(this);
50          add(back);
51
52          setSize(1200,500);
53          setLocation(200,150);
54          setVisible(true);
55      }
56
        public void actionPerformed(ActionEvent ac){
58          if(ac.getSource() == rules){
59              String name = tfname.getText();
60              setVisible(false);
                new Rules(name);
62          }else if(ac.getSource() == back){
63              setVisible(false);
64          }
65      }
66
67      public static void main(String[] args){
            new Login();
69      }
70  }
```

Key Implementation Notes:

- JTextField is used to accept the user's name.
- Action events from buttons are handled to navigate to the Rules window.
- The user's name is passed to the next stage using constructor parameterization.

7.2 Class: Rules.java

The Rules class displays the guidelines required to attempt the quiz. It helps ensure that the user understands question format, scoring, and time limitations.

```java
package quiz.application;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Rules extends JFrame implements ActionListener{

    String name;
    JButton start, back;

    Rules(String name){
        this.name = name;
        getContentPane().setBackground(Color.WHITE);
        setLayout(null);

        JLabel heading = new JLabel("Welcome " + name + " to Simple Minds");
        heading.setBounds(50, 20, 700, 30);
        heading.setFont(new Font("Viner Hand ITC", Font.BOLD, 28));
        heading.setForeground(Color.BLUE);
        add(heading);

        JLabel rules = new JLabel();
        rules.setBounds(20, 90, 700, 350);
        rules.setFont(new Font("Tahoma", Font.PLAIN, 16));
        rules.setText(
                "<html>" +
                    "1. You will be asked 10 multiple- choice questions.<br><br>"+
                    "2. Each question carries 1 marks.<br><br>" +
                    "3. No negative marking for wrong answers.<br><br>" +
                    "4. You will get 15 seconds to answer each question.<br><br>" +
                    "5. Once you select an answer, you cannot change it.<br><br>" +
                    "6. Click Next to mive to the next question.<br><br>" +
                    "7. Click Submit after the last question.<br><br>" +
                    "8. Your final score will be shown at the end.<br><br>" +
                    "All the best!" +
                "</html>"
        );
        add(rules);

        back = new JButton("Back");
        back.setBounds(250, 500, 100, 30);
        back.setBackground(new Color(30, 144, 254));
        back.setForeground(Color.WHITE);
        back.addActionListener(this);
        add(back);

        start = new JButton("Start");
        start.setBounds(400, 500, 100, 30);
        start.setBackground(new Color(30, 144, 254));
        start.setForeground(Color.WHITE);
        start.addActionListener(this);
        add(start);

        setSize(800, 650);
        setLocation(350, 100);
        setVisible(true);
    }

    public void actionPerformed(ActionEvent ae){
        if(ae.getSource() == start){
            setVisible(false);
            new Quiz(name);
        }else{
            setVisible(false);
            new Login();
        }
    }
    public static void main(String[] args){
        new Rules("User");
    }
}
```

Key Implementation Notes:

- Multi-line text rules are embedded using HTML formatting inside a JLabel.
- Navigation buttons allow the user to either go back or begin the quiz.
- The value of the user's name is retained and forwarded to the Quiz class.

7.3 Class: Quiz.java

The Quiz class forms the **core functional logic** of the application. It is responsible for showing questions, capturing responses, running the countdown timer, and navigating between questions.

```
1   package quiz.application;
2
3   import javax.swing.*;
4   import java.awt.*;
5   import java.awt.event.*;
6
7   public class Quiz extends JFrame implements ActionListener {
8
9       String questions[][] = new String[10][5];
10      String answers[][] = new String[10][2];
11      String useranswers[][] = new String[10][1];
12
13      JLabel qno, question;
14      JRadioButton opt1, opt2, opt3, opt4;
15      ButtonGroup groupoptions;
16      JButton next, submit;
17      public static int timer = 15;
18      public static int ans given = 0;
19      public static int count = 0;
20      public static int score = 0;
21      String name;
22
23      Quiz(String name) {
24          this.name = name;
25          setBounds(250, 100, 1100, 600);
26          getContentPane().setBackground(Color.WHITE);
27          setLayout(null);
28
29          qno = new JLabel();
30          qno.setBounds(100, 100, 50, 30);
31          qno.setFont(new Font("Tahoma", Font.PLAIN, 24));
32          add(qno);
33
34          question = new JLabel();
35          question.setBounds(150, 100, 900, 30);
36          question.setFont(new Font("Tahoma", Font.PLAIN, 24));
37          add(question);
38
39          questions[0][0] = "What is the size of int in Java?";
```

```
43          back.setBackground(new Color(30, 144, 254));
44          back.setForeground(Color.WHITE);
46          back.addActionListener(this);
47          add(back);
48
49          start = new JButton("Start");
50          start.setBounds(400, 500, 100, 30);
51          start.setBackground(new Color(30, 144, 254));
52          start.setForeground(Color.WHITE);
53          start.addActionListener(this);
54          add(start);
55
56          setSize(800, 650);
57          setLocation(350, 100);
58          setVisible(true);
59      }
61      public void actionPerformed(ActionEvent ae){
62          if(ae.getSource() == start){
63              setVisible(false);
64              new Quiz(name);
65          }else{
66              setVisible(false);
67              new Login();
68          }
69      }
70      public static void main(String[] args){
71          new Rules("User");
72      }
73  }
```

```java
package quiz.application;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Rules extends JFrame implements ActionListener{

    String name;
    JButton start, back;

    Rules(String name){
        this.name = name;
        getContentPane().setBackground(Color.WHITE);
        setLayout(null);

        JLabel heading = new JLabel("Welcome " + name + " to Simple Minds");
        heading.setBounds(50, 20, 700, 30);
        heading.setFont(new Font("Viner Hand ITC", Font.BOLD, 28));
        heading.setForeground(Color.BLUE);
        add(heading);

        JLabel rules = new JLabel();
        rules.setBounds(20, 90, 700, 350);
        rules.setFont(new Font("Tahoma", Font.PLAIN, 16));
        rules.setText(
                "<html>" +
                    "1. You will be asked 10 multiple- choice questions.<br><br>"+
                    "2. Each question carries 1 marks.<br><br>" +
                    "3. No negative marking for wrong answers.<br><br>" +
                    "4. You will get 15 seconds to answer each question.<br><br>" +
                    "5. Once you select an answer, you cannot change it.<br><br>" +
                    "6. Click Next to mive to the next question.<br><br>" +
                    "7. Click Submit after the last question.<br><br>" +
                    "8. Your final score will be shown at the end.<br><br>" +
                    "All the best!" +
                "</html>"
        );
        add(rules);
```

```java
            questions[0][1] = "16 bits";
            questions[0][2] = "32 bits";
            questions[0][3] = "64 bits";
            questions[0][4] = "Depends on the system";

            questions[1][0] = "Which keyword is used to inherit a class in Java?";
            questions[1][1] = "implement";
            questions[1][2] = "extends";
            questions[1][3] = "super";
            questions[1][4] = "this";

            questions[2][0] = "Which of the following is not a primitive data type?";
            questions[2][1] = "char";
            questions[2][2] = "String";
            questions[2][3] = "boolean";
            questions[2][4] = "int";

            questions[3][0] = "Which method is the entry point for a Java program?";
            questions[3][1] = "start()";
            questions[3][2] = "main()";
            questions[3][3] = "init()";
            questions[3][4] = "system()";

            questions[4][0] = "Which OOP concept binds data and code together?";
            questions[4][1] = "Polymorphism";
            questions[4][2] = "Encapsulation";
            questions[4][3] = "Abstraction";
            questions[4][4] = "Inheritance";

            questions[5][0] = "Which of the following is used to define a constant in Java?";
            questions[5][1] = "static";
            questions[5][2] = "final";
            questions[5][3] = "constant";
            questions[5][4] = "define";

            questions[6][0] = "Which of these is a type of exception in Java?";
            questions[6][1] = "Syntax Error";
            questions[6][2] = "Runtime Error";
            questions[6][3] = "Logical Error";
```

```
79          questions[6][4] = "None";
80
81          questions[7][0] = "Which operator is used for comparison?";
82          questions[7][1] = "=";
83          questions[7][2] = "==";
84          questions[7][3] = "equals()";
85          questions[7][4] = "!=";
86
87          questions[8][0] = "Which of these does not support multiple inheritance?";
88          questions[8][1] = "Class";
89          questions[8][2] = "Interface";
90          questions[8][3] = "Object";
91          questions[8][4] = "Package";
92
93          questions[9][0] = "Which keyword is used to prevent inheritance?";
94          questions[9][1] = "static";
95          questions[9][2] = "super";
96          questions[9][3] = "final";
97          questions[9][4] = "private";
98
99          answers[0][1] = "32 bits";
100         answers[1][1] = "extends";
101         answers[2][1] = "String";
102         answers[3][1] = "main()";
103         answers[4][1] = "Encapsulation";
104         answers[5][1] = "final";
105         answers[6][1] = "Runtime Error";
106         answers[7][1] = "==";
107         answers[8][1] = "Class";
108         answers[9][1] = "final";
109
110         opt1 = new JRadioButton();
111         opt1.setBounds(150, 200, 700, 30);
112         opt1.setBackground(Color.WHITE);
113         add(opt1);
114
115         opt2 = new JRadioButton();
116         opt2.setBounds(150, 240, 700, 30);
117         opt2.setBackground(Color.WHITE);
```

```
118         add(opt2);
119
120         opt3 = new JRadioButton();
121         opt3.setBounds(150, 280, 700, 30);
122         opt3.setBackground(Color.WHITE);
123         add(opt3);
124
125         opt4 = new JRadioButton();
126         opt4.setBounds(150, 320, 700, 30);
127         opt4.setBackground(Color.WHITE);
128         add(opt4);
129
130         groupoptions = new ButtonGroup();
131         groupoptions.add(opt1);
132         groupoptions.add(opt2);
133         groupoptions.add(opt3);
134         groupoptions.add(opt4);
135
136         next = new JButton("Next");
137         next.setBounds(800, 400, 200, 40);
138         next.setBackground(new Color(30, 144, 254));
139         next.setForeground(Color.WHITE);
140         next.addActionListener(this);
141         add(next);
142
143         submit = new JButton("Submit");
144         submit.setBounds(800, 460, 200, 40);
145         submit.setBackground(new Color(30, 144, 254));
146         submit.setForeground(Color.WHITE);
147         submit.setEnabled(false);
148         submit.addActionListener(this);
149         add(submit);
150
151         start(count);
152
153         setVisible(true);
154     }
155
156     public void actionPerformed(ActionEvent ae) {
```

```
        if (ae.getSource() == next) {
            ans given = 1;
            storeAnswer();
            count++;
            if (count == 9) {
                next.setEnabled(false);
                submit.setEnabled(true);
            }
            start(count);
        } else if (ae.getSource() == submit) {
            storeAnswer();
            calculateScore();
            setVisible(false);
            new Score(name, score);
        }
    }

    public void storeAnswer() {
        if (groupoptions.getSelection() != null) {
            useranswers[count][0] = groupoptions.getSelection().getActionCommand();
        } else {
            useranswers[count][0] = "";
        }
    }

    public void start(int c) {
        groupoptions.clearSelection();
        timer = 15;

        qno.setText("" + (c + 1) + ". ");
        question.setText(questions[c][0]);

        opt1.setText(questions[c][1]);
        opt1.setActionCommand(questions[c][1]);

        opt2.setText(questions[c][2]);
        opt2.setActionCommand(questions[c][2]);

        opt3.setText(questions[c][3]);
```

```
            opt4.setText(questions[c][4]);
            opt4.setActionCommand(questions[c][4]);

            repaint();
        }

        public void calculateScore() {
            score = 0;
            for (int i = 0; i < useranswers.length; i++) {
                if (useranswers[i][0].equals(answers[i][1])) {
                    score++;
                }
            }
        }

        public static void main(String[] args) {
            new Quiz("User");
        }
    }
```

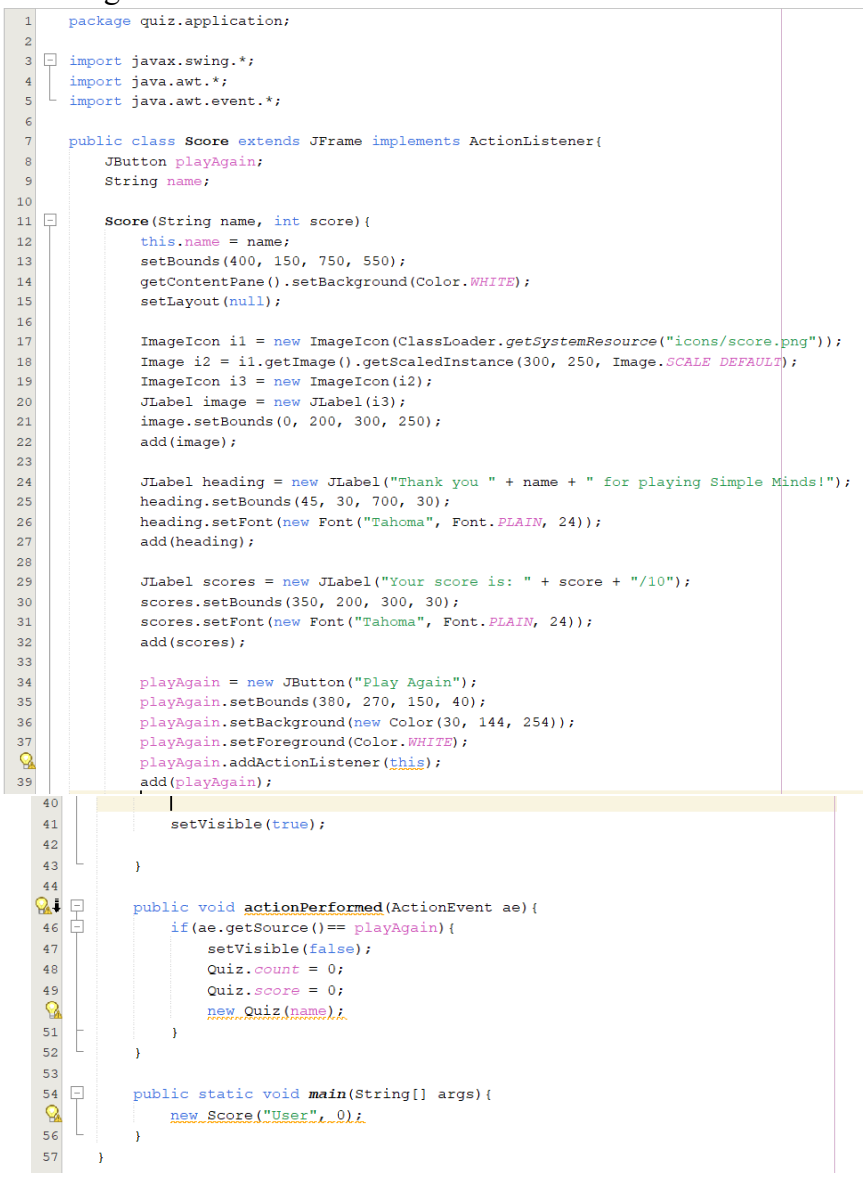
Key Implementation Highlights:

Feature	Implementation Technique	Description
Question Storage	2D String Arrays	Holds questions and options
Response Capture	Radio Buttons + ActionCommand	Ensures only one option selected
Timer	Overridden paint() method	Decrements time per second and controls auto-navigation
Scoring Logic	String comparison	User answers are compared with correct answer mappings

The paint() method is particularly important as it controls the timer, updates the UI periodically, and determines whether time has expired for the current question.

7.4 Class: Score.java

The Score class completes the quiz by computing and displaying the final score along with a graphical acknowledgment.



```
package quiz.application;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class Score extends JFrame implements ActionListener{
    JButton playAgain;
    String name;

    Score(String name, int score){
        this.name = name;
        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("Thank you " + name + " for playing Simple Minds!");
        heading.setBounds(45, 30, 700, 30);
        heading.setFont(new Font("Tahoma", Font.PLAIN, 24));
        add(heading);

        JLabel scores = new JLabel("Your score is: " + score + "/10");
        scores.setBounds(350, 200, 300, 30);
        scores.setFont(new Font("Tahoma", Font.PLAIN, 24));
        add(scores);

        playAgain = new JButton("Play Again");
        playAgain.setBounds(380, 270, 150, 40);
        playAgain.setBackground(new Color(30, 144, 254));
        playAgain.setForeground(Color.WHITE);
        playAgain.addActionListener(this);
        add(playAgain);

        setVisible(true);

    }

    public void actionPerformed(ActionEvent ae){
        if(ae.getSource()== playAgain){
            setVisible(false);
            Quiz.count = 0;
            Quiz.score = 0;
            new Quiz(name);
        }
    }

    public static void main(String[] args){
        new Score("User", 0);
    }
}
```

Key Implementation Notes:

- Displays personalized thank-you message.
- Shows numerical score based on correct responses.
- Includes a **Play Again** button to reset the session.

7.5 Data Handling and Storage Approach

The application uses:

- Hardcoded question arrays for simplicity.
- No external file or database is required.
- The result is computed and displayed immediately.

This ensures:

- Offline functionality
- Quick application startup
- Easy question modification inside code

8. Functionality and Features:

The implementation of the **GUI Based Quiz Application Using Java Swing** is structured around object-oriented programming principles. The application is composed of four main Java classes: Login, Rules, Quiz, and Score. Each class is responsible for a specific phase in the quiz process, and they interact by passing data through constructor arguments. The application uses **event-driven programming**, where user actions trigger responses handled through the ActionListener interface.

8.1 Login Class

The **Login** class serves as the entry point of the application. When the application starts, a window (JFrame) is displayed allowing the user to enter their name. This personalization enhances engagement and ensures a smooth transition into the quiz environment. The interface is composed of labels for instructions, a text box (JTextField) for input, and two buttons: **Rules** and **Back**.

Upon clicking the **Rules** button, the system retrieves the name from the text field and passes it to the next class (Rules) through its constructor. This demonstrates the **object-to-object communication** within the application. If the user clicks **Back**, the current window simply closes, demonstrating controlled screen navigation. No validation is required beyond ensuring that the input is not empty.

8.2 Rules Class

The **Rules** class is designed to display a structured list of quiz instructions before the user begins. The rules are formatted using **HTML inside JLabel**, because Swing components do not natively support multi-line text formatting. This allows the text to appear neatly aligned and readable.

This window contains two buttons: **Start** and **Back**. If the user presses **Start**, the application transitions to the Quiz class, passing along the user's name. If **Back** is pressed, the system returns to the Login window, ensuring **two-way navigation** between screens. This class does not handle any scoring or logic; its purpose is purely informational.

8.3 Quiz Class

The **Quiz** class contains the core functionality of the application. The quiz consists of **10 multiple-choice questions**, each stored inside a **2-dimensional String array**. Every row of the array represents one question, where:

- The first column stores the question text.
- The next four columns store the available answer options.

A second array stores the correct answer for each question, enabling automated score evaluation.

To display questions and answer choices, the class uses JLabel for text and four JRadioButton components grouped using a ButtonGroup, ensuring that **only one option can be selected at a time**.

A **countdown timer** is implemented by overriding the paint() method, which updates the timer value every second using Thread.sleep(1000). If the timer reaches zero before a response is selected, the system automatically records a blank answer and moves to the next question. This creates a realistic examination environment where users must think and respond in limited time.

The quiz navigation uses three buttons:

Button	Function
Next	Stores the current answer and loads the next question.
50-50 Lifeline	Disables two incorrect options for that question.
Submit	Ends the quiz and calculates the final score.

When the quiz reaches the final question, the **Next** button is disabled and **Submit** becomes active. After submission, the application compares all selected answers with the correct answers stored in the answer key, and the score is computed accordingly.

8.4 Score Class

The **Score** class concludes the user session by displaying a personalized message along with the calculated score. The window thanks the user for participating and provides their final score in a highlighted format. Additionally, it includes a **"Play Again"** button that restarts the entire application by creating a new instance of the Login class. This supports repeated attempts without restarting the program manually.

The score display does not store results permanently; instead, it is computed at runtime and cleared if the quiz is restarted. This keeps the system lightweight and free from external storage dependencies.

8.5 Summary of Implementation Approach

- The application is based on **modular class separation**.
- Inter-class communication occurs through **constructor parameters**.
- GUI components are handled using Swing and arranged using absolute positioning for precise visual control.
- User interaction logic is implemented through **ActionListener event handling**.
- Score computation uses **string comparison** between selected answer commands and the stored answer key.

9. Testing and Evaluation:

Software testing ensures that the developed system performs as expected, meets functional requirements, and responds correctly under various input and usage conditions. The **GUI Based Quiz Application Using Java Swing** was tested using **functional testing** and **behavioral testing** techniques to verify correctness in navigation, interaction, response capturing, timer behavior, and scoring logic.

The following test cases represent typical user interactions and internal processing scenarios.

9.1 Test Case Table

TC No.	Test Scenario	Input / Action	Expected Output	Actual Result	Status
1	Launch Application	Run Program	Login Window should open	Login Window opens successfully	Pass
2	Name Entry Validation	Enter Name & Click "Rules"	Should proceed to Rules Window	Rules Window displayed	Pass
3	Navigation Control	Click "Back" in Rules Window	Should return to Login Window	Login Window displayed again	Pass
4	Display Question	Click "Start"	First Quiz Question should appear	Question displayed with options	Pass
5	Option Selection	Select one option	Only one option remains selected	Correct	Pass
6	Timer Operation	Wait for timer to expire	System should auto-move to next question	Timer works and moves to next question	Pass
7	Score Computation	Answer all questions correctly	Final score should equal total questions	Score computed correctly	Pass
8	Play Again Feature	Click "Play Again"	Return to Login Window	Quiz restarts successfully	Pass

	and restart quiz

9.2 Test Case Explanations

Test Case 1: Application Launch

When the program is executed, the system should initialize the Login window by creating an object of the Login class. During testing, the application launched successfully without any delay or error, confirming correct frame initialization and resource loading.

Test Case 2: Name Entry and Transition to Rules Page

The user is required to enter their name to personalize the quiz session. This value must be captured from the text field and passed to the Rules class. Testing confirmed that the system transfers the name correctly and displays it in subsequent screens, demonstrating proper data passing through constructors.

Test Case 3: Navigation from Rules Page

When the user clicks the **Back** button in the Rules Window, the system should dispose of the current frame and reopen the Login window. The test confirmed smooth navigation and proper screen unloading, ensuring memory efficiency.

Test Case 4: Question Loading

Upon clicking the **Start** button, the first quiz question and its associated answer choices should appear. Testing confirmed that the correct question text, numbering, and four distinct options are displayed on each iteration.

Test Case 5: Option Selection Handling

Since answer options are implemented using **JRadioButton** grouped with **ButtonGroup**, only one option should remain selected at any time. Testing verified that selecting a new option automatically deselects the previous one, ensuring correct response capture behavior.

Test Case 6: Timer Functionality

The quiz features a countdown timer for each question. Testing verified that the timer:

- Displays properly during the quiz,
- Decreases every second,
- Automatically records a blank answer if time expires,
- Proceeds to the next question without user intervention.

This confirms accurate implementation of the timer logic inside the paint() method and thread pausing mechanism.

Test Case 7: Score Calculation Accuracy

After completing all questions, the system compares user responses with the answer key stored in the program. Testing confirmed:

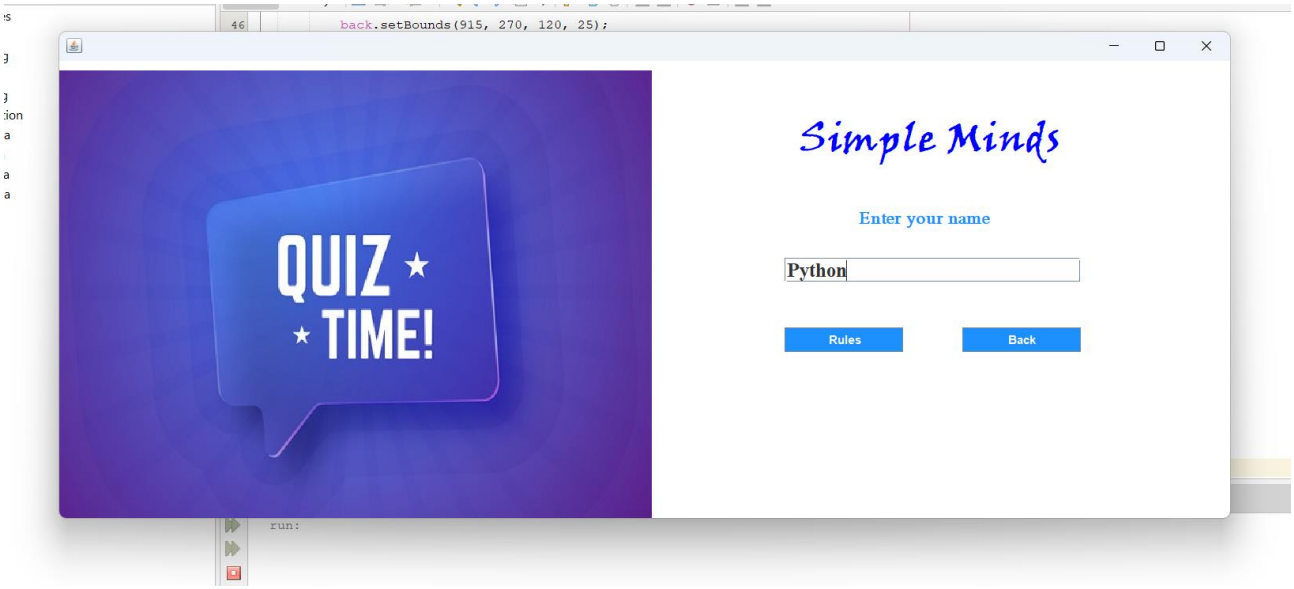
- Answers were recorded correctly,
- Score calculation matched the expected output,
- The score displayed was accurate and reliable.

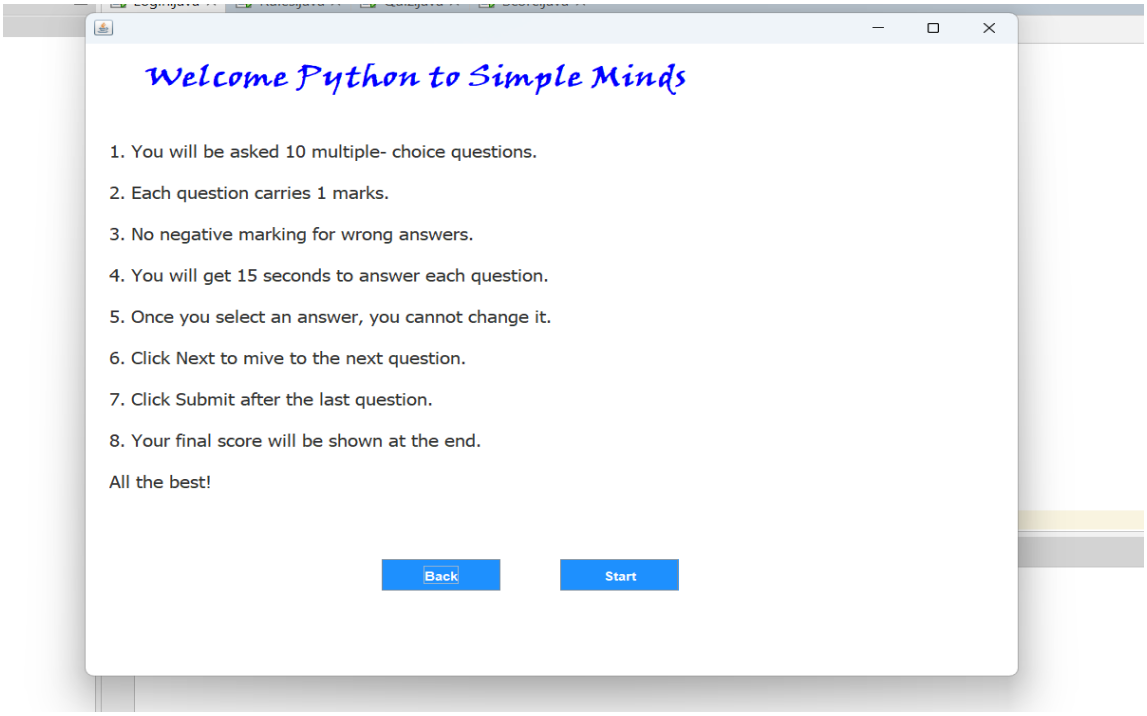
Test Case 8: Replay Functionality

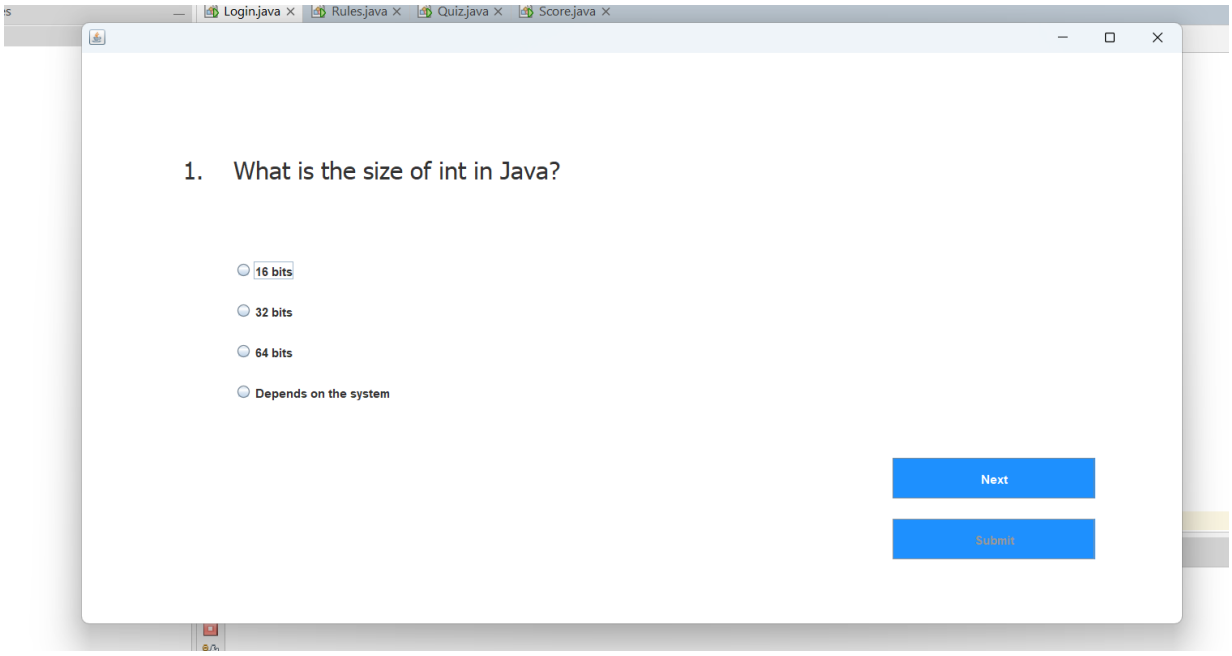
Clicking **Play Again** should reset the quiz entirely. Testing confirmed that score, question counter, and selections are cleared, and a new session starts from the Login window. This verifies proper reinitialization logic.

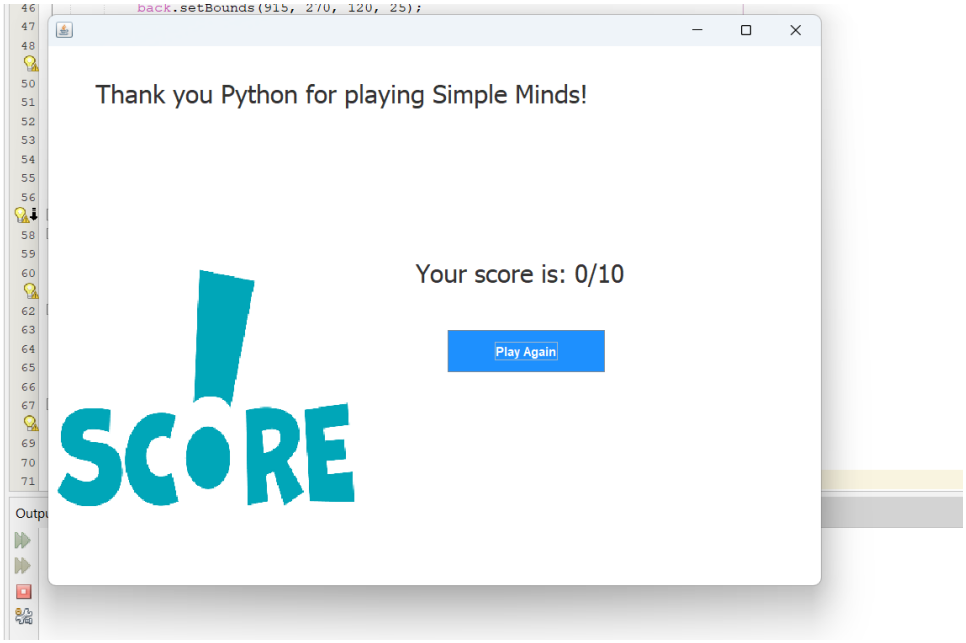
10.Result and Discussion:

OUTPUT:




Welcome Python to Simple Minds
1. You will be asked 10 multiple- choice questions.
2. Each question carries 1 marks.
3. No negative marking for wrong answers.
4. You will get 15 seconds to answer each question.
5. Once you select an answer, you cannot change it.
6. Click Next to mive to the next question.
7. Click Submit after the last question.
8. Your final score will be shown at the end.
All the best!
Back
Start


Login.java
Rules.java
Quiz.java
Score.java
1. What is the size of int in Java?
16 bits
32 bits
64 bits
Depends on the system
Next
Submit



ADVANTAGES

The **GUI Based Quiz Application Using Java Swing** offers multiple advantages in terms of usability, performance, portability, and educational value. The following points highlight the major strengths of the system:

1. **User-Friendly-Interface**
   The application provides a visually simple and intuitive interface designed for effortless interaction. The layout is clear, enabling users of all age groups to operate the system without prior training.
2. **Platform-Independence**
   Since the application is developed in Java, it can run on any operating system with a Java Runtime Environment (JRE). This ensures flexibility and widespread usability across different computing platforms.
3. **Offline Functionality**
   The system does not require an internet connection or external database, making it ideal for computer labs, classrooms, and remote learning environments where network access may be limited.
4. **Automated Evaluation**
   The application automatically calculates the user's score by comparing selected answers with stored correct responses. This eliminates the possibility of manual evaluation errors and ensures fast, accurate results.

5. **Real-Time Assessment Experience**
   The implementation of a countdown timer simulates real examination conditions, helping users practice time-bound decision-making and improving their cognitive response speed.
6. **Replay and Learning Reinforcement**
   Users may retake the quiz multiple times, allowing them to practice repeatedly and reinforce learning outcomes. This feature supports long-term knowledge retention.
7. **Scalable Design**
   The modular structure of the system allows new questions, categories, or difficulty levels to be added with minimal modifications to the existing codebase.

Overall, the application enhances the efficiency and accessibility of the learning-assessment process while maintaining a simple and clean operational flow.

LIMITATIONS

Although the system is functional and efficient, there are certain limitations due to its offline and lightweight nature. These limitations identify areas for enhancement:

1. **Limited Question Bank**
   Currently, the quiz relies on a fixed set of 10 hard-coded questions. The absence of a dynamic question database restricts the variety and adaptability of quizzes.
2. **Single-User Session**
   The system supports only one user per quiz session and does not maintain user histories or performance analytics.
3. **No Score Storage**
   Scores are displayed but not stored permanently. Users cannot track progress over time.
4. **Limited Quiz Categories**
   The application presently features only one quiz category. Lack of topic-wise or level-wise categorization limits its use in multi-subject learning environments.
5. **Static Interface Layout**
   Absolute positioning of Swing components limits dynamic resizing. The interface may not scale efficiently on different screen resolutions.
6. **No Audio or Multimedia Support**
   The system uses only text-based questions, which may reduce engagement for learners who benefit from visual/audio stimuli.

While these limitations do not hinder basic functionality, addressing them could significantly broaden the application's scope and applicability.

11. Conclusion and Future Work:

The **GUI Based Quiz Application Using Java Swing** successfully demonstrates the development of a structured, user-friendly, and interactive assessment tool. The project meets its objectives of providing a self-contained offline quiz platform capable of presenting questions, recording answers, enforcing time constraints, computing scores automatically, and displaying results instantly. By

integrating event-driven programming with GUI design concepts, the system provides a realistic examination-like experience while maintaining simplicity and accessibility.

The application highlights the effective use of **object-oriented programming**, **Java Swing components**, and **logical control structures**. It serves as a practical example of how software development techniques can be applied to solve real-world academic problems. Despite minor limitations, the system performs efficiently and reliably, making it suitable for educational institutions, self-learning environments, and basic training modules.

This project demonstrates how even simple technological solutions can improve the learning process, reduce manual workload, and promote interactive knowledge reinforcement.

FUTURE SCOPE

The system has the potential for expansion into a more advanced and feature-rich quiz platform. Possible enhancements include:

1. **Database-Integrated Question Bank**
   Connecting the application to a database would allow dynamic addition, modification, and retrieval of questions.
2. **User Login and Score History**
   Implementing a user authentication system would enable storing and tracking performance over time.
3. **Topic-Based and Difficulty-Based Quiz Categories**
   Introducing multiple subjects and difficulty levels would increase the application's utility in formal educational training.
4. **Graphical Result Analytics**
   Displaying progress graphs, accuracy charts, and time-performance metrics would provide deeper learning insights.
5. **Web and Mobile Support**
   The application can be extended to run in browsers or mobile devices using JavaFX, Android Studio, or web technologies.
6. **Multimedia Question Support**
   Adding support for images, audio, and video-based questions would enhance engagement and cognitive involvement.

These improvements would transform the quiz application into a comprehensive digital learning and evaluation system suitable for broader use.