5CS019

Object-Oriented Design and Programming

“Quiz Report"

Name: Shubham Raj Pandey

Group: L5CG18

Canvas ID: 2407776

Submission Date: 12 February 2025

Table of Contents

Introduction

The main focus of this coursework lies in developing a Quiz Application based on a GUI that entirely uses the Java MySQL combination. The key aim is to develop an interactive quiz system; through it, players can take quizzes, check their performances, and view reports. At the same time, this should allow an easy process through which an admin can manage all questions and players.

Objectives

- A working Quiz System with multiple-choice questions stored in a database.
- The creation of a structure for the quiz experiences based on difficulty levels (Beginner, Intermediate, Advanced).
- Player data management and performance reports through a database approach.
- An admin console for secure adding, updating, and deleting of quiz questions.
- Statistical reports and leaderboards to monitor player progression and high scores.

Expected Outcomes

- Completely developed quiz app with a user interface.
- A database management system for quiz data that is completely secure and effective.
- A system of dynamic reports on player scores and statistics of quiz.
- A well-documented project that includes a class diagram, test cases and Javadoc

Methodology

The software development lifecycle employed in producing this coursework is the Agile Model that incorporates iteration and increment with feedback at every stage:

Requirement Gathering & Analysis: Identification of key features such as user registration, quiz page interface, leaderboard tracking, etc., to fulfill user requirements.

Design: Wireframes of the application's UI were created, and a database schema for persisting player and quiz data was designed.

Development: The Java classes were developed implementing the Swing based UI and using MySQL to do the backend data operations with smooth interaction of the frontend and database.

Testing: Testing was done continuously throughout the development stage, including unit testing of individual components and integration testing to ensure system-wide coherence.

Deployment & Feedback: An app was deployed for feedback, followed by a careful observation of user interaction for subsequent improvements and bug rectifications given feedback.

The tools involved in the coursework include:

IDE: eclipse

Database: MySQL

UI Framework: Java Swing

Version Control: Git for source code management

Testing Framework: JUnit for unit testing and user-interface manual testing

Implementation

Competitor Class

Competitor class takes on a basic structure in the quiz application and handles individual details of participants plus the score of such participants. Basically, all the significant features such as ID, name, age, country, quiz level, and score are kept in the system. This section elaborates the logic of the class methods and the other added attributes for better functioning.

The Competitor class contains following attributes:

- competitorId → It represents the unique identifier for each competitor.
- competitorName → A Name object representing the competitor's full name, which includes first name, last name, and initials.
- competitorLevel → Defines the difficulty level assigned to the competitor: Beginner, Intermediate or Advanced.
- competitorCountry → Country in which competitor belongs.
- competitorAge → It is used to represent age of the competitor, which is helpful in demographic analysis.
- scores → This is an int array which holds the different scores of the competitor from several quiz attempts.

MYSQL and Arrays

The Competitor class now supports having arrays of scores, meaning that multiple score updates were likely implemented (perhaps for different rounds of a quiz or question categories) and overall score calculations based on these scores.

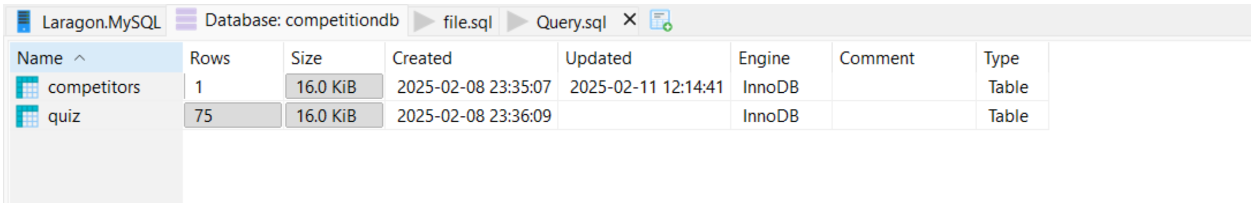
The software implements arrays for managing and updating scores for each quiz because each player will have an individual score stored. Player objects contain arrays to hold scores for different quizzes, updating the respective arrays for player scores after every quiz.

MySQL database connectivity via JDBC (which stands for Java Database Connectivity) was used for the storing and retrieval of player data and scores. The Competitor class connects to the database by establishing a connection using JDBC. That connection is used to execute the following SQL queries:

- INSERT to store player details and scores.
- UPDATE to update player's scores after the session of the quiz.
- SELECT to retrieve player data and scores when required, such as fetching the leaderboard or providing player statistics.

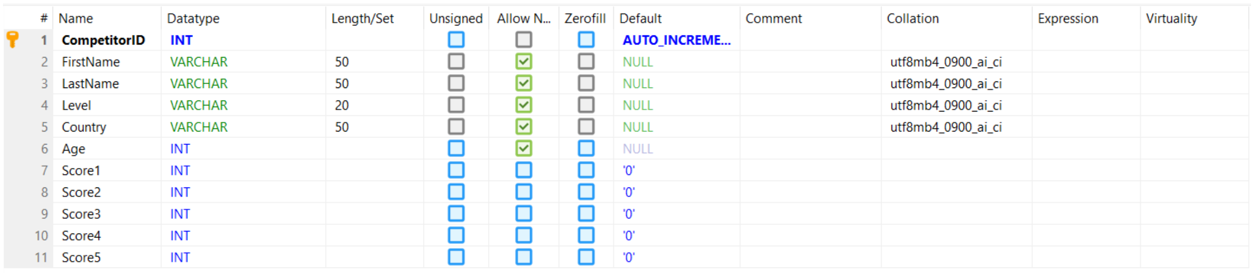
The player data (including scores) in MySQL is stored in a competitor table, and each score is linked to a unique player ID. The class conducts all such operations using SQL for executing queries with the assistance of the JDBC connection.

Table of the competitiondb database:



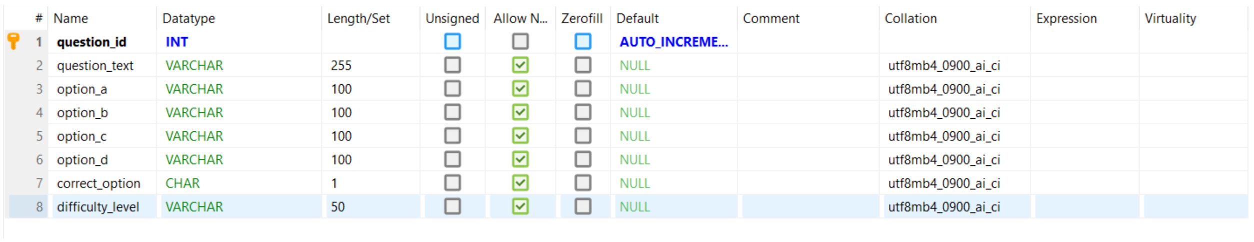
Name	Rows	Size	Created	Updated	Engine	Comment	Type
competitors	1	16.0 KiB	2025-02-08 23:35:07	2025-02-11 12:14:41	InnoDB		Table
quiz	75	16.0 KiB	2025-02-08 23:36:09		InnoDB		Table

Structure of the competitors table:



#	Name	Datatype	Length/Set	Unsigned	Allow N...	Zerofill	Default	Comment	Collation	Expression	Virtuality
1	CompetitorID	INT		☐	☐	☐	AUTO_INCREME...				
2	FirstName	VARCHAR	50	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
3	LastName	VARCHAR	50	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
4	Level	VARCHAR	20	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
5	Country	VARCHAR	50	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
6	Age	INT		☐	✓	☐	NULL				
7	Score1	INT		☐	☐	☐	'0'				
8	Score2	INT		☐	☐	☐	'0'				
9	Score3	INT		☐	☐	☐	'0'				
10	Score4	INT		☐	☐	☐	'0'				
11	Score5	INT		☐	☐	☐	'0'				

Structure of the quiz table:



#	Name	Datatype	Length/Set	Unsigned	Allow N...	Zerofill	Default	Comment	Collation	Expression	Virtuality
1	question_id	INT		☐	☐	☐	AUTO_INCREME...				
2	question_text	VARCHAR	255	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
3	option_a	VARCHAR	100	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
4	option_b	VARCHAR	100	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
5	option_c	VARCHAR	100	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
6	option_d	VARCHAR	100	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
7	correct_option	CHAR	1	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		
8	difficulty_level	VARCHAR	50	☐	✓	☐	NULL		utf8mb4_0900_ai_ci		

MySQL Integration and Reports

The AdminPanel class is implemented to interact with the Competitor class in order to obtain and display reports on user performance. The LeaderboardPage holds responsibility for ranking competitors according to their overall scores, which are obtained dynamically through SQL queries. Furthermore, administrators can add detailed reports based on individual and overall quiz performance that ensure effective and systematic data retrieval.

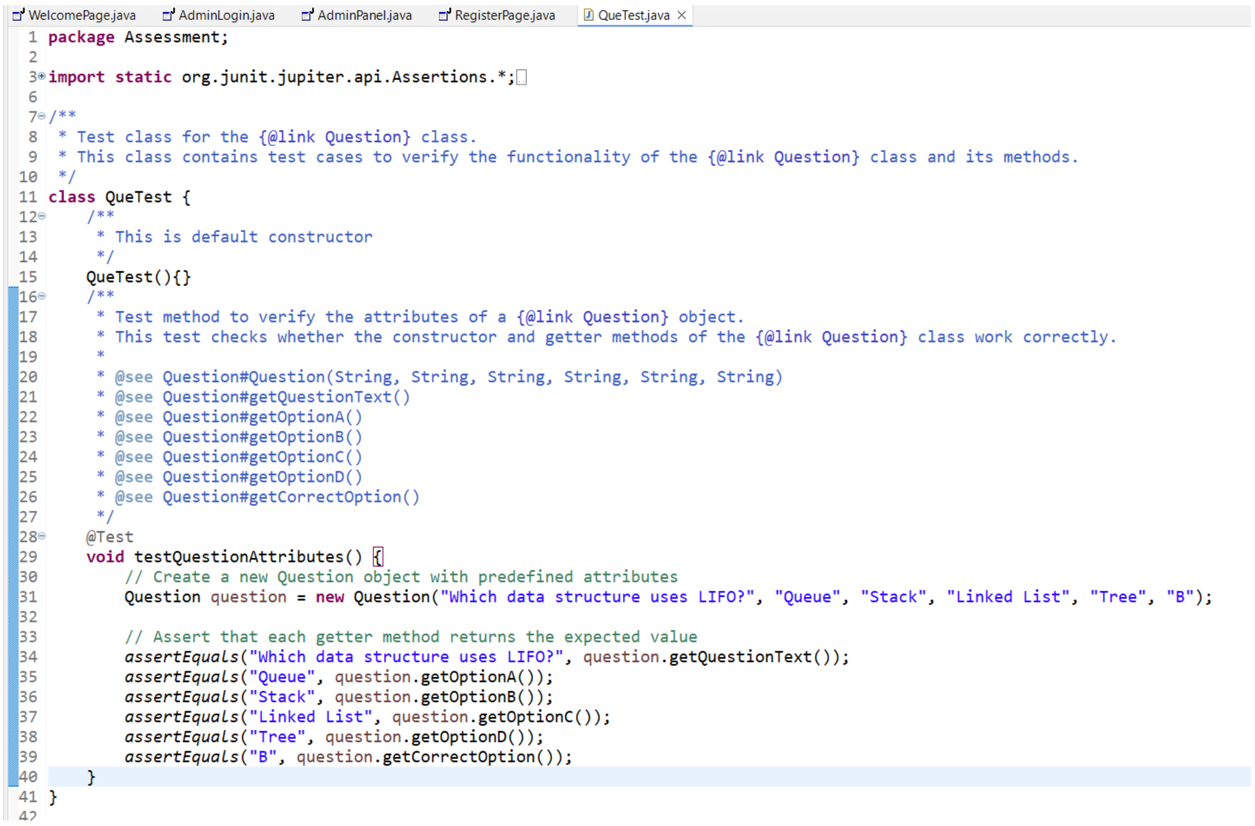
Error Handling

The error-handling features are meant for the invalid inputs, MySQL connection failures, and unauthorized attempts. It prevents the applications from crashing due to such problems and conveys the appropriate messages to users. Some of the main features of error-handling include:

- Invalid Input Handling: Processing a user's input only if it is valid using try-catch blocks.
- The inability to connect to MySQL: Exception handling for database connectivity errors and user-friendly messages on error.
- Errors in Executing Queries: Ensures that SQL queries are executed in a secure environment that deals well with syntax and missing data errors.
- Access Denial: After several failed attempts, the user will not be able to access the application anymore.

Testing

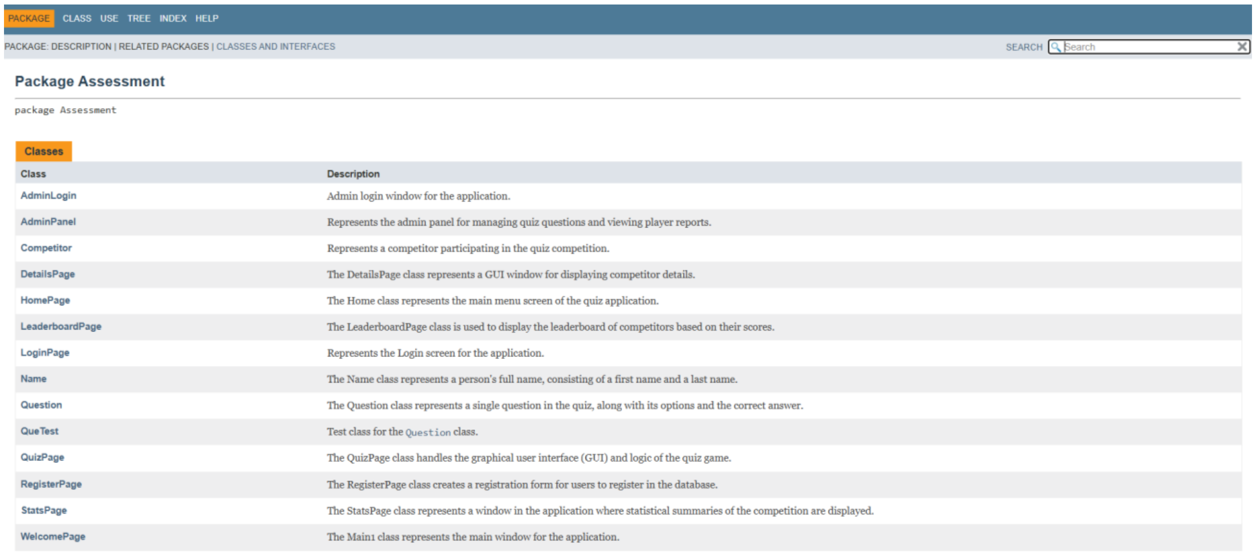
Unit tests have incorporated into the process to check the work of key methods in different classes. The current example has the functionality of the Competitor class checked by the CompetitorTest test class. This checks whether the scores are duly recorded and the final score is calculated accurately. A DatabaseTest checks MySQL connectivity and validates that the query returns the expected results.



```java
package Assessment;

import static org.junit.jupiter.api.Assertions.*;

/**
 * Test class for the {@link Question} class.
 * This class contains test cases to verify the functionality of the {@link Question} class and its methods.
 */
class QueTest {
    /**
     * This is default constructor
     */
    QueTest(){}
    /**
     * Test method to verify the attributes of a {@link Question} object.
     * This test checks whether the constructor and getter methods of the {@link Question} class work correctly.
     *
     * @see Question#Question(String, String, String, String, String, String)
     * @see Question#getQuestionText()
     * @see Question#getOptionA()
     * @see Question#getOptionB()
     * @see Question#getOptionC()
     * @see Question#getOptionD()
     * @see Question#getCorrectOption()
     */
    @Test
    void testQuestionAttributes() {
        // Create a new Question object with predefined attributes
        Question question = new Question("Which data structure uses LIFO?", "Queue", "Stack", "Linked List", "Tree", "B");

        // Assert that each getter method returns the expected value
        assertEquals("Which data structure uses LIFO?", question.getQuestionText());
        assertEquals("Queue", question.getOptionA());
        assertEquals("Stack", question.getOptionB());
        assertEquals("Linked List", question.getOptionC());
        assertEquals("Tree", question.getOptionD());
        assertEquals("B", question.getCorrectOption());
    }
}
```

Javadoc Comments

The **CompetitorList** class includes Javadoc-style comments to enhance code readability and maintainability. These comments follow the standard Javadoc format and provide detailed explanations of the class, its attributes, and methods.


PACKAGE CLASS USE TREE INDEX HELP
PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES
SEARCH
Search

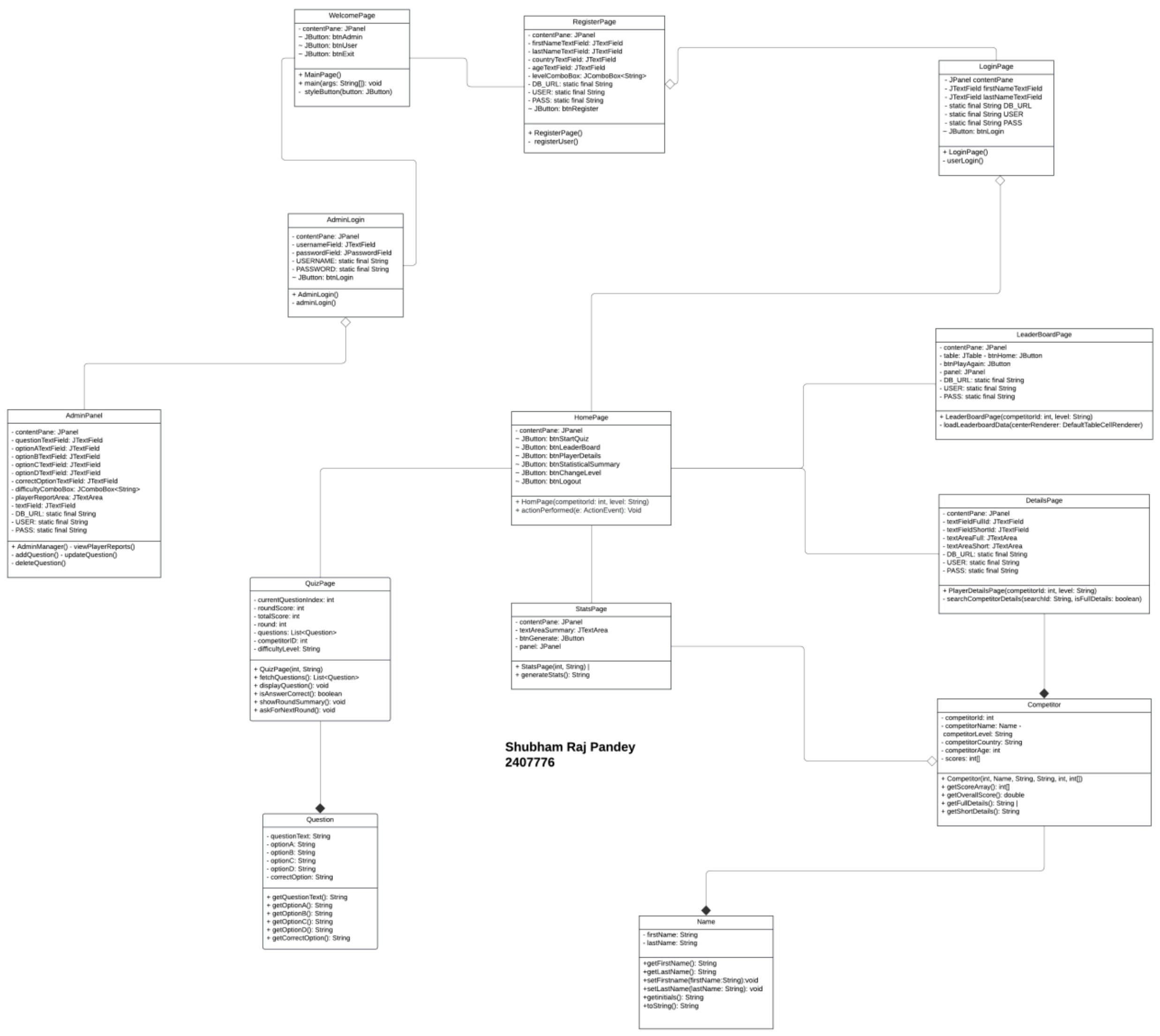
Package Assessment

package Assessment

Classes

Class	Description
AdminLogin	Admin login window for the application.
AdminPanel	Represents the admin panel for managing quiz questions and viewing player reports.
Competitor	Represents a competitor participating in the quiz competition.
DetailsPage	The DetailsPage class represents a GUI window for displaying competitor details.
HomePage	The Home class represents the main menu screen of the quiz application.
LeaderboardPage	The LeaderboardPage class is used to display the leaderboard of competitors based on their scores.
LoginPage	Represents the Login screen for the application.
Name	The Name class represents a person's full name, consisting of a first name and a last name.
Question	The Question class represents a single question in the quiz, along with its options and the correct answer.
QueTest	Test class for the `Question` class.
QuizPage	The QuizPage class handles the graphical user interface (GUI) and logic of the quiz game.
RegisterPage	The RegisterPage class creates a registration form for users to register in the database.
StatsPage	The StatsPage class represents a window in the application where statistical summaries of the competition are displayed.
WelcomePage	The Main1 class represents the main window for the application.

Class Diagram


WelcomePage
- contentPane: JPanel
~ JButton: btnAdmin
~ JButton: btnUser
~ JButton: btnExit
+ MainPage()
+ main(args: String[]): void
- styleButton(button: JButton)
RegisterPage
- contentPane: JPanel
- firstNameTextField: JTextField
- lastNameTextField: JTextField
- countryTextField: JTextField
- ageTextField: JTextField
- levelComboBox: JComboBox<String>
- DB_URL: static final String
- USER: static final String
- PASS: static final String
~ JButton: btnRegister
+ RegisterPage()
- registerUser()
LoginPage
- JPanel contentPane
- JTextField firstNameTextField
- JTextField lastNameTextField
- static final String DB_URL
- static final String USER
- static final String PASS
~ JButton: btnLogin
+ LoginPage()
- userLogin()
AdminLogin
- contentPane: JPanel
- usernameField: JTextField
- passwordField: JPasswordField
- USERNAME: static final String
- PASSWORD: static final String
~ JButton: btnLogin
+ AdminLogin()
- adminLogin()
LeaderBoardPage
- contentPane: JPanel
- table: JTable - btnHome: JButton
- btnPlayAgain: JButton
- panel: JPanel
- DB_URL: static final String
- USER: static final String
- PASS: static final String
+ LeaderBoardPage(competitorId: int, level: String)
- loadLeaderboardData(centerRenderer: DefaultTableCellRenderer)
AdminPanel
- contentPane: JPanel
- questionTextField: JTextField
- optionATextField: JTextField
- optionBTextField: JTextField
- optionCTextField: JTextField
- optionDTextField: JTextField
- correctOptionTextField: JTextField
- difficultyComboBox: JComboBox<String>
- playerReportArea: JTextArea
- textField: JTextField
- DB_URL: static final String
- USER: static final String
- PASS: static final String
+ AdminManager() - viewPlayerReports()
- addQuestion() - updateQuestion()
- deleteQuestion()
HomePage
- contentPane: JPanel
~ JButton: btnStartQuiz
~ JButton: btnLeaderBoard
~ JButton: btnPlayerDetails
~ JButton: btnStatisticalSummary
~ JButton: btnChangeLevel
~ JButton: btnLogout
+ HomPage(competitorId: int, level: String)
+ actionPerformed(e: ActionEvent): Void
DetailsPage
- contentPane: JPanel
- textFieldFullId: JTextField
- textFieldShortId: JTextField
- textAreaFull: JTextArea
- textAreaShort: JTextArea
- DB_URL: static final String
- USER: static final String
- PASS: static final String
+ PlayerDetailsPage(competitorId: int, level: String)
- searchCompetitorDetails(searchId: String, isFullDetails: boolean)
QuizPage
- currentQuestionIndex: int
- roundScore: int
- totalScore: int
- round: int
- questions: List<Question>
- competitorID: int
- difficultyLevel: String
+ QuizPage(int, String)
+ fetchQuestions(): List<Question>
+ displayQuestion(): void
+ isAnswerCorrect(): boolean
+ showRoundSummary(): void
+ askForNextRound(): void
StatsPage
- contentPane: JPanel
- textAreaSummary: JTextArea
- btnGenerate: JButton
- panel: JPanel
+ StatsPage(int, String) |
+ generateStats(): String
Competitor
- competitorId: int
- competitorName: Name -
competitorLevel: String
- competitorCountry: String
- competitorAge: int
- scores: int[]
+ Competitor(int, Name, String, String, int, int[])
+ getScoreArray(): int[]
+ getOverallScore(): double
+ getFullDetails(): String |
+ getShortDetails(): String
Shubham Raj Pandey
2407776
Question
- questionText: String
- optionA: String
- optionB: String
- optionC: String
- optionD: String
- correctOption: String
+ getQuestionText(): String
+ getOptionA(): String
+ getOptionB(): String
+ getOptionC(): String
+ getOptionD(): String
+ getCorrectOption(): String
Name
- firstName: String
- lastName: String
+getFirstName(): String
+getLastName(): String
+setFirstname(firstName:String):void
+setLastName(lastName: String): void
+getinitials(): String
+toString(): String

Test Case

Test Case	Input	Expected Output	Status
Question.getQuestionText()	Which data structure uses LIFO?	Which data structure uses LIFO?	Pass
question.getOptionA()	Queue	Queue	Pass
question.getOptionB()	Stack	Stack	Pass
question.getOptionC()	Linked List	Linked List	Pass
question.getOptionD	Tree	Tree	Pass
question.getCorrectOption()	B	B	Pass

Status Report

The application implements a quiz system that fulfills user registration, quiz participation, and performance metrics. Upon successful login by the user to the system, MySQL integrates data persistence. The admin panel facilitates the effective handling of quiz questions and competitor statistics. The application also builds on error handling and validation to perform well.

Minor enhancements such as improved UI design, increased security, and extended testing would improve the application. Overall though, it is finished and works as required by the core functionalities.

Known Bugs and Limitations

- Few Input Restrictions – Some text fields are not strictly validated causing unexpected data entry.
- Hard Coded Admin Credentials- Admin login uses static credentials which is unsafe and should use a database authentication system.
- Not Encrypted Passwords – The passwords are saved in plain text and tend to be insecure; just hashing it (for example, through BCrypt) would make it secure.
- Limited Error Message–Some error messages are vague and do not point to the fault exactly, adding to the difficulties for the user in debugging.
- Basic UI Design-The user interface is working, but it is not as attractive as it could be, nor is it fully responsive.
- No Multi-User Handling - The system does not currently support multiple admins with different access levels.
- Performance Issues on Large Data -- Application may hang slow with many competitors and quiz questions.
- With these limitations, however, it holds core functionality and therefore works for all standard use cases.

Conclusion

The setting up of the quiz application has more than amply satisfied its core requirements: user registration, participation in quizzes, score management, and the existence of an admin dashboard for performance monitoring. MySQL has been integrated with Java via JDBC, allowing seamless storage and retrieval of course data for smooth running.

Throughout the development cycle, various software engineering principles were utilized: structured design of classes, handling errors, and performing unit testing. Report and statistical summary implementations give very useful insights into user performance.

However, some areas that can still be improved include data validation, securing admin authentication, and refining the user interface for a better experience. Future improvement features could include a role-based access control mechanism, a wide variety of quiz question types, and database queries optimized for large dataset handling.

The project, in its entirety, is a well-structured software development project that successfully satisfies its objectives.