QuizApplication CourseWork Documentation

1.AIM

The purpose of this project is to create and put into use an assessment and quiz creation system that is both scalable and adaptable for usage in educational settings. This system records student performance, facilitates the construction of regular and revision quizzes, and generates statistical data based on quiz outcomes. The project guarantees the application of best practices in design and testing and conforms to the principles of object-oriented programming.

Key features include:

• **Factory Pattern**:
Factories oversee the development of quiz elements and guarantee accurate validation.

• **Interface-Based Design**:
By abstracting quiz operations into interfaces, this modular design makes future extension simple.

• **Defensive programming**:
where required, guarantees immutability and input validation.

• **Java Collections**:
Effective management of test results, guaranteeing accuracy and appropriate student monitoring.

• **JUnit Testing**:
Extensive unit tests to ensure stability.

2.INTRODUCTION

The development, administration, and assessment of student quizzes are all handled by the QuizApplication. It keeps track of students' quiz attempts, calculates the results, and gives students who don't pass the first time around the chance to retake the test. Object-oriented design techniques, such as abstraction, encapsulation, and polymorphism, are used in the construction of the application.

Key aspects:

• Encourages both routine and review quizzes.

• Monitors student performance and provides data based on quiz scores.

• Provides for both free-response and multiple-choice questions.

3. SYSTEM OVERVIEW

Without the need for a user interface, the technology facilitates the creation and assessment of quizzes and may be effortlessly included into more extensive instructional programs.

3.1 Quiz Functionality:

• Regular Quiz:
Creates quizzes at random from a pool of questions, guaranteeing original questions and no duplicates.

• Review Quiz:
Highlights questions that students mis answered on their first try.

• Eligibility:
After completing two standard quizzes, students may be eligible to take revision tests depending on their results.

3.2 Questions Types:

• Open-ended questions:
Provide precise responses but permit capitalisation and truncation of extraneous characters.

• Multiple-Choice Questions:
Provide two to four valid responses, with answers verified in any sequence or spacing.

3.3 Handling of Students

Each student's entire name and birthdate serve as a unique identifier. Based on performance, the system keeps track of quiz attempts and assigns a final score to each student.

3.4 Data Analysis

The apparatus monitors:• The final decision (PASS, FAIL, or Unknown).• Each student's total number of quiz attempts and results

3.5 Quiz Generation Example:

Methods:

- **generateQuiz(int numberOfQuestions)**
  Description: Generates a quiz with a specified number of questions selected randomly from the question pool.
  Parameters: numberOfQuestions: An integer representing the number of questions to include in the quiz.
  Returns: A Quiz object containing g the selected questions.

- **takeQuiz(Student student, Quiz quiz, List answers)**
  Description: Allows a student to take a quiz and evaluates their answers.
  Parameters:
    - student: A Student object representing the student taking the quiz.
    - quiz: A Quiz object representing the quiz being taken.
    - answers: A list of strings representing the answers provided by the student.
    -

  Returns: A double representing the student’s score as a percentage of correct answers.

4. PACKAGE STRUCTURE

4.1 User Interfaces

Defines core functionalities and provides flexibility for the application.

4.1.1 Question Interface:

Handles both free-response and multiple-choice questions.

Methods:

- **getQuestionText()**
  Description: Returns the text of the question.
  Returns: A String representing the question's formulation.

- **isAnswerCorrectOrNot(String answer)**
  Description: Checks if the provided single answer is correct for a free-response question.
  Parameters: answer: A String representing the student's answer.
  Returns: A boolean indicating whether the answer is correct (true) or not (false).

- **getCorrectAnswers()**
  Description: Returns the correct answer(s) for the question.
  Returns: A list of strings containing the correct answers.

4.1.2 Quiz Interface:

Specifies how to create and assess quizzes.

Methods:

- **getQuestions()**
  Description: Returns a list of questions included in the quiz.
  Returns: A List<Question> containing the questions in the quiz.

- **takeQuiz(List answers)**
  Description: Evaluates the student's answers to the quiz.
  Parameters: answers: A List<String> representing the student's answers.
  Returns: A double representing the student's score based on the correctness of their answers.

4.1.3 Student Interface:

Controls quiz attempts and student information.

- **getFullNameOfStudent()**
  Description: Returns the full name of the student.
  Returns: A String representing the student's full name.

- **getDateOfBirth()**
  Description: Returns the student's date of birth.
  Returns: A Date object representing the student's date of birth.

4.2 Implementations

Implements the system's main functionalities.

4.2.1 Multiple Choice Question:

Responds to enquiries with several valid responses.

Methods:

- **getQuestionText()**
  Description: Returns the text of the multiple-choice question.
  Returns: A String representing the question's text.

- **isAnswerCorrectOrNot(String answer)**
  Description: Checks if the provided single answer is correct.
  Parameters: answer: A String representing the student's answer.
  Returns: A boolean indicating whether the answer is correct (true) or not (false).

- **isAnswerCorrectOrNot(Set answers)**
  Description: Checks if the provided set of answers is correct for multiple-choice questions.
  Parameters: answers: A Set<String> representing the student's answers.
  Returns: A boolean indicating whether all provided answers are correct (true) or not (false).

- **getCorrectAnswers()**
  Description: Returns the correct answers for the multiple-choice question.
  Returns: A List<String> containing the correct answers.

4.2.2 FreeResponseQuestion:

Verifies questions with a single response.

Methods:

- **getQuestionText()**
  Description: Returns the text of the free-response question.
  Returns: A String representing the question's text.

- **isAnswerCorrectOrNot(String answer)**
  Description: Checks if the provided answer is correct. Trims leading and trailing whitespace and ignores case sensitivity.
  Parameters: answer: A String representing the student's answer.
  Returns: A boolean indicating whether the answer is correct (true) or not (false).

- **getCorrectAnswers()**
  Description: Returns the correct answer for the free-response question.

Returns: A List<String> containing the correct answer(s).

4.2.3 QuizSystem:
Oversees the creation, assessment, and monitoring of quiz statistics.

4.2.4 StudentStatistics:
Compiles statistics and keeps track of students' quiz efforts.

Methods:

- **addQuizScore(double score)**
  Description: Adds a quiz score to the student's performance record.
  Parameters: score: A double representing the quiz score.
  Returns: None.

- **getQuizAverageScore()**
  Description: Calculates and returns the student's average quiz score.
  Returns: A double representing the average score.

- **getQuizScores()**
  Description: Returns a list of all quiz scores for the student.
  Returns: A List<Double> containing the student's quiz scores.

- **getFinalVerdict()**
  Description: Provides a final verdict for the student based on their performance (PASS/FAIL/TBD).
  Returns: A String representing the final verdict.

4.3 Factories

Encapsulate object creation, ensuring that all components are created with proper validation

4.3.1 QuestionFactory:
Creates and manages different question types.

Methods:

- **createQuestion(String type, String questionText, List correctOptions)**
  Description: Creates and returns a Question object based on the specified type (free-response or multiple-choice).
  Parameters:
  - type: A String representing the type of question ("free" for free-response or "multiple" for multiple-choice).
  - questionText: A String representing the question text.

- correctOptions: A List<String> representing the correct answer(s).

Returns: A Question object representing the generated question.
Throws: IllegalArgumentException if the type is unknown or the correct options are invalid.

4.3.2 QuizFactory:
Handles quiz creation based on question pools.

Methods:

- **createQuiz(List questionPool, int numberOfQuestions)**
  Description: Creates and returns a Quiz object containing a subset of questions selected from the provided question pool.
  Parameters:
  - questionPool: A List<Question> representing the pool of available questions.
  - numberOfQuestions: An integer specifying the number of questions to include in the quiz.

  Returns: A Quiz object with the selected questions.
  Throws: IllegalArgumentException if the number of requested questions exceeds the size of the question pool.

4.3.3 StudentFactory:
Manages student object creation.

Methods:

- **createStudent(String firstName, String lastName, Date dateOfBirth)**
  Description: Creates and returns a Student object with the specified details.
  Parameters:
  - firstName: A String representing the student's first name.
  - lastName: A String representing the student's last name.
  - dateOfBirth: A Date representing the student's date of birth.

  Returns: A Student object with the specified details.
  Throws: IllegalArgumentException if any of the parameters are null or invalid.

5.CLASS AND INTERFACE DOCUMENTATION

5.1 Question Interface
Defines methods for handling quiz questions, including retrieving questions and validating answers.

5.2 Quiz Interface
Handles the generation of quizzes, quiz taking, and result validation.

5.3 Student Interface
Manages student information, including quiz attempts and results tracking.

Methods:

- **generateQuiz(int numberOfQuestions)**
  Description: Generates a quiz with a specified number of random questions.
  Parameters: numberOfQuestions: An integer specifying how many questions the quiz should contain.
  Returns: A Quiz object.
- **takeQuiz(Student student, Quiz quiz, List answers)**
  Description: Allows a student to take a quiz and evaluates their answers.
  Parameters:
  - student: A Student object representing the student.
  - quiz: A Quiz object containing the quiz questions.
  - answers: A List<String> representing the student’s answers.

  Returns: A double representing the score.

- **generateRevisionQuiz(Student student, int numberOfQuestions)**
  Description: Generates a revision quiz based on the student’s incorrect answers from previous attempts.
  Parameters:
  - student: A Student object representing the student.
  - numberOfQuestions: An integer specifying the number of questions for the revision quiz.

  Returns: A Quiz object containing the revision quiz.

- **getQuizAttempts(Student student)**
  Description: Returns the number of quiz attempts made by the specified student.
  Parameters: student: A Student object representing the student.
  Returns: An integer representing the number of quiz attempts.

- **generateStatistics(Student student)**
  Description: Generates and returns a report of the student’s quiz performance, including details such as the number of quiz attempts, revision attempts, quiz scores, and the final verdict (PASS/FAIL).

Parameters:student: A Student object representing the student for whom the statistics are being generated.

Returns: A String that includes the following information:

- The student’s full name.
- Number of quiz attempts.
- Number of revision attempts.
- Quiz scores (for each attempt).
- Final verdict (PASS, FAIL, or TBD).

6. KEY DESIGN CHOICES

6.1 Factory Pattern: Ensures controlled object creation for questions, quizzes, and students.

Description: The application uses the factory design pattern for the creation of key objects (such as Quiz, Question, and Student), ensuring encapsulation and reusability.

- **createStudent(String firstName, String lastName, Date dateOfBirth)**
  Description: A method in the StudentFactory to create a new student object.
  Parameters:
    - firstName: The first name of the student.
    - lastName: The last name of the student.
    - dateOfBirth: The date of birth of the student.

  Returns: A Student object.

6.2 Encapsulation: Protects critical data from unintended modifications.

Description: Critical data (such as quiz results and statistics) is encapsulated within appropriate classes like StudentStatistics, preventing unauthorised modifications.

6.3 Interfaces for Flexibility: Allows easy implementation of future features.

Description: Interfaces (Quiz, Question, Student) are used to abstract behaviours, enabling the system to be extended in the future with minimal changes to the core structure.

- **getQuestions()**
  Description: Abstracts the retrieval of a list of questions in a quiz.
  Returns: A list of Question objects.
- **isAnswerCorrectOrNot(String answer)**
  Description: Abstracts the validation of an answer provided for a quiz question.
  Parameters: answer: A String representing the answer to check.
  Returns: A boolean indicating if the answer is correct.

6.4 Defensive Programming: Includes validation to prevent errors during runtime.

6.5 Use of Java Collections: Efficiently stores and retrieves data.

7. UNIT TESTING DOCUMENTATION

- **JUnit Framework**: Used for comprehensive unit testing.
- **Mocking Dependencies**: Ensures isolated testing of components.
- **Edge Cases**: Validates system behaviour with unexpected or invalid inputs.

Methods:

- **testStudentFactory()**
  Description: Verifies that the StudentFactory correctly creates a student with valid input.
  Parameters: None.
  Returns: None. Verifies that the student's full name and date of birth are correct.
- **testQuestionFactoryFreeResponse()**
  Description: Tests that the QuestionFactory creates a valid free-response question and correctly evaluates the student's answer.
  Parameters: None.
  Returns: None. Verifies that the correct answer is recognized and incorrect answers are rejected.
- **testQuestionFactoryMultipleChoice()**
  Description: Tests the creation of a multiple-choice question using the QuestionFactory and checks for correct and incorrect answers.
  Parameters: None.
  Returns: None. Verifies that multiple correct answers are recognized and wrong answers are rejected.
- **testMaxQuizAttempts()**
  Description: Ensures that a student is restricted to a maximum of two quiz attempts and an exception is thrown on the third attempt.
  Parameters: None.
  Returns: None. Verifies that the system correctly throws an exception for additional attempts beyond the allowed limit.

8. CONCLUSION

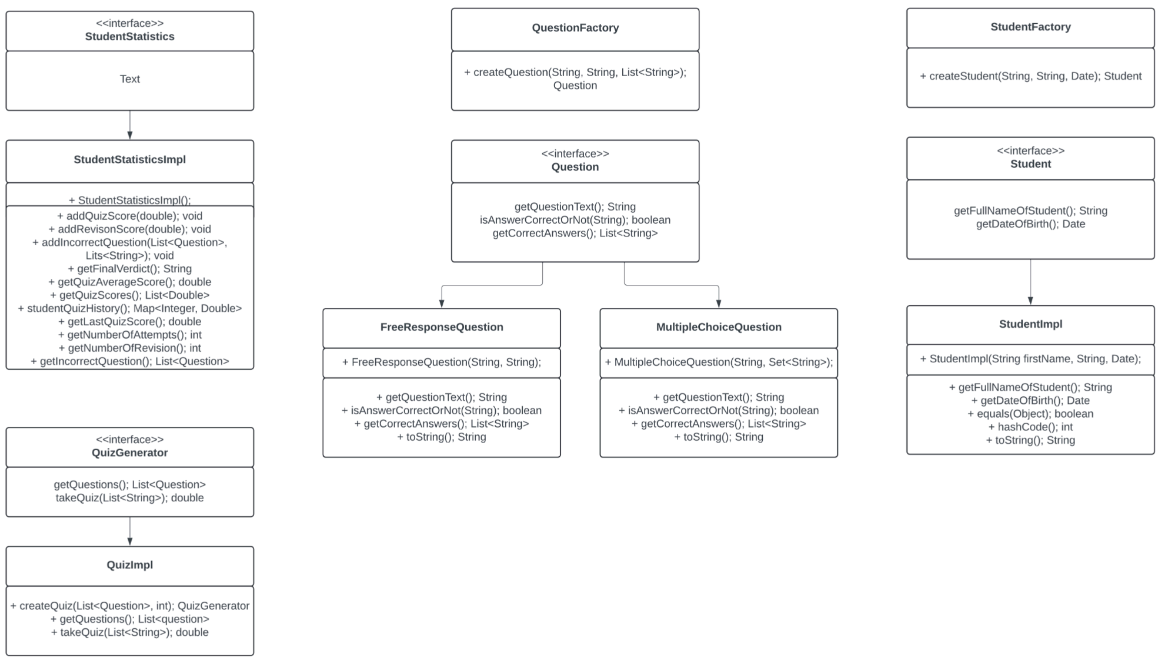
The QuizApplication provides a robust and scalable solution for quiz management in educational settings. The use of design patterns, modular architecture, and thorough testing ensures that the system can be easily integrated into larger applications and extended with future features.

Methods:

- **getQuizAverageScore()**
  Description: Returns the average quiz score for a student based on all their quiz attempts.
  Returns: A double representing the average score.

- **getFinalVerdict()**
  Description: Provides the final evaluation for the student, based on their quiz performance.
  Returns: A String representing the verdict (PASS, FAIL, or TBD).

- **generateStatistics(Student student)**
  Description: Compiles a student's performance into a string, including their scores and quiz history.
  Parameters: student: A Student object representing the student whose statistics are being generated.
  Returns: A String containing the student's performance report.


<<interface>> StudentStatistics
Text
StudentStatisticsImpl
+ StudentStatisticsImpl();
+ addQuizScore(double); void
+ addRevisonScore(double); void
+ addIncorrectQuestion(List<Question>, Lits<String>); void
+ getFinalVerdict(); String
+ getQuizAverageScore(); double
+ getQuizScores(); List<Double>
+ studentQuizHistory(); Map<Integer, Double>
+ getLastQuizScore(); double
+ getNumberOfAttempts(); int
+ getNumberOfRevision(); int
+ getIncorrectQuestion(); List<Question>
<<interface>> QuizGenerator
getQuestions(); List<Question>
takeQuiz(List<String>); double
QuizImpl
+ createQuiz(List<Question>, int); QuizGenerator
+ getQuestions(); List<question>
+ takeQuiz(List<String>); double
QuestionFactory
+ createQuestion(String, String, List<String>); Question
<<interface>> Question
getQuestionText(); String
isAnswerCorrectOrNot(String); boolean
getCorrectAnswers(); List<String>
FreeResponseQuestion
+ FreeResponseQuestion(String, String);
+ getQuestionText(); String
+ isAnswerCorrectOrNot(String); boolean
+ getCorrectAnswers(); List<String>
+ toString(); String
MultipleChoiceQuestion
+ MultipleChoiceQuestion(String, Set<String>);
+ getQuestionText(); String
+ isAnswerCorrectOrNot(String); boolean
+ getCorrectAnswers(); List<String>
+ toString(); String
StudentFactory
+ createStudent(String, String, Date); Student
<<interface>> Student
getFullNameOfStudent(); String
getDateOfBirth(); Date
StudentImpl
+ StudentImpl(String firstName, String, Date);
+ getFullNameOfStudent(); String
+ getDateOfBirth(); Date
+ equals(Object); boolean
+ hashCode(); int
+ toString(); String

Samuel Pillai Sathiyamoorthy
s.p.sathiyamoorthy2@newcastle.ac.uk
Student.no: 240503828