Technical Guide

Quiz Manager

By:
Ashwini Priya Bodda

July 03, 2019

1. Introduction

1.1. Project Background

This project is a console application managing digital quizzes - preparation and execution. This document is prepared by Ashwini Priya Bodda for their Fundamental Java Project in the Computer Science Master's Program at L'École Pour l'Informatique et les Techniques Avancées (EPITA).

1.2. Project Overview

In this project, user can login as a student, take a test and get their marks at the end of the quiz. In addition, user can login as a teacher, create questions and quizzes.

1.3. Project Scope

In Scope :

1. Automatically assembles quizzes using multiple-choice questions
2. Displaying marks for multiple choice questions
3. Creating data access object with CRUD methods
4. Create a configurable properties file for the application.

Out of Scope :

5. Incorporating Associative questions into the GUI
6. Exporting quizzes to PDF
7. Using an algorithm to arrange quizzes
8. Stylizing the GUI
9. Creating different user accounts other than Student
10. Creating user accounts with logging in
11. Providing real questions for the Quiz Manager application

1.4. Project Dependencies

In order to run the program, the following steps are required:

- Install h2 JDBC
- Install Java 12 with eclipse
- Create tables in database with all fields.
- Add questions at every level(1,2,3).
- Input only valid options at every step.
- Add to build path : h2-1.4.199.jar
- Add to build path : log4j.properties
- Add to build path : log4j-1.2.16.jar

1.5. Acronyms and abbreviations

Acronyms	Meaning
CRUD	(Create, Read, Update, Delete) Functions that are implemented in relational database applications
DAO	(Data Access Object) Service class for the application to communicate with the H2 database
UML	(Unified Modeling Language) a standard way to visualize the design of a system
MCQ	(Multiple Choice Question) Question with enumerated possible answers, implemented with radial buttons in our application
JDBC	(Java Database Connectivity) application programming interface (API) for the programming language Java

2. Business Requirements

BR#	Requirement Description
1	Create a quiz based on user's specifications (topics)
2	Use questions stored in database (using h2 database)
3	Allow a user to take a quiz
4	Correct MCQ questions and display result at the end of the evaluation
5	Use CRUD operations on a question (create, read, update, or delete a question)
6	Use existing default credentials to take a quiz

3. System Specifications

3.1. Functional Requirements

BR#	Implementation

1	**Student can perform their quiz when instantiating the Quiz object, which takes a list of topics as one of the parameters.**
	
	//Instantiating Quiz class
	`public class Quiz {`
	
	`static Logger log = Logger.getLogger("Quiz.class");`
	
	`static Connection connection = null;`
	`static Scanner in = new Scanner(System.in);`
	`static User user = null;`
	//Now topic selection
	`QuestionDAO questionDAO = new QuestionDAO();`
	
	`log.info("Do you want to continue as a student......?  Y/N");`
	
	`if (QuizUtil.getBoolean(in.nextLine())) {`
	`Student student = Student.collectStudent();`
	`log.info("Select a topic..");`
	
	`Map<Integer, String> result = new HashMap<>();`
	`try {`
	`result = questionDAO.getTopics();`
	`log.info(result.toString());`
	`Iterator<Integer> itr = result.keySet().iterator();`
	`while (itr.hasNext()) {`
	`int key = itr.next();`
	`log.info(key + ".  " + result.get(key));`
	`}`
	`} catch (SQLException e) {`
	`log.error(e);`
	`}`

2	**Students can use the questions that are already used and stored in the database.**

```
while (readlevel);
                List<Question> questions = 
questionDAO.getQuestionsByTopicAndLevel(result.get(key), 
level);

                int correctAnswers = 0;
                String difficultyLevel;
                if (!questions.isEmpty()) {
                      int i = 0;
                      do {
                      Question question = questions.get(i++);

      log.info(question.getDescription());
                           int j = 1;
                           for (Choice choice : 
question.getOptions()) {
                                 log.info((j++) + ". " + 
choice.getValue());
                           }
                           log.info("Enter your option (option 
number): ");
                           boolean readChoice = true;
                           while (readChoice) {
                                 try {
                                      int userChoice = 
Integer.parseInt(in.nextLine());
                                      Choice choice = 
questionDAO.getChoices(question.getId());
                                      if 
(choice.getValue().equals(question.getOptions().get(userChoice 
- 1).getValue())) {
                                            correctAnswers++;
                                      }
                                      readChoice = false;
                                 } catch (Exception e) {
                                      log.info("Invalid 
option, Please try again...");
                                 }
                           }

                           difficultyLevel = 
question.getLevel();
                      }
```

3	**Student can take a quiz by entering Student ID, Name, topics, difficulty level of questions.**

```
QuestionDAO questionDAO = new QuestionDAO();

           log.info("Do you want to continue as a 
student......?  Y/N");

           if (QuizUtil.getBoolean(in.nextLine())) {
                Student student = Student.collectStudent();
                log.info("Select a topic..");

                Map<Integer, String> result = new HashMap<>();
                try {
                      result = questionDAO.getTopics();
                      log.info(result.toString());
                      Iterator<Integer> itr = 
result.keySet().iterator();
                      while (itr.hasNext()) {
                           int key = itr.next();
                           log.info(key + ".  " + 
result.get(key));
                      }
                } catch (SQLException e) {
                      log.error(e);
                }

                log.info("Enter topic number");
                boolean readTopicNumber = true;
                int key = 0;
                do {
                      try {
                           key = 
Integer.parseInt(in.nextLine());
                           readTopicNumber = false;
                      } catch (Exception e) {
                           log.info("Invalid input, Please try 
again...");
                      }
                } while (readTopicNumber);

                log.info("Enter difficulty level (1. Easy 2. 
Medium 3. Hard) : ");
                boolean readlevel = true;
                int level = 0;
                do {
                      try {
                            level = 
Integer.parseInt(in.nextLine());
                            readlevel = false;
                      } catch (Exception e) {
                            log.info("Invalid input, Please try 
again...");
                      }
                }
```

4 **Student can have his quiz evaluated and results displayed at end of the quiz.**

```
while (readlevel);
                List<Question> questions = 
questionDAO.getQuestionsByTopicAndLevel(result.get(key), 
level);

                int correctAnswers = 0;
                String difficultyLevel;
                if (!questions.isEmpty()) {
                      int i = 0;
                      do {
                            Question question = 
questions.get(i++);

     log.info(question.getDescription());
                            int j = 1;
                            for (Choice choice : 
question.getOptions()) {
                                  log.info((j++) + ". " + 
choice.getValue());
                            }
                            log.info("Enter your option (option 
number): ");
                            boolean readChoice = true;
                            while (readChoice) {
                                  try {
                                        int userChoice = 
Integer.parseInt(in.nextLine());
                                        Choice choice = 
questionDAO.getChoices(question.getId());
                                        if
(choice.getValue().equals(question.getOptions().get(userChoice
- 1).getValue())) {
                                            correctAnswers++;
                                      }
                                      readChoice = false;
                                 } catch (Exception e) {
                                      Log.info("Invalid
option, Please try again...");
                                 }
                           }

                           difficultyLevel =
question.getLevel();
                      } while (i < questions.size());
                      Log.info("Student ID : " +
student.getStudentId() + ", Student Name : " +
student.getName()
                                 + " Correct answers = " +
correctAnswers + ", Total questions = " + questions.size()
                                 + ", Topic = " +
result.get(key) + ", Difficulty Level = " + difficultyLevel);
                } else {
                      Log.info("No questions available with
selected options ");
                }
                }
```

5 **Students can search questions by difficulty and topics.**

```
            log.info("Enter topic number");
            boolean readTopicNumber = true;
            int key = 0;
            do {
                try {
                    key = 
Integer.parseInt(in.nextLine());
                    readTopicNumber = false;
                } catch (Exception e) {
                    log.info("Invalid input, Please try 
again...");
                }
            } while (readTopicNumber);

            log.info("Enter difficulty level (1. Easy 2. 
Medium 3. Hard) : ");
            boolean readlevel = true;
            int level = 0;
            do {
                try {
                    level = 
Integer.parseInt(in.nextLine());
                    readlevel = false;
                } catch (Exception e) {
                    log.info("Invalid input, Please try 
again...");
                }
            } while (readlevel);
            List<Question> questions = 
questionDAO.getQuestionsByTopicAndLevel(result.get(key), 
level);

            int correctAnswers = 0;
            String difficultyLevel;
            if (!questions.isEmpty()) {
                int i = 0;
                do {
                    Question question = 
questions.get(i++);

    log.info(question.getDescription());
                    int j = 1;
                    for (Choice choice : 
question.getOptions()) {
                        log.info((j++) + ". " +
choice.getValue());
                            }
                            Log.info("Enter your option (option
number): ");
                            boolean readChoice = true;
                            while (readChoice) {
                                  try {
                                        int userChoice =
Integer.parseInt(in.nextLine());
                                        Choice choice =
questionDAO.getChoices(question.getId());
                                        if
(choice.getValue().equals(question.getOptions().get(userChoice
- 1).getValue())) {
                                              correctAnswers++;
                                        }
                                        readChoice = false;
                                  } catch (Exception e) {
                                        Log.info("Invalid
option, Please try again...");
                                  }
                            }

                            difficultyLevel =
question.getLevel();
                      }
```

6 **Admin can create, read, update, and delete a question.**

```
public int insertQuestion(Question questions) throws
SQLException {

           PreparedStatement statement = null;
           try {

                statement =
connection.prepareStatement(Queries.INSERTINTOQUESTIONS,
Statement.RETURN_GENERATED_KEYS);
                statement.setString(1,
questions.getDescription());
                statement.setString(2, questions.getTopic());
                statement.setString(3, questions.getLevel());
                statement.executeUpdate();

                try (ResultSet generatedKeys =
statement.getGeneratedKeys()) {
                      if (generatedKeys.next()) {
                           log.info("Question added
successfully...");

     questions.setId(generatedKeys.getInt(1));
                      } else {
                           throw new SQLException("Creating
user failed, no ID obtained.");
                      }
                }

                List<Choice> options = questions.getOptions();
                OptionsDAO optionsDAO = new
OptionsDAO(connection);
                for (Choice option : options) {
                      option.setQuestionId(questions.getId());
                      optionsDAO.insertOption(option);
                }

           } catch (SQLException e) {
                log.error(e);
           } finally {
                DatabaseConnection.close(connection,
statement, null);

           }
           return 0;
     }
```

```
public boolean deleteQuery(int id) {

		PreparedStatement statement = null;
		try {
			if (new 
OptionsDAO(connection).deleteChoice(id)) {
				connection.commit();
				statement = 
connection.prepareStatement(Queries.DELETEQUESTION);
				statement.setInt(1, id);
				statement.toString();
				statement.execute();
				return true;
			}
		} catch (Exception e) {
			log.error(e);
		} finally {

			try {
				DatabaseConnection.close(connection, 
statement, null);
			} catch (SQLException e) {
				log.error(e);
			}

		}

		return false;
	}

	/**
	 * first deleting existing options and then adding new 
option
	 * @param question
	 * @return
	 */
public static  Question collectQuestion(Scanner in) {
		Question question = new Question();

		Log.info("Enter the topic");
		question.setTopic(in.nextLine());

		Log.info("Enter dificulty Level (1,2,3) : ");
		question.setLevel(in.nextLine());

		Log.info("Enter a Question");
		question.setDescription(in.nextLine());
```

	```
	log.info("How many choices do u want to add ");
	boolean read = true;
	int choiceCount = 0;
	while(read) {
	try {
	choiceCount =
	Integer.parseInt(in.nextLine());
	read = false;
	}
	catch(NumberFormatException ex) {
	log.info("Invalid input, Please try
	again...");
	}
	}
	boolean hasValue = false;
	for (int i = 0; i < choiceCount; i++) {
	Choice option1 = new Choice();
	log.info("Enter  choice");
	option1.setValue(in.nextLine());

	log.info("is valid choice Y/N");
	boolean isValid =
	QuizUtil.getBoolean(in.nextLine());
	option1.setValid(isValid);
	question.setOptions(option1);
	if(isValid) {
	hasValue = isValid;
	}
	}

	if(!hasValue) {
	log.info("please re-enter a qestion with valid
	ans");
	log.info("Terminating......");
	System.exit(200);
	}
	return question;

	}

	}
	static List<Question> displayQuestions() {
	List<Question> questionList = new
	QuestionDAO().getAllByTopic("all");
	int index1 = 0;
	for (Question question : questionList) {
	```

	```
	                *log*.info((index1++)+" " + "Topic - " +
	question.getTopic()+" - - " + question.getDescription());
	            }

	            return questionList;
	      }
	 }
	```

7	**Admin can use default credentials(admin : admin) to manage quiz.**

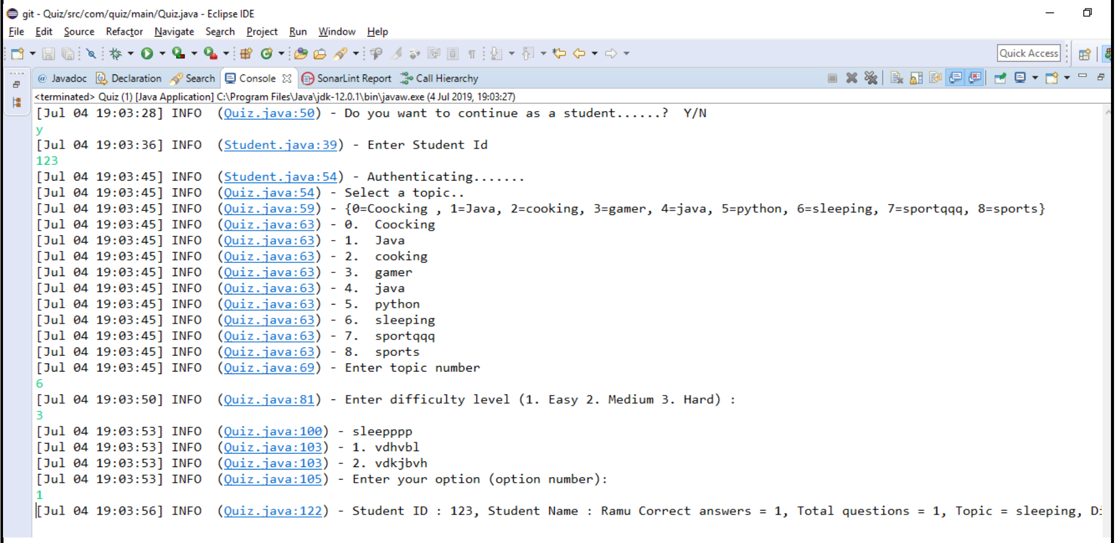
```
public static boolean checkUser(String name, String password)
{
            try {
                  user = new UserDAO().getUserByName(name,
password);
            } catch (SQLException e) {
                  log.error(e);
            }

            if (user == null) {
                  log.info("name or password enterd is wrong");
                  log.info("To re-try press Y/N");
                  return !QuizUtil.getBoolean(in.nextLine());
            } else {
                  return true;
            }
      }
```

```
public class User {

	int id;
	String name;
	String password;

	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
```

8	**Student can run the Java application using console.** 

git - Quiz/src/com/quiz/main/Quiz.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Javadoc Declaration Search Console SonarLint Report Call Hierarchy

```
<terminated> Quiz (1) [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (4 Jul 2019, 19:03:27)
[Jul 04 19:03:28] INFO  (Quiz.java:50) - Do you want to continue as a student......?  Y/N
y
[Jul 04 19:03:36] INFO  (Student.java:39) - Enter Student Id
123
[Jul 04 19:03:45] INFO  (Student.java:54) - Authenticating.......
[Jul 04 19:03:45] INFO  (Quiz.java:54) - Select a topic..
[Jul 04 19:03:45] INFO  (Quiz.java:59) - {0=Coocking , 1=Java, 2=cooking, 3=gamer, 4=java, 5=python, 6=sleeping, 7=sportqqq, 8=sports}
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 0.  Coocking
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 1.  Java
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 2.  cooking
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 3.  gamer
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 4.  java
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 5.  python
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 6.  sleeping
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 7.  sportqqq
[Jul 04 19:03:45] INFO  (Quiz.java:63) - 8.  sports
[Jul 04 19:03:45] INFO  (Quiz.java:69) - Enter topic number
6
[Jul 04 19:03:50] INFO  (Quiz.java:81) - Enter difficulty level (1. Easy 2. Medium 3. Hard) :
3
[Jul 04 19:03:53] INFO  (Quiz.java:100) - sleepppp
[Jul 04 19:03:53] INFO  (Quiz.java:103) - 1. vdhvbl
[Jul 04 19:03:53] INFO  (Quiz.java:103) - 2. vdkjbvh
[Jul 04 19:03:53] INFO  (Quiz.java:105) - Enter your option (option number):
1
[Jul 04 19:03:56] INFO  (Quiz.java:122) - Student ID : 123, Student Name : Ramu Correct answers = 1, Total questions = 1, Topic = sleeping, D:
```

3.2. Non-functional Requirements

3.2.1. The program shall be able to run on Eclipse console.

3.2.2. The program shall be able to read a configuration property set from a file on the filesystem which will avoid hardcoded parameters.

3.2.3. The data is stored in h2 databases.

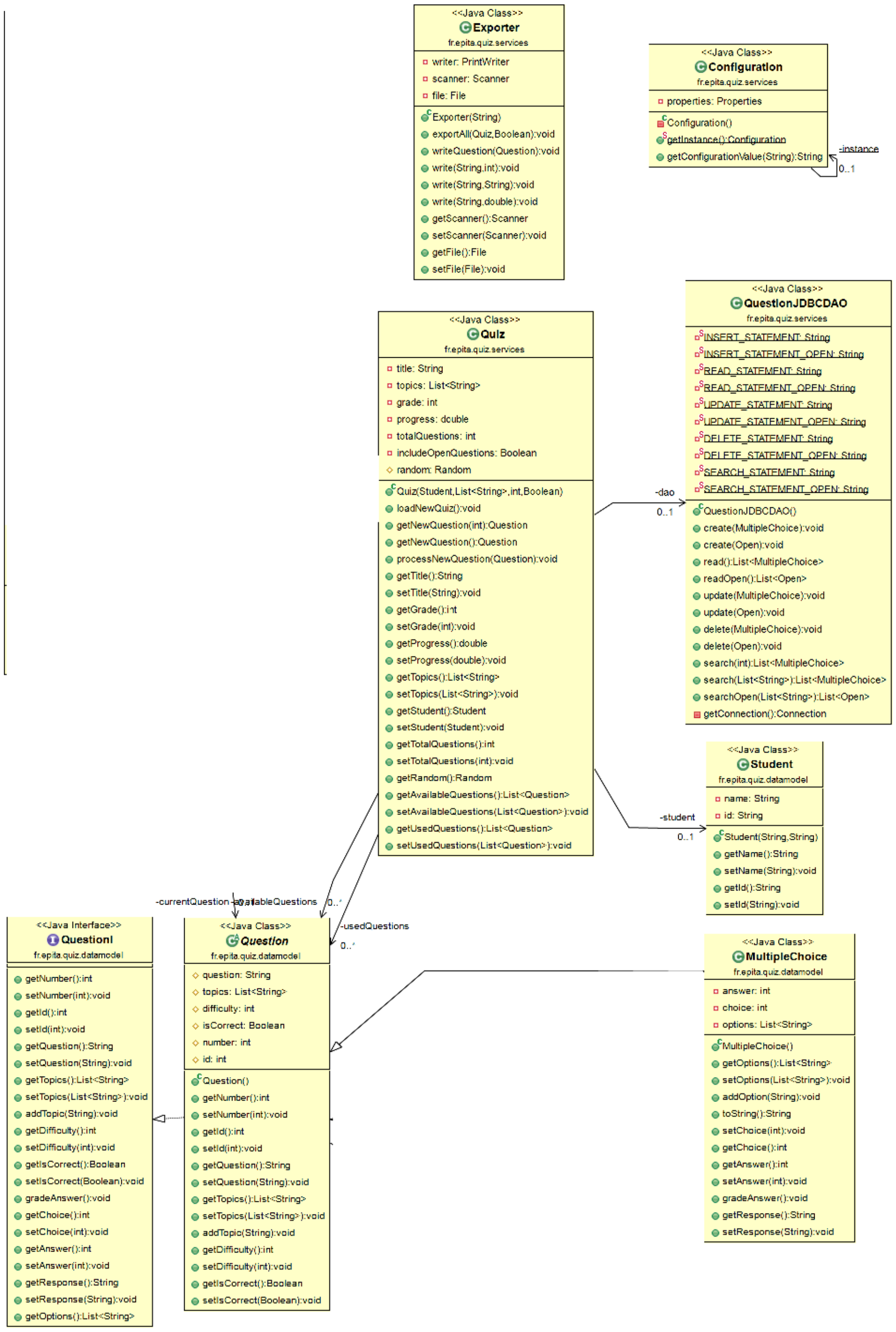
4. User Interface Design

Refer to the User Guide.

5. Hardware Requirements

#	**Hardware**	**Requirement**
1	Operating System	Compatible with Windows, Mac OS X, Linux
2	RAM	Minimum required 124MB
3	Disk Space	Minimum required 124MB
4	Processor	64-bit, four-core, 2.5 GHz minimum per core

UML Class Diagram



<<Java Class>>
Exporter
fr.epita.quiz.services
writer: PrintWriter
scanner: Scanner
file: File
Exporter(String)
exportAll(Quiz,Boolean):void
writeQuestion(Question):void
write(String,int):void
write(String,String):void
write(String,double):void
getScanner():Scanner
setScanner(Scanner):void
getFile():File
setFile(File):void
<<Java Class>>
Configuration
fr.epita.quiz.services
properties: Properties
Configuration()
getInstance():Configuration
getConfigurationValue(String):String
-instance
0..1
<<Java Class>>
QuestionJDBCDAO
fr.epita.quiz.services
INSERT_STATEMENT: String
INSERT_STATEMENT_OPEN: String
READ_STATEMENT: String
READ_STATEMENT_OPEN: String
UPDATE_STATEMENT: String
UPDATE_STATEMENT_OPEN: String
DELETE_STATEMENT: String
DELETE_STATEMENT_OPEN: String
SEARCH_STATEMENT: String
SEARCH_STATEMENT_OPEN: String
QuestionJDBCDAO()
create(MultipleChoice):void
create(Open):void
read():List<MultipleChoice>
readOpen():List<Open>
update(MultipleChoice):void
update(Open):void
delete(MultipleChoice):void
delete(Open):void
search(int):List<MultipleChoice>
search(List<String>):List<MultipleChoice>
searchOpen(List<String>):List<Open>
getConnection():Connection
<<Java Class>>
Quiz
fr.epita.quiz.services
title: String
topics: List<String>
grade: int
progress: double
totalQuestions: int
includeOpenQuestions: Boolean
random: Random
Quiz(Student,List<String>,int,Boolean)
loadNewQuiz():void
getNewQuestion(int):Question
getNewQuestion():Question
processNewQuestion(Question):void
getTitle():String
setTitle(String):void
getGrade():int
setGrade(int):void
getProgress():double
setProgress(double):void
getTopics():List<String>
setTopics(List<String>):void
getStudent():Student
setStudent(Student):void
getTotalQuestions():int
setTotalQuestions(int):void
getRandom():Random
getAvailableQuestions():List<Question>
setAvailableQuestions(List<Question>):void
getUsedQuestions():List<Question>
setUsedQuestions(List<Question>):void
-dao
0..1
-student
0..1
<<Java Class>>
Student
fr.epita.quiz.datamodel
name: String
id: String
Student(String,String)
getName():String
setName(String):void
getId():String
setId(String):void
-currentQuestion
-availableQuestions
0..*
-usedQuestions
0..*
<<Java Interface>>
QuestionI
fr.epita.quiz.datamodel
getNumber():int
setNumber(int):void
getId():int
setId(int):void
getQuestion():String
setQuestion(String):void
getTopics():List<String>
setTopics(List<String>):void
addTopic(String):void
getDifficulty():int
setDifficulty(int):void
getIsCorrect():Boolean
setIsCorrect(Boolean):void
gradeAnswer():void
getChoice():int
setChoice(int):void
getAnswer():int
setAnswer(int):void
getResponse():String
setResponse(String):void
getOptions():List<String>
<<Java Class>>
Question
fr.epita.quiz.datamodel
question: String
topics: List<String>
difficulty: int
isCorrect: Boolean
number: int
id: int
Question()
getNumber():int
setNumber(int):void
getId():int
setId(int):void
getQuestion():String
setQuestion(String):void
getTopics():List<String>
setTopics(List<String>):void
addTopic(String):void
getDifficulty():int
setDifficulty(int):void
getIsCorrect():Boolean
setIsCorrect(Boolean):void
<<Java Class>>
MultipleChoice
fr.epita.quiz.datamodel
answer: int
choice: int
options: List<String>
MultipleChoice()
getOptions():List<String>
setOptions(List<String>):void
addOption(String):void
toString():String
setChoice(int):void
getChoice():int
getAnswer():int
setAnswer(int):void
gradeAnswer():void
getResponse():String
setResponse(String):void