|  |
| --- |
|  |
| The Question Bank Project |
| CS12320 |

|  |
| --- |
| Nia Hawkins [nih25@aber.ac.uk]  4-21-2024 |

Contents

[Introduction 2](#_Toc164603262)

[Functional requirements 2](#_Toc164603263)

[Design 2](#_Toc164603264)

[Testing 3](#_Toc164603265)

[Screenshots 5](#_Toc164603266)

[Evaluation 10](#_Toc164603267)

# Introduction

This project has involved building a program that allows teachers to create banks of questions, and students to take quizzes on the questions. I have created a text-based command line interface which allows the user to carry out the 10 functional requirements.

I have also implemented additional functionality, allowing ….

This document details the development and testing of the program and evaluates how successfully it meets the requirements.

## Use Case Diagram

A diagram of a question

Description automatically generated

Figure 1: Use case diagram for the question bank.

# Design

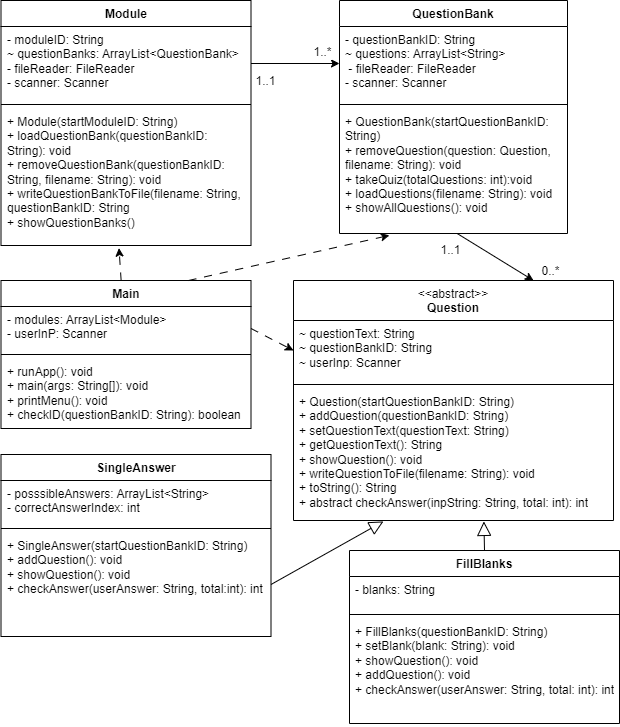
Figure 2 is the initial class diagram created to plan the project. As a result, it is missing some classes including in the final implementation. An updated version is shown below.

Figure 2: The initial class diagram.

A screenshot of a computer screen

Description automatically generated

Figure 3: The updated class diagram.

Main

Figure 4: The updated class diagram.

This is the main class from which the program is run. main creates an instance of the Main class, and runApp shows the menu options and prompts the user to input a menu option, until the user choses to quit the application.

Menu

This class handles the text-based menu functionality of the program. It determines which classes and methods to initialise, and call based on the users input.

Module

This class handles a collection of question banks, stored in an ArrayList questionBanks. The class has a constructor to initialise module objects. removeQuestionBank() deletes a question bank from the program, by removing it from the ArrayList and the database using FileReader, FileWriter and BufferedWriter. loadQuestionBanks() reads each question bank from the database into questionBanks ready to be displayed to the user. This method also implements a hash to remove duplicate question banks in the ArrayList after the file has been read, as the format of storing data results in duplication of the question bank identifiers. showQuestionBanks() outputs all of the question banks stored in questionBanks. writeBankToFile() saves a new, empty question bank to the database. The method checks that the bank is not already in the database, and if not, it is appended to the file.

QuestionBank

This class handles many questions, stored in an ArrayList questions. The class has a constructor to initialise objects. takeQuiz() allows the user to take a quiz. The method allows the user to navigate questions when possible, initialises the appropriate question objects and calls the other methods that handle specific aspects of the quiz, which makes the code more maintainable. handleQuizQuestion() displays a quiz question to the user, along with their answer if they have inputted one, and calls the method to check the answer. showResults() shows the users score, number of unanswered questions and the time taken to complete the quiz. loadQuestions() reads the questions from the file into questions. removeQuestions() deletes a question from the file, using FileReader, FileWriter and BufferedWriter. The question is modified so an empty question bank remains in the file. showAllQuestions() outputs all questions in questions with a question number.

Question

This class is an abstract class and provides methods used in specialised subclasses. The class has a constructor to initialise objects. addQuestion() adds a new question to the file, but is implemented in subclasses. setQuestionText() sets the question text of a question. getQuestionText() returns the question text. setPossibleAnswers() sets the answer options fr a question from a String that has been read from the database. showQuestion() shows a question, with its question number. writeQuestionToFile() saves a question to the database, by first checking if an empty question bank exists to edit, and then writing the question to the file in the appropriate place. toString() returns a questions attributes separated by “;;” to be written to the database. checkAnswer() is an abstract method which will check the users quiz question answer.

QuestionType

This is an enum which specifies the different question types.

FillBlanks, SingleAnswer, TrueFalse

These classes handle the fill-the-blanks, single answer, and true-false question types respectively, and are subclasses of Questions. They 3 overloaded constructors used in different scenarios. showQuestion() shows a question for a quiz. addQuestion() adds a new question to the database. checkAnswer() checks an inputted answer to a quiz question and returns an updated score.

## Pseudocode

METHOD takeQuiz(totalQuestions: integer)

Questions is LIST OF Question

quizQuestions is LIST OF Question

total is integer

currentType is QuestionType

userAnswers is LIST OF String

questionIndex is integer

inpString is String

firstQuestion is boolean

startTime is long

// Randomize question order

Collections.shuffle(questions)

quizQuestions = new ArrayList<Question>()

IF questions.size() > totalQuestions THEN

trimmedList = questions.subList(0, totalQuestions)

quizQuestions.addAll(trimmedList)

END IF

total = 0

questionIndex = 0

firstQuestion = TRUE

startTime = currentTimeMillis()

WHILE TRUE DO

IF NOT firstQuestion THEN

/\*

Navigate questions

b - back

n - next

s - submit

q - quit

\*/

INPUT inpString

IF inpString.equals("b") AND questionIndex > 0 THEN

questionIndex = questionIndex - 1

ELSE IF inpString.equals("b") AND questionIndex == 0 THEN

OUTPUT "This is the first question, enter a different menu option"

CONTINUE

ELSE IF inpString.equals("n") AND questionIndex < quizQuestions.size() - 1 THEN

questionIndex = questionIndex + 1

ELSE IF inpString.equals ("n") AND questionIndex >= quizQuestions.size() - 1 THEN

OUTPUT "This is the last question, enter a different menu option"

CONTINUE

ELSE IF inpString.equals ("q") THEN

BREAK

ELSE IF inpString.equals("s") THEN

endTime = currentTimeMillis()

timeTaken = endTime - startTime

showResults(total, quizQuestions.size(), userAnswers, timeTaken)

BREAK

ELSE

OUTPUT "Invalid menu option entered"

CONTINUE

END IF

END IF

// Get and output the question

currentType = quizQuestions.get(questionIndex).questionType

question = SelectQuestion.initialseQuestion(quizQuestions.get(questionIndex).questionBankID,

currentType, quizQuestions.get(questionIndex).questionText,

quizQuestions.get(questionIndex).possibleAnswers,

quizQuestions.get(questionIndex).answerIndex)

// Show the question and the user's answer if they have inputted an answer

handleQuizQuestion(question, questionIndex, userAnswers, total)

// Show navigation options

OUTPUT "b - back n - next s - submit quiz q - quit"

firstQuestion = FALSE

END WHILE

END METHOD

METHOD handleQuizQuestion(Question question, int questionIndex, ArrayList<String> userAnswers, int score

question.showQuestion(questionIndex + 1)

IF userAnswers.size() > questionIndex THEN

OUTPUT "Your answer: " userAnswers.get(questionIndex)

END IF

OUTPUT "If you are unsure, press enter to enter your answer as empty"

INPUT inpString

userAnswers.add(questionIndex, inpString)

END METHOD

# Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Requirement | Description | Inputs | Expected outputs | Pass/Fail | Comments |
|  | FR1 | Create empty question bank | Enter question bank identifier “CS12320:0004” | “New question bank added”, “New question bank saved” | Pass |  |
|  | FR2a | Add new single choice questions to a question bank | Enter question bank identifier, question, options, and the index of the correct answer. | Show the question and output “New question saved” | Pass |  |
|  | FR2b | Add new fill the blanks questions to a question bank | Enter question bank identifier, question and the missing text. | “New question saved” | Pass |  |
|  | FR3 | List questions from an existing question bank | Enter question bank identifier | Show all questions in the question bank |  |  |
|  | FR3 | Remove questions from a question bank |  |  |  |  |
|  | FR4 | Delete an empty question bank |  |  |  |  |
|  | FR6a | Take a quiz by listing question banks |  |  |  |  |
|  | FR6b | Take a specific quiz using question bank identifier |  |  |  |  |
|  | FR7 | End quiz and show statistics |  |  |  |  |
|  | FR8 | Display quiz questions in a random order |  |  |  |  |
|  | FR9 | Allow the student to navigate between questions in a quiz |  |  |  |  |
|  | FR10 | Display single choice questions in an appropriate format and allow student to choose an answer |  |  |  |  |
|  | FR10 | Display fill-the-blanks questions in an appropriate format and allow the student to input missing words or phrases |  |  |  |  |
|  | NFR2 | Display the menu and allow the user to select an option though a text-based interface |  |  |  |  |
|  |  | Validate the question bank identifier inputted by the user |  |  |  |  |
|  |  | Save an empty question bank to the database |  |  |  |  |
|  |  | Get a module’s question banks from the database |  |  |  |  |
|  |  | Add a question bank to a module |  |  |  |  |
|  |  | Remove a question bank from a module |  |  |  |  |
|  |  | Write a new question to the database |  |  |  |  |
|  |  | Convert object into a String |  |  |  |  |
|  |  | Set the value of the correct answer index |  |  |  |  |
|  |  | Set question type |  |  |  |  |
|  |  | Quit the application |  |  |  |  |

## Screenshots

Test 1

A black background with white text

Description automatically generated

Test 2

A screenshot of a computer program

Description automatically generated

Test 3

A computer screen with white text

Description automatically generated

Test 4

Test 5

Test 6

Test 7

Test 8

Test 9

Test 10

Test 11

Test 12

Test 13

Test 14

A computer screen shot of a black screen

Description automatically generated

Test 15

Test 16

Test 17

Test 18

Test 19

Test 20

Test 21

Test 22

Test 23

Test 24

Test 25

Test 26

Test 27

A black background with white text

Description automatically generated

A computer screen shot of a black background

Description automatically generated

# Evaluation

## Approach to the problem

After creating use case and class diagrams, I first implemented the Main class by creating the main menu functionality, originally inside the runApp() method. After noticing how large the main class became, the menu functionality was moved into its own class to aid readability and maintance of the code.

I then moved to the Module class and began implementing the addQuestionBank() method as this seemed the next logical step in functionality. Initially this focused on adding a bank and storing the relationship. File reading and writing could not be implemented at this stage as it relied on the questions being written in the file.

After this, I implemented QuestionBank functionality. I started by writing addQuestion(), but it could not be fully completed at this stage due to the absence of appropriate methods for the different question types. I then started considering how best to implement the different questions, and reconsidering the inherited attributes and methods from the Question class. I created some getters and setters, which were later removed as they were not needed. I made further changes to the attributes, eventually deciding on those in the final implementation. Once this was decided, I worked on methods to add questions of the different question types, and a way for the program to determine the question type. Once the program was able to get user input and instantiate a question object, I created the file writing and reading methods for questions, and then question banks in the Module class, and validation on the question bank identifiers. After initial implementation, many small changes were made, including creating an enumeration for question types, only deleting the question data from a file and leaving the empty question bank, and lots of bug fixing.

Once I was happy that the program could add, delete and show questions and question banks correctly, I moved on to implementing the quiz feature. Initially, I implemented it in one large method, but decided to split it up into smaller ones to reduce code duplication and improve readability. Additional constructors were added to SingleAnswer and FillBlanks classes. I first allowed the program to output a question and allow an answer and navigation to be inputted. Once working, answer checking and quiz statistics were implemented, and finally both required ways to start a quiz.

## Creativity and innovation

## Difficulties

At the start of the project, I found it difficult to find the best way to implement the different question types. At the time, I did not know about abstract classes, but once I learnt about them, it made implementation easier. I also found it difficult to decide on the attributes of the question classes, but the final implementation makes it easier to read and write from the file as the same attributes are used by the different question types.

Deleting a question from the file was also difficult to work out as due to duplication of question bank identifiers in the file, program could not just search for this and delete it as it could delete the wrong question. Instead, the line number had to be found, by showing all questions in the file, and getting the user to input the question number before the deleting method is run.

## Further improvements

All functional and non-functional requirements were met but some improvements to the code could be made.

The structure of the questions stored in the database could be improved. The final implementation results in duplication of the question bank identifiers. In the future, all questions could be stored in the same line or underneath the same question bank identifier.

The program uses lots of linear searches to output and delete questions and question banks, which work well on small number of questions, but would become slow with lots of questions. I would implement a binary search and sort the file to improve efficiency.

## Mark

As I have successfully implemented all requirements and produced detailed documentation about the project, I believe I would get a mark around 75%.