SESSION: 2019/20 DIET 1

Module Title: Programming 2

Module Code: M2I324178-19-A

Level: 2

Module Leader: Martin L Gallacher

Hand-out time (GMT):

13:00 on Tuesday 29-10--2019 – GMT

Hand-in time (GMT). Hand in before:

17:00 on Tuesday 26-11--2019 – GMT

Contents

Programming 2 Coursework Specification (2019-2020)

The scenario

A local college is considering the construction of a quiz application to support student learning.

As part of a feasibility study, a Java application is required to collate potential questions and persist them to file storage. Initially, two type of questions are proposed: a simple text response; and, a simple numeric response; future expansion will add multiple choice, YES/NO questions etc.

All question types have the following common attributes:

- *Id* (Integer)
- *Question Text* (String)
- *Topic Area* (String)

Simple Text Questions additionally have a String *Response* attribute; while **Simple Numeric Questions**, alternatively, have an Integer *Response* attribute.

Question data is stored in a single file; either delimited text format or object file format.

Model Class Diagram

See Appendix 1

Requirements

A starter project is supplied for a console menu-driven application. The starter project contains a controller class with stubs for methods implementing the required functionality listed below:

1. Load question data from a specified file into a question collection implemented using a repository and list in *id* order
2. Add a new question (**Simple Text Response** or **Simple Numeric Response** as required) to the question collection
3. Modify the *Question Text* for a specified question in the question collection
4. List, for a specified *topic area*, question data in *question text* order
5. Persist the collection to a specified file

The starter project contains starter model, repository and controller classes which require to be completed. A sample delimited text file is provided in the project top-level directory; named <*questions.txt*>. Note… in order to test your code properly, you will need to work with a file that is larger, i.e. with more data so you are advised to populate the file further with your own data (remember it must remain consistent with the format of your code).

Architecture

The architectural pattern in the labs is to be used with classes as detailed in Appendix 2.

Development Approach

Drawing on the techniques you have learned in the labs, you will follow an incremental development approach, with the following 5 increments:

An incremental approach should be used with the following increments:

1. **Load questions from a file and list in Id order**
   This fulfils Requirement 1. To complete this increment you should:

a. Complete the **Question**, **SimpleTextQuestion** and **SimpleNumericQuestion** classes with required fields and methods as depicted in the class diagram (see Appendix 1).
b. Decide on which type of collection you want the repository to define and make the necessary adjustments to the **Repository** class and **RepositoryInterface**.
c. Partially complete the **QuestionController** class by completing the constructor which creates a new **Repository** object from a file if specified by the user.
d. Partially complete the **QuestionController** class to implement the *listQuestionDataInIdOrder()* method.
e. Create a **DAOImpl** class to realize the **DAOInterface** using either text file or object file format.
f. Modify the **Repository** class to use the **DAOImpl** class to load questions from a specified file.

Document the results of your testing using screen snippets copied into a Word document. Take a copy of the **QuestionController** class and name it as *QuestionController_Increment1.java*.

2. Add new question

This fulfils Requirement 2. To complete this increment, you should:

a. Fully implement the *addQuestion()* method in the **QuestionController** class; this should ask the user for required details, create a new **SimpleTextQuestion** or **SimpleNumericQuestion** object and add it to the **Repository** object.

Document the results of your testing using screen snippets copied into a Word document. Take a copy of the **QuestionController** class and name it as *QuestionController_Increment2.java*.

3. Modify the Question Text of a specified question

This fulfils Requirement 3. To complete this increment you should:

a. Complete the *modifyQuestion()* method in the **QuestionController** class; use a user supplied *id* value to retrieve the **Question** to update and then modify the *Question Text* with data supplied by the user.

Document the results of your testing using screen snippets copied into a Word document. Take a copy of the **QuestionController** class and name it as *QuestionController_Increment3.java*.

4. List, for a specified topic area, question data in order of question text

This fulfils Requirement 4. To complete this increment you should:

a. Complete the *listTopicQuestionDataInTextOrder()* method in the **QuestionController** class using a user supplied topic area value.
b. Ensure the **Question** class defines a method to compare questions on the *QuestionText* attribute.

Document the results of your testing using screen snippets copied into a Word document. Take a copy of the **QuestionController** class and name it as *QuestionController_Increment4.java*.

5. Persist question collection to a file

This fulfils Requirement 5. To complete this increment you should:

a. Implement the *store()* method of the **DAOImpl** class if you have not already done so.
b. Implement any **necessary** *toString(DELIMITER)* methods.

Narrative: students should write a narrative summary (in a word document) of the (attempted) implementation of each of the 5 increments. The narrative should explain the methods and objects used, and provide screen snippets of console output – you can use this section to explain any unresolved problems and how you have attempted to fix them.

Submission: requirements and procedure

Please do the following:

1. Create a folder named as follows: <studentname_studentsurname>
2. Copy your NetBeans project folder in to the <studentname_studentsurname> folder
3. Copy your narrative document in to the <studentname_studentsurname> folder. Please ensure that the cover page is copied as the first page of this document.
4. Compress <studentname_studentsurname> folder to < studentname_studentsurname.zip> and upload the zip file to GCU Learn (a GCU Learn 'announcement' will be made as to where you should upload this to).

Marking Scheme

Requirement	Marks
A) Narrative document	5
B) Question class	6
C) SimpleTextQuestion class	3
D) SimpleNumericQuestion class	3
E) Repository class	3
F) DAOImpl class	5
G) Increment1 QuestionController class	7
H) Increment2 QuestionController class	4
I) Increment3 QuestionController class	5
J) Increment4 QuestionController class	5
K) Increment5 QuestionController class	4
Total	50

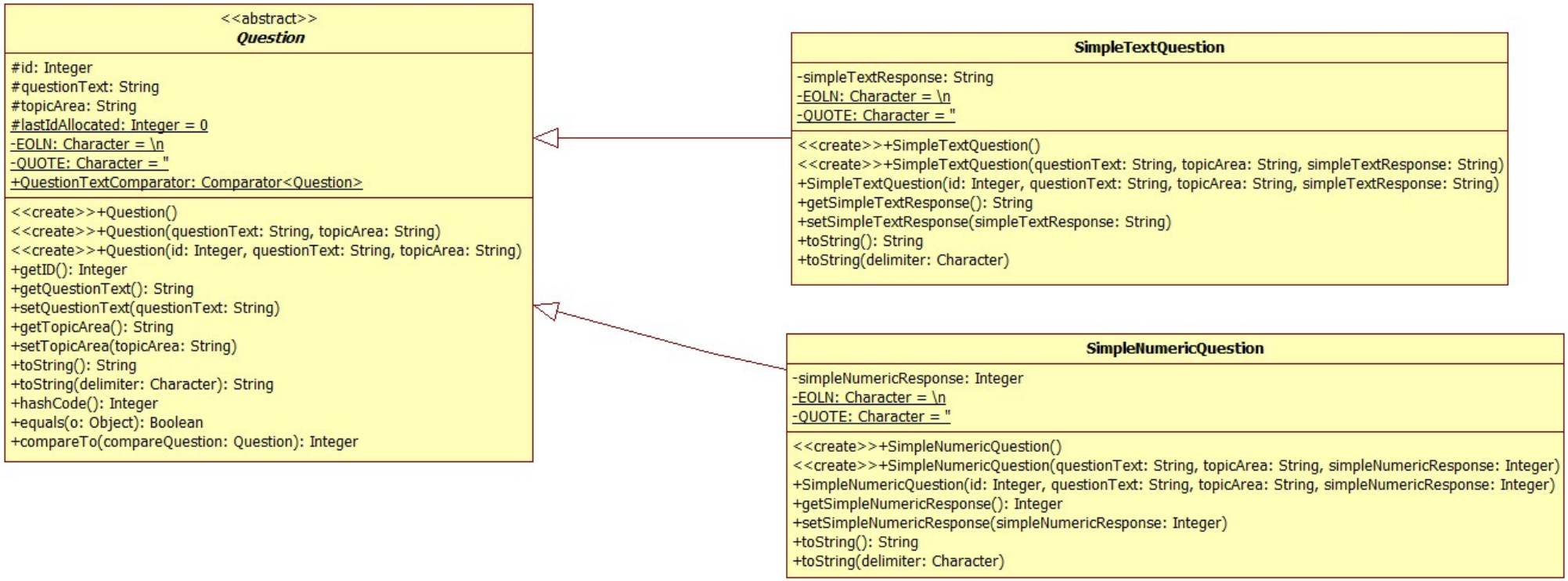
Test Data

```
1,"Simple Text Question 1","Topic Area 1","STQ","Simple Text Question 1 Response"
2,"Simple Numeric Question 1","Topic Area 2","SNQ",5
3,"Simple Text Question 2","Topic Area 1","STQ","Simple Text Question 2 Response"
4,"Simple Numeric Question 2","Topic Area 3","SNQ",25
```

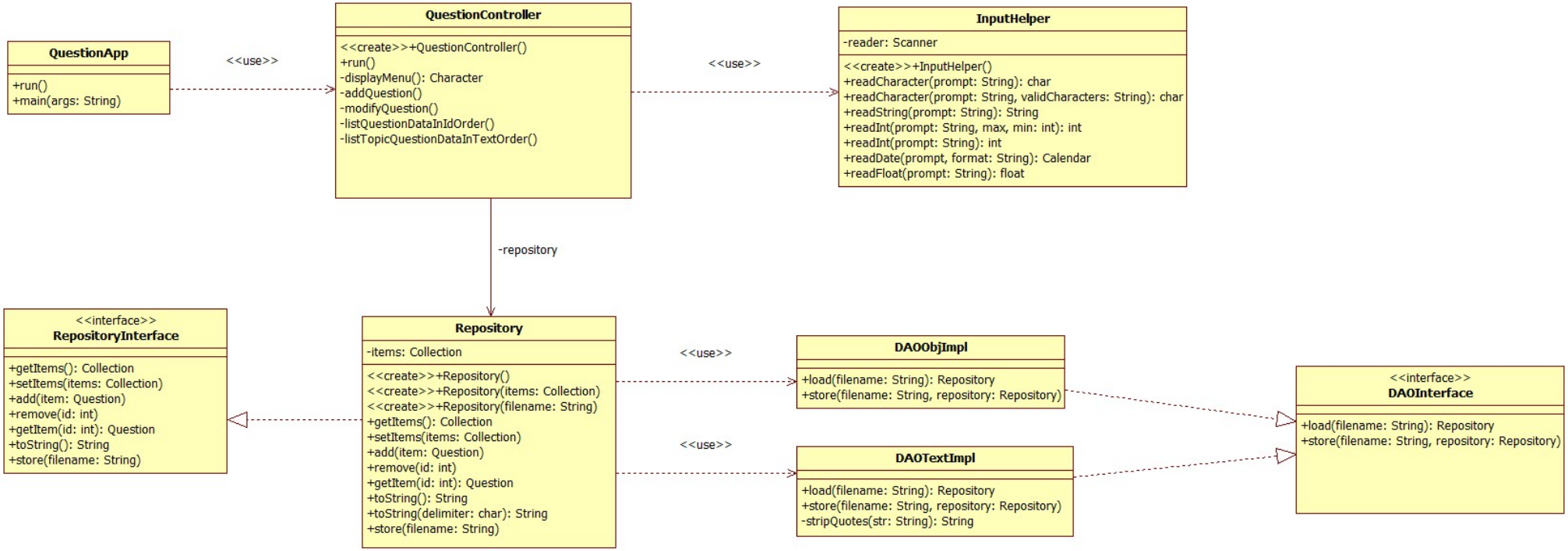
Starter Project Test Run

```
run:
Question App
============


Question Id Order
=================

A. Add Question     B. Modify Question  C. List Topic Question Data In Text Order     Q. Quit
Enter choice:
A
Add Question
============
Question Id Order
=================

A. Add Question     B. Modify Question  C. List Topic Question Data In Text Order     Q. Quit
Enter choice:
B
Modify Question
===============
Question Id Order
=================

A. Add Question     B. Modify Question  C. List Topic Question Data In Text Order     Q. Quit
Enter choice:
C
Question Text Order
===================
Question Id Order
=================

A. Add Question     B. Modify Question  C. List Topic Question Data In Text Order     Q. Quit
Enter choice:
Q
Thank you for using Question App. Good bye.

BUILD SUCCESSFUL (total time: 12 seconds)
```

Appendix 1


<<abstract>>
Question
#id: Integer
#questionText: String
#topicArea: String
#lastIdAllocated: Integer = 0
-EOLN: Character = \n
-QUOTE: Character = "
+QuestionTextComparator: Comparator<Question>
<<create>>+Question()
<<create>>+Question(questionText: String, topicArea: String)
<<create>>+Question(id: Integer, questionText: String, topicArea: String)
+getID(): Integer
+getQuestionText(): String
+setQuestionText(questionText: String)
+getTopicArea(): String
+setTopicArea(topicArea: String)
+toString(): String
+toString(delimiter: Character): String
+hashCode(): Integer
+equals(o: Object): Boolean
+compareTo(compareQuestion: Question): Integer
SimpleTextQuestion
-simpleTextResponse: String
-EOLN: Character = \n
-QUOTE: Character = "
<<create>>+SimpleTextQuestion()
<<create>>+SimpleTextQuestion(questionText: String, topicArea: String, simpleTextResponse: String)
+SimpleTextQuestion(id: Integer, questionText: String, topicArea: String, simpleTextResponse: String)
+getSimpleTextResponse(): String
+setSimpleTextResponse(simpleTextResponse: String)
+toString(): String
+toString(delimiter: Character)
SimpleNumericQuestion
-simpleNumericResponse: Integer
-EOLN: Character = \n
-QUOTE: Character = "
<<create>>+SimpleNumericQuestion()
<<create>>+SimpleNumericQuestion(questionText: String, topicArea: String, simpleNumericResponse: Integer)
+SimpleNumericQuestion(id: Integer, questionText: String, topicArea: String, simpleNumericResponse: Integer)
+getSimpleNumericResponse(): Integer
+setSimpleNumericResponse(simpleNumericResponse: Integer)
+toString(): String
+toString(delimiter: Character)

Appendix 2


QuestionApp
+run()
+main(args: String)
<<use>>
QuestionController
<<create>>+QuestionController()
+run()
-displayMenu(): Character
-addQuestion()
-modifyQuestion()
-listQuestionDataInIdOrder()
-listTopicQuestionDataInTextOrder()
<<use>>
InputHelper
-reader: Scanner
<<create>>+InputHelper()
+readCharacter(prompt: String): char
+readCharacter(prompt: String, validCharacters: String): char
+readString(prompt: String): String
+readInt(prompt: String, max, min: int): int
+readInt(prompt: String): int
+readDate(prompt, format: String): Calendar
+readFloat(prompt: String): float
-repository
<<interface>>
RepositoryInterface
+getItems(): Collection
+setItems(items: Collection)
+add(item: Question)
+remove(id: int)
+getItem(id: int): Question
+toString(): String
+store(filename: String)
Repository
-items: Collection
<<create>>+Repository()
<<create>>+Repository(items: Collection)
<<create>>+Repository(filename: String)
+getItems(): Collection
+setItems(items: Collection)
+add(item: Question)
+remove(id: int)
+getItem(id: int): Question
+toString(): String
+toString(delimiter: char): String
+store(filename: String)
<<use>>
DAOObjImpl
+load(filename: String): Repository
+store(filename: String, repository: Repository)
<<use>>
DAOTextImpl
+load(filename: String): Repository
+store(filename: String, repository: Repository)
-stripQuotes(str: String): String
<<interface>>
DAOInterface
+load(filename: String): Repository
+store(filename: String, repository: Repository)