SCIT

School of Computing and Information Technology

Spring 2015

CSCI213/MCS9213/CSCI813 — Java Programming and Applications

Assignment 2 (10 marks)

Due Time and Date:

23:59:59 on Sunday, 13 September

Objectives

This assignment requires you to design and implement Java classes to support a Java quiz system using Java JDBC technology. You will apply your knowledge and skills in Java JDBC design and implementation to develop a system that stores Java quiz questions in an embedded Java DB database.

Upon completion of this assignment, you should be able to create Java programs to connect to a database, execute SQL statements and process result sets using JDBC. In addition, you should be familiarised with use of the Java DB, in particular the embedded Derby to support your Java applications.

Tasks and Requirements

In this assignment, you will design and implement Java classes that support a Java quiz system using Java DB. The application program, called `JavaQuizDB.java`, is provided in the Appendix A. You should not modify the application program. Instead, you should design and implement your classes to support it. The application has a text-based user interface that allows a user to register his/her name and take the Java quiz. The user interface class, called `UserInterface.java` and the `Student` class are provided in the `UserInterface` package with documentation. The classes that represent multiple choice questions and true-and-false questions are also provided in the `QuestionLibrary` package with documentation. Your major work is to create a class, called `MyQuestionDB.java` to implement the `QuestionDB` interface to interact with the Java DB. The `QuestionDB` interface is provided in the Appendix B. All the interaction of your code to the database should be realised using the methods specified by the `QuestionDB` interface.

General Requirements:

- You should observe the common principles of OO programming when you design your Java classes, which include abstraction, encapsulation, inheritance and polymorphism.
- You should create your classes by making use of features of the Java classes
- You should make proper documentation and implementation comments in your codes where they are necessary.
- Logical structures and statements are properly used for specific purposes.
- Your code should comply with the major requirements of Java Code Conventions.

Structure of the Java packages:

The Java classes created in this assignment will have the following package structure. You may add more supporting classes in appropriate packages if necessary.

/WorkingDirectory
```
      JavaQuizDB.java
      derby.jar
      /au/edu/uow
            /UserInterface
                  UserInterface.class
                  Student.class
            /QuestionLibrary
                  Question.class
                  MultipleChoiceQuestion.class
                  TrueAndFalseQuestion.class
            /QuestionDB
                  QuestionDB.java
                  MyQuestionDB.java
```

The `UserInterface` and `QuestionLibrary` packages with documentation are provided.

Task 1: Database management

Requirements:

You program will be responsible to setup the database, create and populate the table, and remove the table when it is no longer needed. The Derby RDMS will be used in embedded mode. The release of the Derby database engine and embedded driver library, `derby.jar`, to be used in this assignment is provided. This library should be located in your working directory.

You should use a properties file, called `database.properties`, located in your working directory to manage the database. The file contains the following properties:

```
jdbc.drivers
jdbc.url
derby.system.home
```

The Derby system home directory should be called `MyQuestionDB` located in your working directory, which is set using `System.setProperty` method before the JDBC driver is loaded. The database to store all question should be called `JavaQuestionDB` in the Derby system home directory.

Task 2: Database creation and population

Requirements:

All questions are stored in a table called Questions in the database. The table is created using the following SQL statement:

```
CREATE TABLE Questions (Q_ID INT NOT NULL GENERATED ALWAYS AS
                        IDENTITY (START WITH 1, INCREMENT BY 1),
                        question CLOB,
                        choices CLOB,
                        answer INT,
                        CONSTRAINT primary_key PRIMARY KEY (Q_ID))
```

You should populate the table using the `addQuestion` method specified in the `QuestionDB` interface with questions from the question file with a format as specified in the Appendix C. This table will store both multiple choice questions and true-and-false questions.

Before the program exits, the table should be removed. Each execution of the program will start with a new table.

All above functions should be implemented programmatically in your program.

Task 3: Design and Implementation of the `MyQuestionDB` Class

Requirements:

You should design and implement a class called `MyQuestionDB` to implement the `QuestionDB` interface to interact with the database. This class should be created in the `au.edu.uow.QuestionDB` package. Read the `QuesitonDB` interface to understand the requirements for the implementation.

This class should use JDBC API to interact with a Derby database in the embedded mode.

The `buildQuestionDB` method creates and populates the database table with questions from the question file using the `addQuestion` method specified in the `QuestionDB` interface. The `makeQuiz` method should return a list of randomly selected questions from the database using the `getQuestion` method specified in the `QuestionDB` interface. You should use the classes provided in the `QuestionLibrary` package to represent and manage both types of the questions. Read the accompanying documentation of the package to understand the usage of the package.

The JDBC driver and database URL should be loaded into `Properties` object from the `database.properties` file.

Each method specified in the `QuestionDB` interface must be implemented by interacting directly with the database. The program should not buffer a collection of question objects.

Submission

- The **`@author`** line in all your source code files should include **your Subject Code**, **your name**, **student ID**, and **Unix login name**.
- IMPORTANT: Any submission that cannot be decompressed may receive zero mark. Any submission that resulted in compiling errors may have 50% marks deducted.
- You should first zip your files including subdirectory paths into a file called called **`a2.zip`**. For this assignment you submit all your programs including the Java quiz application, Question interface and the generated API documentation, basically all files in the working

directory and subdirectories **excluding** files that may be generated by the IDE you might have used. Failing to do so may result in deduction of marks.

- Submit the file from your University Unix account on Banshee (`banshee.uow.edu.au`) as follows:

  ```
  turnin -c csci213 -a a2 a2.zip
  ```

  or, if you submit it after the deadline:

  ```
  turnin -c csci213 -a a2-late a2.zip
  ```

- Use the following command to check the submitted file:

  ```
  turnout -c csci213 -a a2
  ```

  or, if you submit it after the deadline:

  ```
  turnout -c csci213 -a a2-late
  ```

Marking Scheme

Mark to 0.5 divisions.

For CSCI213/MCS9213/CSCI813 students.

General requirements	-0.5 ~ -1.0 mark for each error
Task 1	2
Task 2	3
Task 3	5

Appendix A: Java Quiz Application Program

(This program is available for downloading)

You should not modify this program. Instead, you should create your classes to support the program.

```
import java.util.List;
import au.edu.uow.QuestionDB.*;
import au.edu.uow.QuestionLibrary.*;
import au.edu.uow.UserInterface.*;

/**
 * This is the application that tests students with Java quiz.<br>
 * This program is provided for the CSCI213 Assignment. <br>
 * Note: You should not modify this program. This program will use
 *       classes you created as instructed in the Assignment paper.
 *
 * @author Lei Ye
 *
 */
public class JavaQuizDB {

	/**
	 * This constant specifies the number of questions in each quiz.
	 */
	final private static int NoOfQuestions = 5;

	/**
	 * This is the entry point of the application
	 * @param args Command line options.
	 */
	public static void main(String[] args){

		if(args.length == 1){
			/* You should create the MyQuestionDB class */
			MyQuestionDB myQuestionDB = new MyQuestionDB();
			boolean isQuestionLibraryReady = myQuestionDB.buildQuestionDB(args[0]);

			if (isQuestionLibraryReady == true) {
				System.out.println("===== Java Quiz, v1.0 =====\n");
				/* The UserInterface class is provided */
				UserInterface ui = new UserInterface();

				System.out.println("-- Student login --");
				/* The Student class is provided*/
				Student student = ui.getStudent();

				System.out.println("\n-- Quiz begins --");
				List<Question> quiz = myQuestionDB.makeQuiz(NoOfQuestions);
				ui.startQuiz(quiz, student);
				myQuestionDB.cleanUpDB();

				System.out.println("\n-- Student marks  --");
				ui.showStudentMarks(student);
			}else{
				System.err.println("Error: failed to build a question database.
				                    Exiting ...");
				System.exit(0);
			}
		}else{
			System.err.println("Usage: java JavaQuizDB questionFileName");
		}
	}
}
```

Appendix B: QuestionDB Interface

(This program is available for downloading)

You should not modify this interface declaration. Instead, you should create your classes to implement it.

```
package au.edu.uow.QuestionDB;

import java.util.List;
import au.edu.uow.QuestionLibrary.Question;

/**
 * This interface specifies the requirements of implementation classes
 *
 * This program is provided for the CSCI213 Assignment. <br>
 * Note: You should not modify this program but create a suitable
 *       class to implement this interface.
 *
 * @author Lei Ye
 * @version v.1.0
 *
 */
public interface QuestionDB {

	/**
	 * This method creates and populates a table in the database
	 *   storing the questions from the question file. This method
	 *   should use the addQuestion method to insert question content
	 *   into the database
	 * @param questionFile The file name of the question file
	 * @return True if the database is successfully populated
	 * @see #addQuestion(Question)
	 */
	boolean buildQuestionDB(String questionFile);

	/**
	 * This method returns the total number of questions in the database
	 * @return The total number of questions in the question database
	 * @see #buildQuestionDB
	 */
	int getTotalNumberOfQuestions();

	/**
	 * This method returns the question from the database at the given position
	 * @param questionIndex The index of the question in the database
	 * @return The question object
	 */
	Question getQuestion(int questionIndex);

	/**
	 * This method adds a question to the database
	 * @param question The question object to be added to the database
	 * @return True if the operation is successful
	 */
	boolean addQuestion(Question question);

	/**
	 * This method removes the created table from the database
	 * @return True if the operation is successful
```

```
         * @see #buildQuestionDB(String)
         */
        boolean cleanUpDB();

        /**
         * This method makes a quiz from the question database
         *   with the number of questions as specified. This method
         *   should use the getQuestion method to retrieval question
         *   content from the database
         * @param noOfQuestions - the number of questions in a quiz
         * @return Quiz questions in a list
         * @see #getQuestion(int)
         */
        List<Question> makeQuiz(int noOfQuestions);
}
```

Appendix C: Question Library File Format

All questions and their sections in the question library document are marked with start-tags and matching end-tags, a format conforming to XML 1.0 Specification.

A XML document is essentially a marked up string of characters, which can be read as a text file.

The root tag for the document is `JavaQuestions`. The multiple choice questions, tagged as `MQuestion`, have three sections that are `question`, `choices` and `answer` while the true and false questions, tagged as `TFQuestion`, have two sections that are `question` and `answer`. Sections may appear in any order and each section may have different numbers of lines or choices.

You should not change the format. A sample question file, called `questions.xml`, is provided for your conformance test.

A segment of an example question file is shown below.

```
<?xml version="1.0"?>
<JavaQuestions>
<MQuestion>
<question>
Which of the following statements is correct?
</question>
<choices>
A class in Java can extend one or more classes
A class in Java can extend one class and implement one or more interfaces
A class in Java can extend one interface and implement one or two classes
A class in Java can extend one class and implement one interface
</choices>
<answer>
2
</answer>
</MQuestion>
<TFQuestion>
<question>
Superclasses can contain abstract methods.
</question>
<answer>
true
</answer>
</TFQuestion>

. . .

</JavaQuestions>
```