Project 10: Quiz Creator II

Assigned: November 9, 2010
Due: Sunday, November 13, 2010 by 11:59 PM

Deliverables:

The following project files must be uploaded to Web-CAT by the due date and time specified above (see the Lab Guidelines for information on submitting project files). **If you are not able to submit your project via Web-CAT, then you must e-mail your project to gofflau@auburn.edu before the deadline. Therefore, it is important to give yourself time to submit via e-mail if you have trouble submitting to Web-CAT.**

- QuizQuestion.java
- MultipleChoice.java
- ShortAnswer.java
- Quiz.java
- IndexCompare.java
- CategoryIndexCompare.java

Specifications:

Requirements: The Quiz class represents a quiz with a name and a set of multiple choice and short answer questions. All questions that are added to the quiz must have an index. Questions with the same same index or question text can be added to the quiz, but cannot have the same index AND question text.

Questions that are unsorted are ordered based the order in which they were added to the quiz. By default questions are sorted based on the length of their question text, but there should also be an option to sort questions based on their index and an option to sort questions based on their category (multiple choice first) and then their index.

QuizQuestion.java

- **Design**: Modify your QuizQuestion class so that the **natural ordering** of all QuizQuestion objects is based on the number of characters in the question text (i.e. the number of characters in the getQuestionText return). In other words, the return value of quizObj1.compareTo(quizObj2) is negative if the number of characters in the question text of quizObj1 is less than that of quizObj2.

Quiz.java

- **Design**: The Quiz class should have two constructors: one takes no parameters and one takes a String representing the name of the Quiz.

  The following public constants should also be included as type int:
  - `MULTIPLE_CHOICE`, `SHORT_ANSWER`, and `ALL_QUESTIONS`,
  - `QUESTION_LENGTH`, `INDEX` and `CATEGORY`

  The Quiz class should also contain the following methods:
  - `getName` and `setName`: getter and setter methods for the name of the Quiz (a String). The name of the quiz should be "Today's Quiz" (case sensitive) unless the

name is set via the constructor or setName. The setName method should return true and set the name only if it is invoked with an actual parameter that is **not** an empty string, a String that is whitespace only, or a null value. For example, the following method invocations should return false if quizObj is an object of type Quiz:

```
quizObj.setName("");
quizObj.setName("\t");
quizObj.setName(null);
```

- `addQuestion`: Takes any object that is a subclass of QuizQuestion as a parameter. Returns true and adds the question if and only if the question is not already present in the Quiz based on the equals method defined in QuizQuestion or if the index of the question has not been set.

- `removeQuestion`: Takes a question index (a String) as a parameter and removes all questions in the quiz with the specified index. Returns the number of questions that were removed as an int value.

- `questionList`: The questionList method takes no parameters and returns an **unsorted** ArrayList of QuizQuestion objects (generic type QuizQuestion). The questionList method must be overloaded as follows:
  - `questionList(int questionType)`
    If QuizQuestion.MULTIPLE_CHOICE is sent as a parameter, then the method returns an **unsorted** list of all multiple choice questions. If QuizQuestion.SHORT_ANSWER is sent as a parameter, then the method returns an **unsorted** list of all short answer questions. If QuizQuestion.ALL_QUESTIONS is sent as a parameter, then the method should return an **unsorted** list of all questions.
  - `questionList(int questionType, int sortType)`
    The return is filtered by type based on the first parameter as specified in the previous method description. Additionally the return should be sorted based on the following values of sortType:
    - If sortType is equal to QuizQuestion.QUESTION_LENGTH, then the return should be sorted based on the natural ordering of QuizQuestion objects.
    - If sortType is equal to QuizQuestion.INDEX, then the return should be sorted based on the questions' indices, which are strings.
    - If sortType is equal to QuizQuestion.CATEGORY, then the return should be sorted based on their category and then their index (use the CategoryIndexCompare class).

IndexCompare.java

- **Design**: The IndexCompare class implements the Comparator interface (with generic type QuizQuestion) and defines the compare method in such a way that QuizQuestion objects are ordered based on the alphabetic ordering of their indices (case insensitive). For example, a

question with index "4A" is before a question with index "4b" and a question with index "4a" precedes a question with index "4B".

CategoryIndexCompare.java

- **Design**: The CategoryIndexCompare class implements the Comparator interface (with generic type QuizQuestion) and defines the compare method in such a way that QuizQuestion objects are ordered as follows:
  - multiple choice always precede short answer questions.
  - if the questions have the same category, then they are ordered based on index.

  In other words, multiple choice questions always come before short answer questions on the quiz. If two questions are both of the same type, then they are sorted based on their index.

Code / Test: Because you will need to add an unknown number of questions, it is recommended that you use **one ArrayList object** to store all question objects. Use the *instanceof* reserved word to determine an object's type (see the project description). As always, try to keep variables local when appropriate (you only need two instance variables and constants in the Quiz class). You will lose points if you get any errors while compiling, so be sure to always use generic types properly. Example:

```
Note: _________.java uses unchecked or unsafe operations.
```

You will need to preserve the natural ordering of questions even when you return a sorted ArrayList in a method. In order to do so you will have to create a new ArrayList when sorting questions (a local variable). You can either create a new ArrayList and add all of the QuizQuestion objects or use the clone method in Object.

JUnit tests will not be required for this project, but it is highly recommended that you perform unit testing via the interactions pane.