Module 4 Lab -- Questionnaires

Due Saturday by 11:59pm **Points** 100

> The goal of this lab is to design and implement a union data type and to practice the application of double dispatch.

Questionnaires

Many online questionnaire tools like SurveyMonkey, Doodle Poll, etc. (even Canvas and Handins) allow creating a questionnaire made of several questions. Questions are of different types from those with yes/no answers to larger free form-text type questions.

Each question, irrespective of type, has the following common aspects:

- It has the text of the question itself.
- It allows a way to enter an answer as a `String` and get the evaluation result of the answer. The string it returns is either "Correct" or "Incorrect".

The types of questions for our questionnaire are:

1. **Likert:** can be answered on a fixed, 5-point Likert scale (Strongly Agree, Agree, Neither Agree nor Disagree, Disagree, Strongly Disagree). This type of question can be created by passing the text of the question. Since this question asks an opinion, there is no "correct" answer. An answer can be entered as one of the option numbers, numbered from 1 in the above order. Any valid option number is a "correct" answer.
2. **Multiple choice:** offers several options, only one of which is correct. This type of question can be created by passing the text of the question, the correct answer and the options. A question may have at least 3 options, but no more than 8. An answer can be entered as one of the option numbers in a string. For example, "1", "3", etc. Option numbers start at 1.
3. **Multiple select:** offers several options, but there are multiple correct answers. This type of question can be created by passing the text of the question, the correct answer and the options. A question may have at least 3 options, but no more than 8. Both the correct answer and the user's answer are entered as the option numbers in a string. For example, "1", "1 3", "4 1", etc. Option numbers start at 1. An answer is correct if and only if it contains all the correct options and none of the incorrect ones.
4. **True/False:** can be answered in one of two ways: true or false. This type of question can be created by passing the text of the question and the correct answer. The only valid answer to this type of question is a "True" or "False".

For all the question types, an invalid answer is deemed as incorrect.

In addition to all of them, we also wish to create a question bank that stores questions and orders them so that they can be picked quickly to assemble a questionnaire. To do this, we wish to maintain questions in a certain order:

- All true/false questions should be before any multiple-choice questions.
- All multiple-choice questions should be before any multiple-select questions.
- All multiple-select questions should be before any Likert questions.
- Within a question type, they should be ordered in the lexicographical (dictionary) order of their question text.

For example, one should be able to put several of these objects into an array, and sort it using the `Arrays.sort` method (a similar method exists in the `Collections` class if you already have an `List`). For example:

```
// Objects a-h are created before the lines below
Question[] questionnaire = {a, b, c, d, e, f, g, h};
Arrays.sort(questionnaire);
```

You may read more about `Arrays.sort` **here (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html)** and `Collections.sort` **here (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Collections.html)**.

What to do

Package: questions

Design and implement the data for the above in a way that captures their similarities and accurately represents the relevant data. To do this, you should implement a `Question` interface that (at the minimum) provides the ability to `getText` for a question as well as as an `answer` method that takes the provided answer and determines whether it is correct or not. Next implement each type of question: `Likert`, `MultipleChoice`, `MultipleSelect`, and `TrueFalse`.

Testing

Whenever you write a class, you should also write tests for that class that proves not only that your code CAN work but that it WILL ALWAYS work. Your goal here is to write tests to achieve as close to 100% coverage as possible. But even more importantly, your tests should be sufficient to ***convince someone else that your code works correctly***.

What to Submit

1. Create a zip file that directly contains only your src/ and test/ folders. When you unzip the file, you must see only these two folders.
2. Log on to the **Handins server (https://handins.ccs.neu.edu/)**
3. Navigate to this lab and submit the zip file.

4. Wait for a few minutes for the grader's feedback to appear, and take action if needed.

Grading Criteria

This assignment will be assessed via the automatic testing available on the Handins server. You should ensure that your code does not have any code style violations and that it passes all of the test cases. This is worth 1% of your final grade.

Additionally, your code will be assess via peer evaluation. Each student will be required to review one other student's code for an additional 0.5% of their final grade. During the peer review your code will be evaluated based on three criteria:

1. Design and organization of classes, interfaces, and methods including
    - How appropriately you captured the data and operations in the implementation
    - Whether code looks well-structured and clean (not unnecessarily complicating things or using unwieldy logic)
    - How well your design and implementation embraces various design principles that you have learned so far
2. Understandability and quality of documentation including
    - Whether the documentation enhances the understandability of the code rather than just repeating how something is implemented
    - Whether it is stand alone, meaning whether the documentation is sufficient for use without reading the implementation
3. Quality and coverage of tests including
    - Whether tests are modular -- do you write one big, monolithic test method or does your test class have separate methods for each test
    - Whether tests are written in such a way so that the reader knows what is being tested
    - Whether the tests convince the reader that if all tests pass then the code it is testing works correctly