CS 342 – Software Design – Spring 2018
Term Project Part I - Revised
Development of Question, Answer, and Exam Classes

Due: **Thursday 1 February. Electronic copy due at 3:00 P.M. Optional paper copy may be handed in during class.**

Overall Assignment

The overall goal of this term project is to develop a system for the creation, management, and grading of exams containing randomly ordered questions and answers, so that each student's exam is unique while still being fair and equitable.

The goal of this first stage is to develop some simple (textual, multiple choice) Question and Answer classes, and an Exam class to contain the questions. An ExamTester class will also be needed, to drive and test the other three classes. Desired functionality of the overall system at the conclusion of this part includes the following:

1. Create an Exam object containing Question objects, each containing Answer objects:

2. Print the exam. This can be simple text printing to the screen for now.

3. Randomly reorder the questions on the exam, and randomly reorder the answers to each question. Then print the exam again.

4. "Select" an answer to some or all of the questions, as if a student had completed the exam.

5. "Grade" the exam, by totaling the value of the selected answers, and report the overall score, along with the ~~correct and selected answers~~ **value contributed** for each question.

Answer Class Details

The Answer class represents a textual multiple-choice answer, and has the following public interface:

- Answer(String) – A constructor for creating the Answer object. By default new Answers are unselected and have no value.
- print(int position) : void – Prints the answer to the screen. The input parameter indicates the position of this answer in the list of answers, 1 for the first answer, 2 for the second, and so on.
- setSelected(Boolean) : void – An answer is selected (true) when a student has selected this answer. It becomes unselected (false) if the student changes their mind or selects another answer when only one answer is allowed.
- setValue(**double, double**) : void – sets the value(s) of this Answer if it is selected or unselected respectively. The default for each is zero if unset. Set values can be positive or negative.
- getValue(void) : **double** – Get the number of points that this answer contributes to the exam score, based on whether it is selected and its value when it is selected.

Question Class Details

The Question class represents a textual multiple-choice question, and has the following public interface:

- Question(String) – A constructor for creating the Question object. New Questions do not have Answers until they are added.
- AddAnswer(Answer) – Add an Answer to a Question. By default new answers are added to the end of the existing list.
- print(int position) : void – Prints the question to the screen, along with all of its answers. The input parameter indicates the position of this question in the list of questions, 1 for the first question, 2 for the second, and so on.
- ~~getAnswer(int position) : Answer – returns (a reference to) the answer in the given position within the question.~~
- selectAnswer(int position) : void – selects the Answer in the given position. This may or may not require unselecting other previously selected Answers.
- unselectAnswer(int position) : void – unselects the Answer in the given position.
- reorderAnswers(void) : void – Randomly reorders the Answers within the Question.
- getValue(void) : **double** – Get the number of points that this question contributes to the exam score, based on which answers are currently selected.

Exam Class Details

The Exam class represents a gradable collection of Questions, and has the following public interface:

- Exam(String) – A constructor for creating the Exam object. New Exams do not have Questions until they are added. The input parameter is the title / header of the exam, to be printed before the questions. (Note that the title / header may consist of multiple lines.)
- AddQuestion(Question) – Add a Question to an Exam. By default new questions are added to the end of the existing list.
- print(void) : void – Prints the exam to the screen, along with all of its questions.
- getQuestion(int position) : Question – returns (a reference to) the question in the given position within the exam.
- reorderQuestions(void) : void – Randomly reorders the Questions within the Exam.
- getValue(void) : **double** – Get the overall value (score) of the exam. This method should work regardless of how many (if any) of the Questions have been answered.

ExamTester Class Details

The ExamTester class is the test driver for the other classes, and contains main( ). It should first print your name and netID, and then implement the 5 steps outlined on the first page of this assignment, along with suitable reporting and diagnostic messages. You may also want to create and test individual Answer and Question objects before testing the Exam class, particularly during early development.

Your ExamTester can either get input from the user or have it hard-coded. Later in the semester we will learn how to make use of data files of questions and answers.

Note that your classes will be tested both by running your ExamTester main( ) routine and by a separate test driver written by the graders, so make sure your classes work with any reasonable inputs, not just with certain special inputs.

Additonal Methods May Be Needed

In the spirit of information hiding, the internal data and workings of the class have not been specified. **All data should be kept private**, and you may need/want to add additional methods both public and private to those listed here. In particular, constructors, destructors, setters, and getters are often needed but not specified. Be careful, however, as too many setters and getters can destroy information hiding.

If you feel a need to add additional public methods, make sure to document them well in your readme file. It is probably good to discuss them with an instructor as well, in case that method will be needed by all students.

Simplifying Assumptions That May Be Relaxed Later

- All questions are multiple-choice type questions.
- Questions and answers consist of plain text only. No special characters, fonts, images, or other media required. Numbers, if present, will not be evaluated for their numerical value.
- Each question has exactly N answers, where N = 5. (Set it up as a variable, but leave it at 5 for now.)
- Only one answer may be chosen for any question. Selecting a new answer replaces the old answer. (This may be relaxed as an optional enhancement. See below.)
- The questions and answers may be rearranged freely without consequence. There are no "none of the above" or "all of the above" answers, and no answer refers to any other by position. Likewise no question is required to either precede or follow any other question. (This may also be relaxed as an optional enhancement.)
- For the most part, you can assume good input at this point, though it would be good to consider some possible bad input where it is reasonable to do so. Later in the semester we will learn about Exceptions for rigorous handling of exceptional situations.
- There is no great concern over formatting at this point, though the output should be clear and readable, and it would be good to indent questions and answers. There are no concerns now over special fonts, centering the title, preventing questions from splitting across pages, producing printable (PDF format) files, etc.

Required Output

- All programs should print your name and netID as a minimum when they first start.
- Beyond that, the system should function as described above.

Evolutionary Development

It is recommended that you first develop and test the Answer class, and then the Question class, and finally the Exam class. The ExamTester class can evolve as the other classes progress.

Other Details:

- The TA must be able to build your programs by typing "make". If that does not work using the built-in capabilities of make, then you need to submit a properly configured makefile along with your source code. As always, you are free to develop wherever you wish, but the TA will be grading the programs based on their performance on the CS department machines.

What to Hand In:

1. Your code, **including a makefile and a readme file,** should be handed in electronically using Blackboard.
2. The purpose of the readme file is to make it as easy as possible for the grader to understand your program. If the readme file is too terse, then (s)he can't understand your code; If it is overly verbose, then it is extra work to read the readme file. It is up to you to provide the most effective level of documentation.
3. The readme file must specifically provide direction on how to run the program, i.e. what command line arguments are required and whether or not input needs to be redirected. It must also specifically document any optional information or command line arguments your program takes, as well as any differences between the printed and electronic version of your program.
4. If there are problems that you know your program cannot handle, it is best to document them in the readme file, rather than have the TA wonder what is wrong with your program.
5. If you have added any optional enhancements, make sure that they are well documented in the readme file, particularly if they require running the program differently in any way.
6. A printed copy of your program, along with any supporting documents you wish to provide, (such as hand-drawn sketches or diagrams) may be handed in **to the TA** on or shortly after the date specified above. Any hard copy must match the electronic submission exactly.
7. Make sure that your **name and your ACCC account name** appear at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

It is course policy that students may go above and beyond what is called for in the base assignment if they wish. These optional enhancements will not raise any student's score above 100 for any given assignment, but they may make up for points lost due to other reasons. Note that all optional enhancements need to be clearly documented in your readme files. The following are some ideas, or you may come up with some of your own:

- Allow for the selection of multiple answers, but only for specified questions.
- Allow for certain answers that must be locked into certain positions, such as "none of the above" or "all of the above".
- Allow for questions to have more than N answers, where only the "right" answer and N – 1 "wrong" answers are displayed. Reordering the answers may change which wrong answers are displayed, but the right answer must always be included.