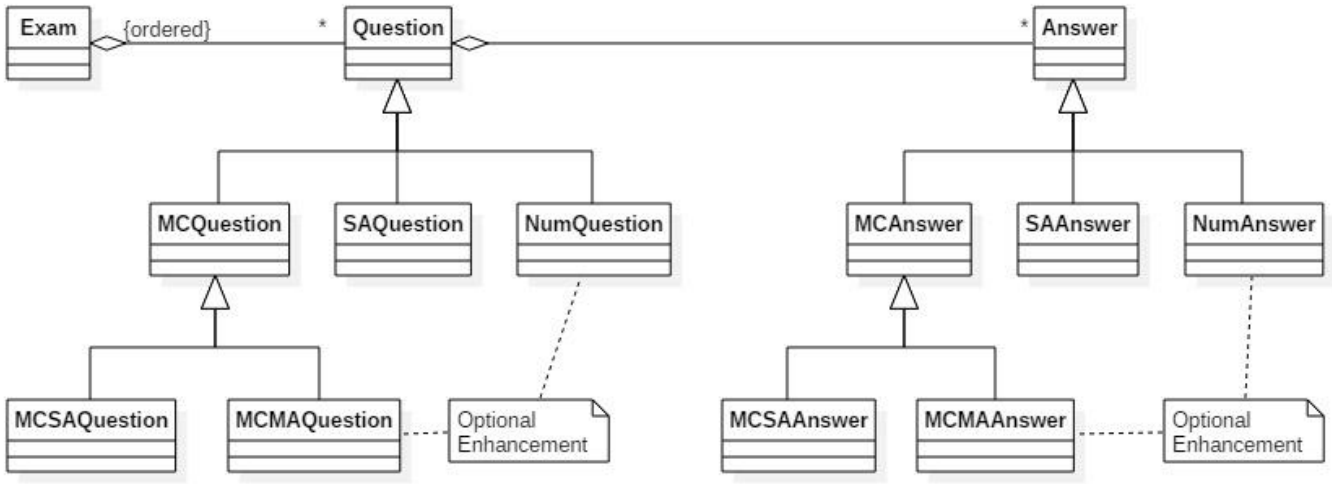
CS 342 – Software Design – Spring 2018
Term Project Part II
Development of Question, Answer, and Exam Classes

Due: **Wednesday 21 February. Electronic copy due at 3:30 P.M. Optional paper copy may be handed in during class.**

Overall Assignment

The next stage of development is to provide for different types of questions and answers on the exams, by converting the Question and Answer classes into related subtrees. In brief the task is to convert from:

to:

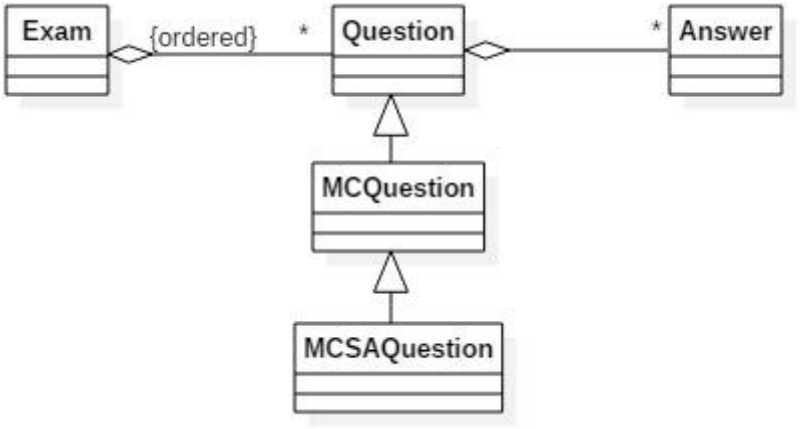


Refactoring

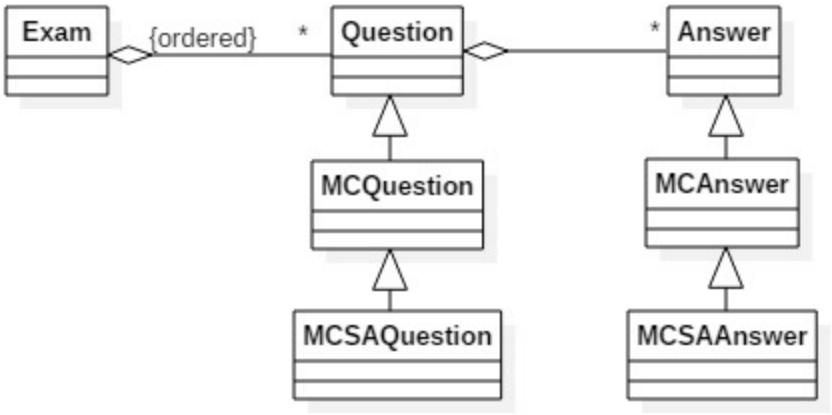
Refactoring is a process of changing the structure of a design without changing the overall functionality of the program. It is often done between development stages, either to improve the quality of the existing system or to prepare for the next development. In this case, both are called for. As a first step towards the next release, you should do the following:

1. Eliminate the int position argument from the print( ) methods in the Question and Answer classes. Now the Question class will be responsible for printing the letter labels in front of each Answer, and the Exam class will be responsible for printing the numbers in front of each Question.
2. Change the numbering system of all "position" parameters to be zero-based, instead of one-based. Note that the first question on the exam should still be printed with the number 1, and the first answer with the letter A, but internally those would both be in position zero.

3. Eliminate the getQuestion( ) method from the Exam class, and replace it with functionality to manipulate the questions indirectly. See details below on the new Exam class. You should complete steps 1-3 and verify the program is still working before continuing.
4. The fourth refactoring step is to split the Question class into three classes – Question, MCQuestion, and MCSAQuestion. This should mostly involve moving preexisting code and functionality into the new classes. At that point the new class diagram should look like:



5. Finally split Answer into Answer + MCAnswer + MCSAAnswer:



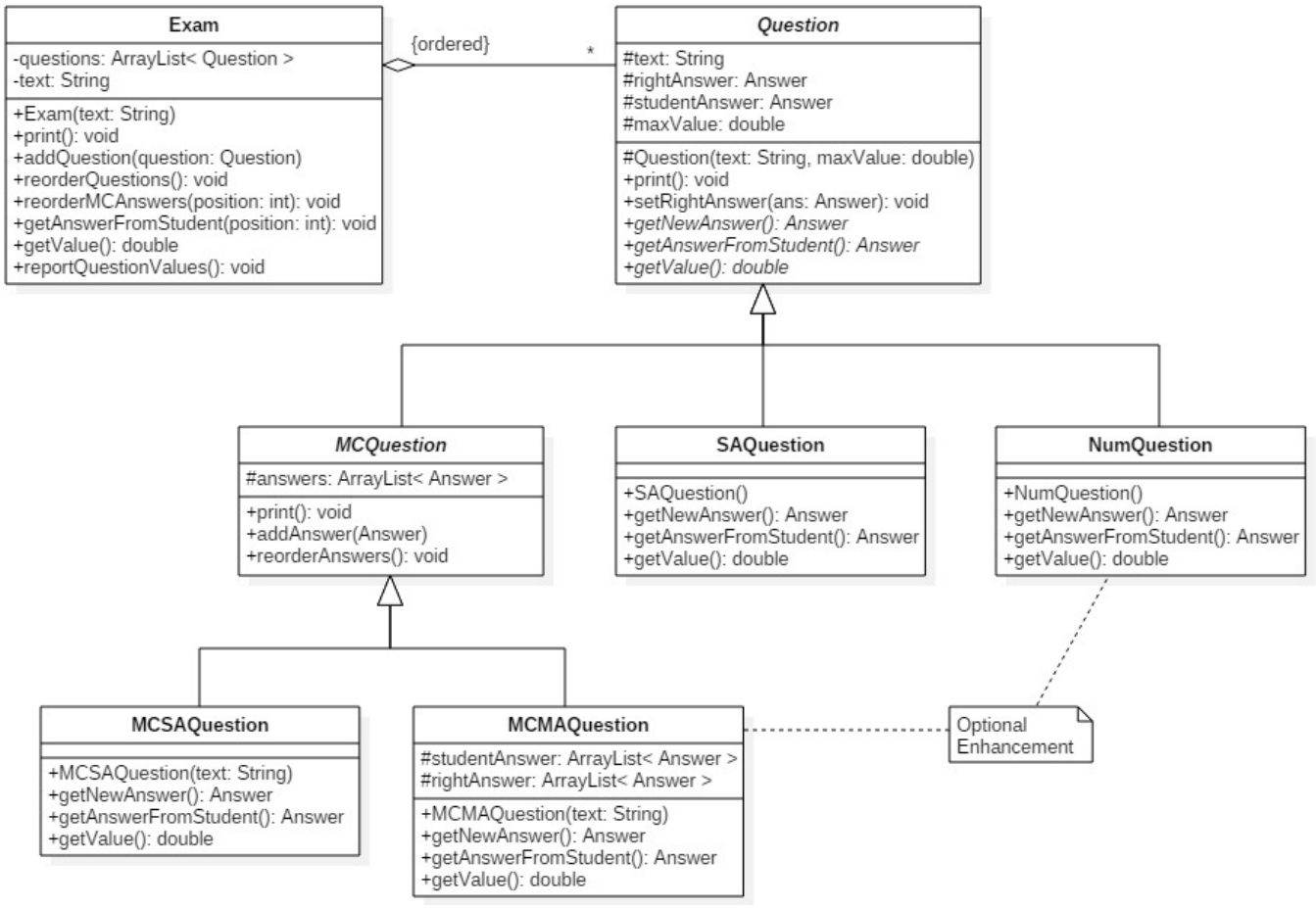
After the refactoring is complete, then new functionality can be added by developing the SAQuestion and SAAnswer classes. The other classes shown are left as optional enhancements.

Acceptance Tests

At the end of this release, the ExamTester main( ) should:

1. Create an Exam with at least two of each kind of Question.
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 the multiple choice questions. Then print the exam again.
4. Get "student" answers from the person running the program. (For testing purposes the respondent should get at least one right and one wrong of each question type.)
5. "Grade" the exam, by totaling the value of the questions, and report the overall score, along with the value contributed for each question.

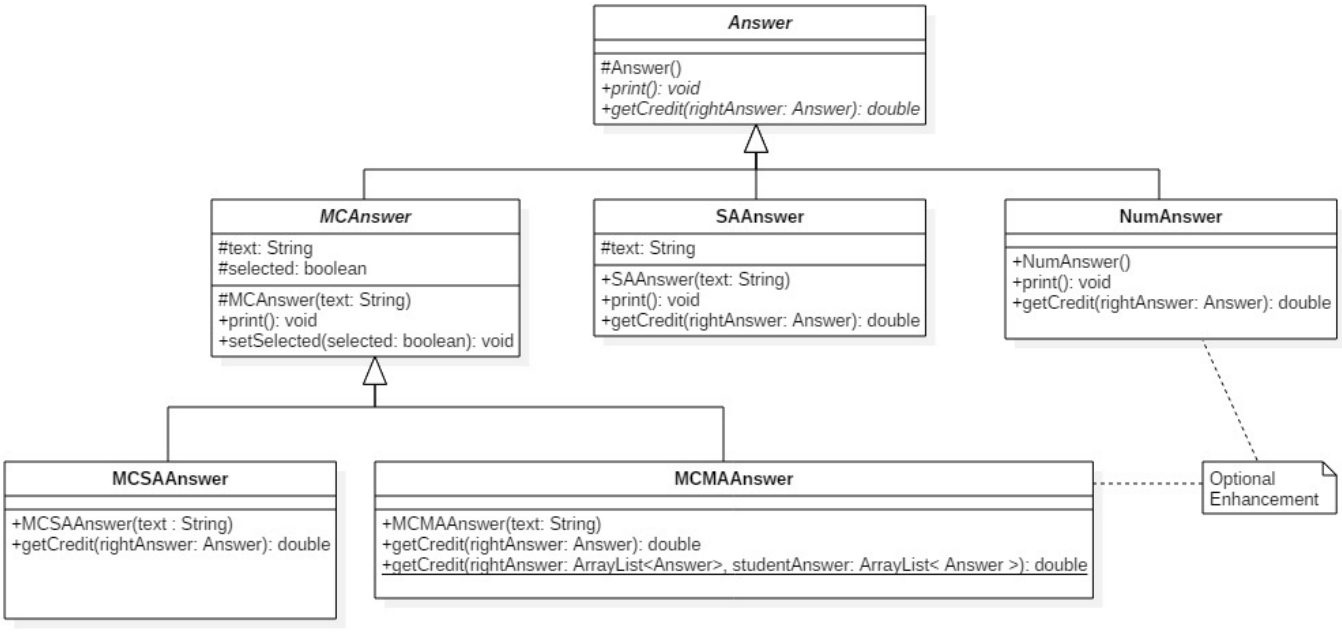
Question Subtree Class Details



The Question class is now an abstract parent class for all kinds of questions, including MCQuestion for all kinds of multiple-choice, SAQuestion for short answers, and NumQuestion for questions with numerical answers. MCQuestion has two more specific subtypes, MCSAQuestion for multiple choice with single answers ("normal" multiple choice questions) and MCMAQuestion for multiple choice with multiple answers ("Choose all that apply" or "Choose N of the following" questions.)

- Question(String, double) – A constructor for creating the Question object. The maxValue is what the question is worth if an answer receives full credit.
- print( ) : void – Prints the question to the screen. The MCQuestion class overrides this method to also print the available answers.
- **getNewAnswer ( ) : Answer** – This method creates and returns a new Answer object, suitably matched to the Question type of the object through which this method is called. Note that it is abstract in the Question class, and overridden in concrete classes to return the appropriate kind of Answer. (SAQuestion.getNewAnswer( ) returns a new SAAnswer, etc.) This mechanism is an example of the Factory Method design pattern.
- getNewAnswerFromStudent( ) : void – This also creates a new Answer of the appropriate type, similar to getNewAnswer. The differences are that this method gets the answer from the student/ keyboard, and the answer is stored in the studentAnswer field of the class instead of being returned.
- getValue(void) : double – Get the number of points that this question contributes to the exam score, based on the maxValue of the question and evaluation of the current student answer. (Answer classes have a getCredit( ) method which return a value from 0.0 to 1.0)
- AddAnswer(Answer) and reorderAnswers(void) have now been moved down into the MCQuestion class.
- selectAnswer( ) and unselectAnswer() have now been replaced with the more general getAnswerFromStudent( ).

Answer Subtree Class Details



Answer Class Details

The Answer class is now an abstract parent class for all kinds of Answers, corresponding to the matching kinds of Questions.

- print() : void – Prints the answer to the screen. Note that this is abstract in the Answer class.
- **getCredit(rightAnswer : Answer) : double** – This method compares the current answer to the correct answer (passed in), and returns a 0.0 for wrong answers, 1.0 for answers deserving full credit, and something in between for answers deserving partial credit.
  - For the MCSAAnswer class this can probably be a simple text string comparison.
  - For the SAAnswer class this could become a sophisticated (and intelligent?) evaluative comparison, but a simple (case insensitive) string comparison is probably a good start.
  - For the MCMAAnswer class an overloaded version takes two ArrayLists of Answers. This method should be static.

Exam Class Details

- There is no getQuestion( ) method. The Exam class should service requests to act upon its Questions as needed.
- reorderMCAnswers(position : int) : void – For multiple choice questions only, this method reorders the answers of the question. If the position parameter is negative, then all MC questions get their answers reordered. Otherwise the position indicates which Question should have its answers reordered, provided that that Question is a MC type question.
- getAnswerFromStudent(position : int) : void – See previous explanation of "position".
- 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.
- reportQuestionValues( ) : void – This method produces a table of the values of each Question on the Exam, and the total.

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 above under "Acceptance Tests", 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.

When creating the Exam, the ExamTester can either get input from the user or have it hard-coded or read in data from a data file – See file format below. (The getAnswerFromStudent( ) methods should read their input from the keboard.)

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.

Optional Question Data File Format

If you write your program to read in question data from a file, (highly recommended), your data file should follow the following guidelines:

- Two kinds of questions are defined for the data file, short answer and multiple choice.
- An entry for a short answer must start with "SAQuestion" on a line by itself. The question text appears on the following line, followed by the right answer on the line following that.
- An entry for a multiple choice question must start with either "MCSAQuestion" or "MCMAQuestion" on a line by itself. The following lines are identical for both types. The first line following the type is the text of the question itself, and the line after that contains an integer for the number of answers to follow. Then come a series of lines starting with a floating point value for the credit the answer is worth followed by the text of the Answer.
- All question and answer texts must be contained on a single line, of arbitrary length. A \n may be included to indicate line breaks when printing.
- Each question must be separated from the following by a blank line.
- Sample:

```
SAQuestion
What OO principle is implemented with private variables?
Information Hiding

MCSAQuestion
What do you put on a peanut butter and jelly sandwich?
5
0.5 Peanut butter
0.5 Jelly
1.0 Peanut butter and jelly
0.0 Onions
0.0 Horseradish

SAQuestion
What did 8 bytes say to the bartender?
Make us a double
```

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 or protected**, 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.)~~
- (Some of the crossed out assumptions are still true for some classes, but not all, particularly if one does optional enhancements such as the NumQuestion and/or MCMAQuestion types.)
- 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

The section above on Refactoring outlines the recommended approach to development.

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:

- The NumQuestion class for numerical questions and the MCMAQuestion class for multiple choice questions accepting multiple answers are left as optional enhancements.
  - For numerical questions some consideration must be given to how close the student answer must be to the right answer to receive credit.
  - A subclass of NumQuestion could be developed for RGNumQuestion, a randomly generated numerical question. For example, "What is the volume of a cylinder with a height of R1 meters and a diameter of R2 meters?", where R1 and R2 are randomly generated within a given range.
- Allow for certain answers that must be locked into certain positions, such as "none of the above" or "all of the above".
- Allow for multiple choice questions to have a large pool of possible answers, and only display/print a fixed-size subset of the pool. (E.g. print 5 possible answers from a pool of 20) Reordering the answers may change which wrong answers are displayed, but the right answer must always be included.
- The short answer question type has the potential for quite sophisticated answer evaluation in its getCredit method, including synonyms, misspellings, or even AI.