CS 342 – Software Design – Spring 2018
Term Project Part IV
Developing Applications Using Exam-Related Classes

Due: **Wednesday 11 April. Electronic copy due at 3:30 P.M.**
Optional paper copy may be handed in during class.

Overall Assignment

Up until this point we have been developing a set of Exam-related classes, and testing them with a test driver, ExamTester. Now we are going to develop three related applications making use of the classes that have already been developed. We will also be introducing group work, and providing more leeway for groups to make design changes as needed. Each student will be responsible for providing and maintaining approximately 1/3 of the support classes (Exam, SAQuestion, etc.) and one of the following three applications:

- ExamBuilder – Used by an instructor to create and modify exams.
- ExamTaker – Used by students to take an exam and record their answers.
- ExamGrader – Used by an instructor to evaluate students' answers and determine scores.

Refactoring / Cleanup

The first step before adding new functionality is to refactor and clean up any issues that need it. In this case, the following refactorings will be needed:

- All classes need a constructor that takes a Scanner as an argument. In particular this was not previously listed for MCSAAnswer and MCMAAnswer.
- The saveStudentAnswer**s** ( ) and restoreStudentAnswer**s** ( ) methods should all end in an 'S', in all classes in which they appear.
- Include the name of the Exam file inside the student answer file, on a second line right after the student identity information. This should be compared against the Exam file in use when grading, and possibly in other relevant situations.
- Include time and date information of when a file was last saved at the beginning of each file written. In the Exam file that should go on a line right after the exam title, and in the Answer file it should go right after the line containing the Exam file name.
- The following modifications may take a little more effort:
  - Suggestion: If Question and Answer contain Strings for the label to be written in the data file, such as "SAQuestion" , then some of the code to write to files can be moved up into the Question and Answer classes. (Constructors will need to set the field appropriately.)
  - There was a question as to whether an MCAnswer saved in a file should always have a "creditIfSelected" value stored with it or if that is only saved in Exam files. If such a value is saved in Answer files, then the next question is what value to save, since that field is really only supposed to pertain to the choices available of a multiple-choice question.
  - The numQuestion and numAnswer classes will no longer be optional. This may involve significant additions, not necessarily part of the simple refactoring/cleanup step.

Division of Labor

This assignment and the next will require working together in groups, nominally containing 3 students each. Each student will be responsible for providing and maintaining approximately 1/3 of the support classes, and writing one of the three applications to be described in greater detail shortly. So one of the first things each group should do is to determine who is going to provide which classes, and who will write which application. The following division of classes is suggested, as discussed in class:

1. Answer, SAQuestion, SAAnswer, NumQuestion, and NumAnswer.
2. Exam, Question, and MCQuestion.
3. MCAnswer, MCSAQuestion, MCSAAnswer, MCMAQuestion, MCMAAnswer, ScannerFactory.

Other divisions may be possible, so long as the division is approximately equal by some measure, and all members of the group agree that it is fair. (The classes in each group should be closely related.) When the work is handed in, a documentation file should state who was responsible for which application and which supporting classes.

Once the division of labor is determined, groups should create a new directory, populate it with the classes provided by each member of the group, and try to run an ExamTester application based on the mixed class contributions. Once that is working successfully, each member can begin work on their application, making modifications as necessary to their share of the support classes, and getting modified classes from other team members as they become available.

ExamBuilder Acceptance Tests

At the end of this release, the ExamBuilder main( ) should implement an infinite loop in which the following occurs:

1. Present the Instructor with a menu of available choices.
2. Read the Instructor’s choice, and perform the requested action.
3. Repeat from step 1 until the Instructor selects “Quit” (or something to that effect.)

- The selected choices shall include the following (at a minimum.)
  - Load a saved Exam from a file.
  - Add questions interactively.
  - Remove questions interactively.
  - Reorder questions, and/or answers.
  - Print the Exam, to the screen or to a file suitable for hard-copy printing.
  - Save the Exam, using the file format given in HW3 or a minor variation thereof.
  - Quit
- Optional Enhancement: Read in multiple Exams, and merge the questions into one Exam.
- Optional Enhancement: Read in file(s), but interactively decide which question(s) to add to the exam.

ExamTaker Acceptance Tests

At the end of this release, the ExamTaker main( ) should perform the following:

1. Get student information.
2. Load an arbitrary Exam file, following the file format specified in HW3 (as modified above) or a close variation thereof as agreed to by your group.
3. Get answers from the student:
    a. Allow students to skip answers, and come back to them later. (You may need to resolve what happens if a student wants to leave one or more questions unanswered.)
    b. Allow students to change their answers.
4. [Optional] Show confirmation of student choices (ungraded) before saving and exiting.
5. Save student answers to a file, containing also the name of the corresponding exam file as described above and in class.

- Optional Enhancement: What support can be provided for a student to repeat an exam?
- Optional Enhancement: Can the student be given the option of sorting the questions in real time, say with the unanswered questions at the beginning, while still storing the answers in the same order as in the original exam file?

ExamGrader Acceptance Tests

At the end of this release, the ExamGrader main( ) should perform the following:

1. Load up an Exam file and an Answer file, confirming that they are a matched set.
    a. If only an Answer file is provided, perhaps as an optional command-line argument, then automatically load up the corresponding Exam file.
2. Evaluate the answers, and report the results to the screen.
3. Store the results to a CSV (comma separated values) file:
    a. Student identity, total score, list of comma-separated scores for each question.
4. Optional Enhancement: Read in one Exam file and a number of Answer files, creating a table of results. The CSV file should have one line (row) for each Answer file processed.
5. Optional Enhancement: Support Excel format files as well as CSV.

Design Freedom

Your systems should use the classes developed for HW3 as a starting point, with roughly 1/3 provided by each of the group members. Beyond that you have leeway to make your own design adjustments, so long as the three applications you turn in are compatible with each other. (Files written by one of your applications should be readable by the others, using the same set of supporting classes.)

Your design choices, good or bad, may affect your grade accordingly.

Required Output

- All programs should print your name and netID as a minimum when they first start, along with the names of your group mates.
- Beyond that, each application should function as described above.

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. For this assignment each group shall submit one package of files, containing all three applications and all the supporting classes needed to build and run them. The documentation file must clearly indicate who is responsible for which application, and for which supporting classes.
2. Your code, **including a makefile and a readme file,** should be handed in electronically using Blackboard.
3. 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.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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:

- See above.
- See also previous assignments for additional ideas, or the "wish list" discussed in class.