PPA Assignment 4

This coursework is designed to test the content from Topics 1, 2, 3, 4 and 5.

Do you think there is a problem with any of the content below? Let us know immediately at programming@kcl.ac.uk.

For this week's assessment, consider the following scenario, and then complete the tasks that follow it:

We would like to create an exam attempt marker. Exams questions can be of multiple types, such as: one that requires a single numerical value as its answer, one that requires a either 'yes' or 'no' as its answer, or a multiple choice question which requires one or more of its options to be selected. An exam attempt is marked using a mark scheme, which is a special version of an exam attempt in which each question's answer is set to its correct value, and the total possible mark for the question set to its maximum. Marking an exam attempt awards a mark, out of the total number possible, for each question. Then, the sum of the mark awarded for each question is calculated to give a total mark for the attempt, out of the total possible for the exam. Finally, the attempt's total mark is converted to a degree classification; either 1.1, 2.1, 2.2 or 0.0.

1. Model this scenario based on the following requirements:

    1. The first question type is a `NumericalQuestion`. A numerical question will have one `answer`, used either for a student's attempt or for the correct value. It will also have a `mark`, used either for the mark awarded to the student in an attempt or the total mark available for the question in a mark scheme. Both of these pieces of data are whole number values. When we create a numerical question, we should be able to specify values for these pieces of data. (1 mark)

2. The next question type is a `BooleanQuestion`. A boolean question will have one `answer`, used either for a student's attempt or for the correct value. This can either be *yes* or *no*. This kind of question will also have a `mark`, used either for the mark awarded to the student in an attempt or the total mark available for the question in a mark scheme. When we create a boolean question, we should be able to specify values for these pieces of data. (1 mark)

3. The final question type is a `MultipleChoiceQuestion`. A multiple choice question will have three possible options: `option1`, `option2` and `option3`. In addition, a multiple choice question also has a `mark`, used either for the mark awarded to the student in an attempt or the total mark available for the question in a mark scheme. Each of these options can either be selected or not selected. When we create a multiple choice question, we should be able to specify values for all of these pieces of data. (1 mark)

4. Create a class to represent an `Exam`. This will be used either for a mark scheme or a student's attempt at the exam. An Exam consists of three questions as follows: `question1` which is a `NumericalQuestion`, `question2` which is a `BooleanQuestion`, and `question3` which is a `MultipleChoiceQuestion`. It also has a `totalMark`, which is used either for the mark awarded to the student for the entire exam attempt, or the total mark available for the entire exam in a mark scheme. When we create a copy/object of this class, we should be able to specify values for all of these pieces of data. (1 mark)

5. Create a class to represent a `Marker`, which will be used for the process of marking exams. A marker has two behaviours. The first is that it can `markAttempt`, which works as follows:

    1. It performs the marking procedure on a supplied `examAttempt`, using a supplied `markScheme`, then returns the total mark awarded to the attempt.

    2. For Question 1, the attempt must have the same answer as the answer in the mark scheme, in order to receive the maximum mark for this question. However, the answer in the attempt might be 1 higher or 1 lower than the answer in the mark scheme, in which case the question in the attempt is awarded 1 mark lower than the maximum mark. Furthermore, the answer in the attempt might be *up to and including* 5 higher or 5 lower than the answer in the mark scheme, in which case the attempt receieves 1 mark for this question. Otherwise, the question is awarded a mark of 0.

3. For Question 2, the attempt must have the same answer as the answer in the mark scheme in order to receive the maximum mark. Otherwise, the question is awarded a mark of 0.

4. For Question 3, each option in an attempt that is the same as its corresponding option in the mark scheme results in one mark being awarded to the question in the attempt. This means that the first option in the attempt must be the same as the first option in the mark scheme in order to get one mark, the second option in the attempt must the the same as the second option in the mark scheme in order to get one mark, and the same is true for the third option.

5. The mark awarded for each question in an attempt must be stored in that question, and the `totalMark` of the attempt should be updated accordingly.

(2 mark)

6. A `Marker` can also `convertMarksToClassification`, which converts a supplied `mark` number to a numerical degree `classification`, which is then returned. This is done using the following supplied values: `firstBoundary`, `upperSecondBoundary` and `lowerSecondBoundary`. In order to obtain a classification of *1.1*, the mark must be at least the value in `firstBoundary`. A mark of less than this but higher than or equal to `upperSecondBoundary` is awarded a classification of *2.1*. A mark of lower than this but at least equal to the value in `lowerSecondBoundary` is awarded a classification of*2.2*. Any lower mark number results in a fail, represented as a classification of *0.0*. (2 mark)

2. Create a class `MarkExams`, which can be compiled and run from the command line. Use this class to do the following (in order), using the classes and methods you have created for Question 1.

1. Create a `NumericalQuestion` and name the variable holding it *nqMarkScheme*, to be used for the mark scheme. The answer for this question is *198* and the maximum mark available is *6* (1 mark)

2. Create a `BooleanQuestion` and name the variable holding it *bqMarkScheme*, to be used for the mark scheme. The answer for this question is *true* and the maximum mark available is *1*. (1 mark)

3. Create a `MultipleChoiceQuestion` and name the variable holding it *mcpMarkScheme*. This will be used for the mark scheme. Set the mark as 3 and the options as follows:

1. option1 as false,
2. option2 as false,
3. option3 as false,

(1 mark)

4. Create a mark scheme and name the variable holding it *markScheme*. It's set of questions are made up of the questions defined for the mark scheme in the previous sub-questions. (1 Mark)

5. Create a `NumericalQuestion` and name the variable holding it *nqAttempt*, to be used for a student's attempt at the exam. The answer for this question is *196* and the mark is 0 (because the question has not yet been marked). (1 mark)

6. Create a `BooleanQuestion` and name the variable holding it *bqAttempt*, to be used for a student's attempt at the exam. The answer for this question is *true* and the mark is *0* (because the question has not yet been marked). (1 mark)

7. Create a `MultipleChoiceQuestion` and name the variable holding it *mcpAttempt*. This will be used for the exam attempt. Set the mark as 0 (because the attempt has not yet been marked) and set the (attempted) options as follows:

   1. option1 as false,
   2. option2 as false,
   3. option3 as false,

(1 mark)

8. Create an exam attempt and name the variable holding it *examAttempt*. It should be made up of the attempts at questions defined in the previous subquestions. (1 Mark)

9. Perform the marking process on the exam attempt. For each question, print out the mark awarded and the total possible for that question e.g., "Question 1: 1 out of 1". (1 mark)

10. Print the total mark awarded for the entire attempt. Then, convert this total to a classification and print this classification, using the following grade

boundaries:

- A mark of at least 9 is awarded a classification of 1.1.
- A mark that is lower than the previous boundary, but is at least 7 is awarded a classification of 2.1.
- A mark that is lower than the previous boundary, but is at least 6 is awarded a classification of 2.2.
- A mark lower than the previous boundary is awarded a classification of 0.0.

Once completed, submit your assignment using the link marked `Assignment 4: Nexus Submission Link' on KEATS.

You must complete the plagiarism and collusion training before submitting this assignment.

You must also submit complete documentation of your solution. You will find a sample piece of documentation in the Support section on KEATS marked `Sample Assignment Documentation'. Submit your documentation using the link marked `Assignment 4: Documentation Submission' on KEATS.

Students who do not submit documentation along with their code, or vice-versa, will receive a mark of zero.

Any submitted code or documentation that is found to be unduly similar to the code or documentation submitted by any other student(s), will result in a penalty for those involved.

Provisional mark for your code will be released on KEATS within one week of submission. Final assignment grades will be submitted to the exam board at the end of the semester, and will take into consideration the quality of your documentation and the quality of the comments written into your code directly.

For all other queries, see the Support section on KEATS, specifically the document marked `Introduction'.