Lab 06-Questions
Taylor Yee and Richard McCormick
10-25-19
Section 3

1. *Problem Statement*
   The purpose of this lab is to allow us to practice the concepts of inheritance and polymorphism by creating subclasses for three different types of Question classes, all of which are to be child classes of the Question parent class. We are required to create classes to handle numerical questions, fill-in-the-blank questions, and multiple-choice (MC) questions with the possibility of more than one correct answer.

   Important features of this problem:
   a. NumericQuestion class, which needs to handle numerical questions with an accuracy of 0.01
   b. FillInQuestion class, which needs to handle the user typing in a phrase that must exactly match the correct answer provided by the test code
      i. The question and answer will be provided in a single string, unlike other questions
   c. AnyCorrectChoiceQuesiton class, which needs to handle MC questions that:
      i. Require a numerical choice response
      ii. May have more than one correct answer
      iii. Must have all the correct answers in a single string variable, separated by spaces
         - Here, we realized that the **contains()** method would be useful
   d. We are not allowed to change any of the source code (Question.java, ChoiceQuestion.java, and QuestionsApp.java)
   e. The **extends** and **super** keywords, which are paramount to the concept of inheritance and allowing us to reuse code

2. *Planning*
   When we realized that all of our question classes would be derived from the Question class, we decided that looking at the source code first would be the best approach. From there, we could see the basic methods that our classes would inherit, as well as understand how the test code would create question objects. For example, regarding the NumericQuestion class, we knew that we would have to override the **setAnswer** and **checkAnswer** methods in order to handle mathematical operations. Regarding the FillInQuestion class, we realized that we would have to split the text provided to us into the question and answer parts, which meant that we would have to override the **setText,**

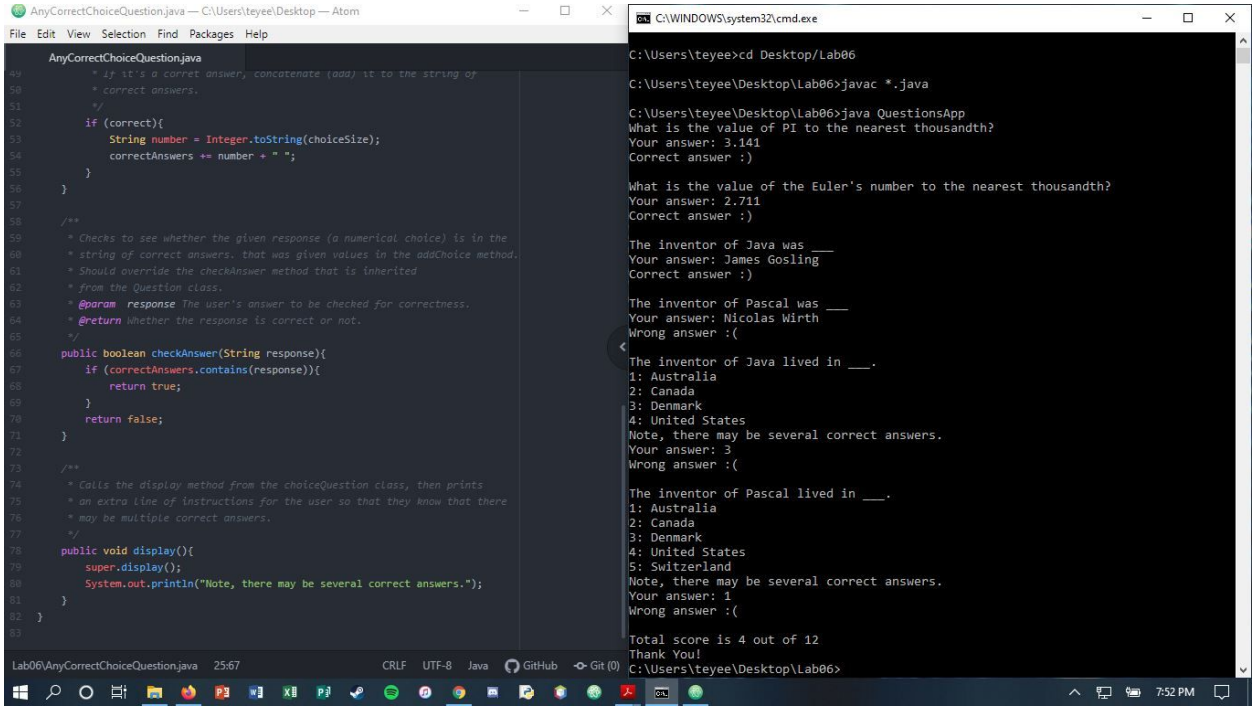
checkAnswer, and **display** methods. Finally, with the AnyCorrectChoiceQuestion class, we had to first look at the ChoiceQuestion class, which our class was immediately derived from. After examining the functionality of this class, we decided that we needed to alter it so that we could accommodate multiple correct answers, concatenate those into an answer string, and then check to see whether the user's response was in the string of correct answers. This meant overriding the **addChoice**, **checkAnswer**, and **display** methods, but here, we could also use the **super** keyword so that we did not have to retype code that was already in place.

3. *Implementation and Testing*

   When it came time to implement our approach, the NumericQuestion class was fairly straightforward. When we were checking our answer, however, we needed to convert the user's response to a double, and then take the absolute value of $response - answer$ so that the answer could be either over or under the correct answer by 0.01.

   The FillInQuestion class was also fairly straightforward. We initialized instance variables for the question and answer, and then were able to split the provided text into an array with two elements: the question and the answer. These values were stored in our instance variables, and the question was used in the **setText** and **display** methods, while the answer was used in the **checkAnswer** method.

   With the AnyCorrectChoiceQuestion class, we created instance variables for a choiceSize integer variable and a correctAnswers string. In the **addChoice** method, we invoked the method of the same name from the ChoiceQuestion class, then added 1 to the choiceSize variable to keep track of how many choices had been added. The correct answer numbers were then concatenated to the correctAnswers string, which was used in the **checkAnswer** method.



4. *Reflection*

Overall, this lab was successful in allowing us to practice using the concepts of polymorphism and inheritance. The majority of the difficulties that we encountered throughout this lab were not programming errors, but rather, conceptual errors. Understanding how inheritance and method overriding work were crucial to the working implementation demonstrated above. We used constructors to set our instance variables to default values, but this was not necessary in order for the code to work. However, this is good practice for future programs where we will need to work with larger structures and amounts of code. One alternative solution that we could have pursued would have been in our AnyCorrectChoiceQuestion class, and we could have stored the correct answers in an array to check against, instead of a correctAnswers string. This would have allowed us to use a for-loop to check the user's answer against the array, but this also would have added an extra step, and been against the requirements given to us.