Coding for the Quiz

Due Tuesday, February 07 at 8 a.m.

CSE 1325 - Spring 2023 - Homework #3 - Rev 0 - 1

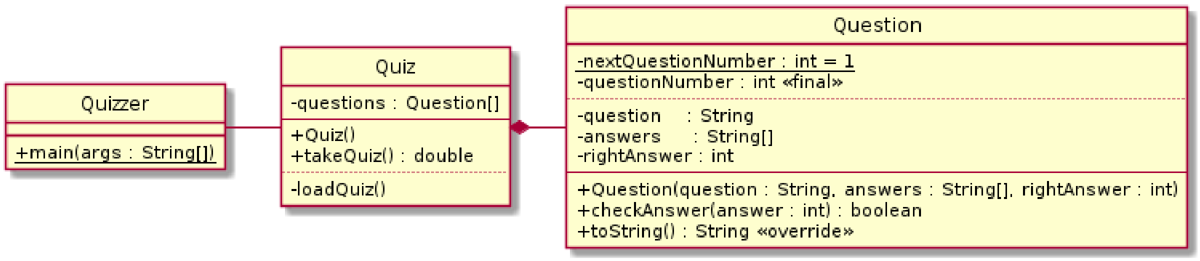
Assignment Overview

Most students are familiar with the concept of a multiple choice quiz. A question is asked, several possible answers are offered, and the student selects the answer that is "most correct". This gives us at least a couple of possible classes - the quiz itself, and each question on the quiz.

Sounds like a homework assignment!

Full Credit

In your git-managed directory **cse1325/P03/full_credit**, write the classes as shown in the class diagram and provide a build.xml file for Ant. As always, deliver your code via your GitHub repository - add, commit, and push. **Follow the class diagram closely!** Later, you will be given more flexibility but for now, follow the diagram as written.


Quizzer
+main(args : String[])
Quiz
-questions : Question[]
+Quiz()
+takeQuiz() : double
-loadQuiz()
Question
-nextQuestionNumber : int = 1
-questionNumber : int «final»
-question : String
-answers : String[]
-rightAnswer : int
+Question(question : String, answers : String[], rightAnswer : int)
+checkAnswer(answer : int) : boolean
+toString() : String «override»

Class Question represents a single question on the quiz.

- Question includes 5 *private* fields.
    - The actual `question` asked (a String), the proposed `answers` (an array of String OR an `ArrayList<String>` as you please), and the number of the `rightAnswer` (from 1 to the number of questions).
    - Questions are self-numbering. The object's question number is in *final* field `questionNumber`. Final means that once initialized in the constructor, `questionNumber` can never change. Self-numbering works using a *static* field (thus underlined in the UML) named `nextQuestionNumber`, which must be initialized in-line (that is, using `= 1` on the field itself). In the Question constructor, set final field `questionNumber` to the value of `nextQuestionNumber++`. Since a static field is shared among all objects, each instance of Question will get the next sequential number as its questionNumber!
- The Question constructor initializes the fields as discussed above. It should also perform *data validation* - if rightAnswer is less than 1 or greater than the number of answers, throw a new `IllegalArgumentException` with a clear message explaining the cause of the exception.
- The toString() method returns the questionNumber, question, and answers as an appropriately formatted String, for example:

```
1. The capital of Texas is
  1) Dallas
  2) Houston
  3) Austin
  4) Arlington
```

Class Quiz contains and gives the quiz itself.

- The `questions` array (OR `ArrayList<Question>` as you please) references one or more Question objects comprising the quiz.
- The constructor simply calls `loadQuiz()`.
- *Private* method `loadQuiz()` loads hard-coded new `Question` object references into field `questions`. Include any 2 or more questions on any topic that you like except those in the example below. (If you plan to do the bonus, save time and do it here!)
- Public method takeQuiz() prints each Question object, asks for the proposedAnswer (an int), and passes proposedAnswer to that Question's checkAnswer method. If true, the question was answered correctly (increment a counter of correct answers), if false do nothing. After all questions have been asked and answered, return the number of correctly answered questions divide by the total number of questions.

Class Quizzer contains the main method.

- Start a try/catch clause with the try.
- Instance the Quiz as variable `quiz`.
- Call `quiz.takeQuiz()` and print the score it returns along with an optional encouraging message. Multiply by 100 if you'd like the tradition grade range of 0 to 100.
- Add a catch at the end. If a Question constructor throws an `IllegalArgumentException`, print an error message to `System.err` (including the exception's message) and return a non-zero error code.

Add, commit, and push all files. Additional information that you may find helpful follows the Extreme Bonus.

Bonus

Replace the general questions you wrote in full credit with at least 5 questions on *new* topics we have covered thus far, or will cover (if you like a challenge), in CSE1325.

Extreme Bonus

You have two options.

Option 1 - Load Questions

How much more useful would the program be if it read the quiz from a text file! After opening the text file, I suggest that for each question you read the question String (`file.nextLine()`), correct answer int (`Integer.parseInt(file.nextLine())`), and then the possible answers (`file.nextLine()`) until an empty line is read. Repeat until you reach the end of file.

Option 2 - Shuffle Answers

How dreary for everyone's quiz to look identical. Shuffle the answers array or ArrayList for each question (without losing track of which answer is correct) at the start of `Quiz.takeQuiz()` or as each question is presented to the student. It's a bit harder than it looks!

Hints!

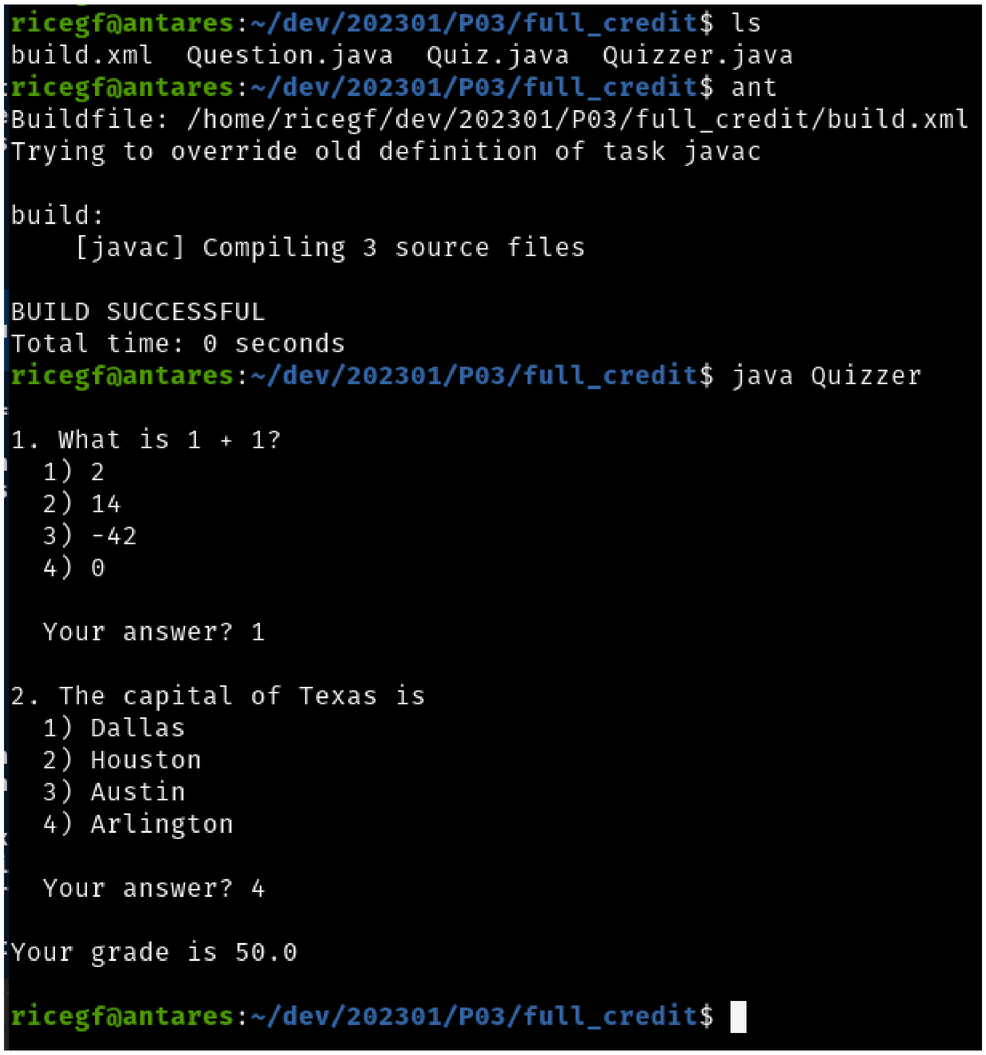
Approach

I encourage you to work in *baby steps*. Start with the *least dependent* class in the diagram (`Question`), and write a little as possible for it to compile. Once it compiles, write a little more. Keep it up! Soon it will be written.

Then move to the *next* least dependent class (`Quiz`) and repeat. Finally, write `Quizzer` and begin testing. A professional would write regression tests as they went, but I don't require that. Yet.

Have I mentioned yet that we *love* questions? And don't let me have all the fun - email *all 4 TAs* along with me each time, and use that Reply All button in Outlook!

Example Output



```
ricegf@antares:~/dev/202301/P03/full_credit$ ls
build.xml  Question.java  Quiz.java  Quizzer.java
ricegf@antares:~/dev/202301/P03/full_credit$ ant
Buildfile: /home/ricegf/dev/202301/P03/full_credit/build.xml
Trying to override old definition of task javac

build:
    [javac] Compiling 3 source files

BUILD SUCCESSFUL
Total time: 0 seconds
ricegf@antares:~/dev/202301/P03/full_credit$ java Quizzer

1. What is 1 + 1?
  1) 2
  2) 14
  3) -42
  4) 0

  Your answer? 1

2. The capital of Texas is
  1) Dallas
  2) Houston
  3) Austin
  4) Arlington

  Your answer? 4

Your grade is 50.0

ricegf@antares:~/dev/202301/P03/full_credit$ 
```

The Professor's cse1325-prof

The professor for this class provides example code, homework resources, and (after the due date) suggested solutions via his cse1325-prof GitHub repository.

If you haven't already, clone the professor's cse1325-prof repository with "`git clone https://github.com/prof-rice/cse1325-prof.git`". This will create a new directory called "cse1325-prof" in the current directory that will reflect the GitHub repository contents.

The cse1325-prof directory doesn't automatically update. To update it with the professor's latest code at any time, change to the cse1325-prof directory and type "`git pull`". If that ever fails, just delete the directory and clone it again.

Suggested build.xml

Unlike the Makefile you may have used with C, a single build.xml will cover most (not quite all) of our projects this semester. It's listed here, although you can copy it to your directory from `cse1325-prof/00/code_from_slides` more easily.

IMPORTANT: You need one build.xml file for each directory from which you will run `ant`. For most assignments this semester, you'll need a copy of the same build.xml file in the full_credit, bonus, and extreme_bonus directories. Don't forget to add them to GitHub!

```xml
<?xml version="1.0"?>
<project name="CSE1325 Project" default="build">
  <presetdef name="javac">
    <javac includeantruntime="false" />
  </presetdef>

  <target name="build" description="Compile source tree java files">
    <javac debug="true" failonerror="true">
      <src path="."/>
    </javac>
  </target>

  <target name="clean" description="Clean output files">
    <delete>
      <fileset dir=".">
        <include name="**/*.class"/>
      </fileset>
    </delete>
  </target>
</project>
```

Additional Hints

Coming soon!