Assignment Quiz

Object Orientation

Spring 2021

1 Learning Goals

After completing this assignment, you should be able to:

- Create subclasses of classes
- Use inheritance in Java
- Store objects in a List
- Iterate through a collection of objects using an enhanced for-loop

2 Quiz

For this assignment, you will implement a pop quiz that asks the user questions.

You will not focus on creating a beautiful interface or formulating the smartest question. What we care about is making different sorts of questions and finding suitable Java representations for them.

There are different sorts of questions, all of which display different answer choices and expect different response formats from the user.

3 The Question Class

A question contains at least the following methods:

```java
public String toString()
public boolean isCorrect(String answer)
public String correctAnswer()
```

- The method `toString` should return a string that can be used to display the question.
- You can use `isCorrect` to check whether an answer is correct.
- The method `correctAnswer` returns a string of the correct answer as a printable text. You can use this text to show the correct answer when the user makes a mistake.

Define an **abstract class** `Question` to store questions. This class contains at least all the methods mentioned above.

Each question has a `score`. The score is a number between 1 and 5. If the score is specified with an incorrect value, or none at all, the question should have a score of 3. Hint: implement this logic in the score's setter.

4 Open Questions

An open question is a question that expects a string as an answer.

This means that a good question from this category only has one correct answer.

Create a subclass for open questions. A well formulated question has exactly one answer. This answer is defined during the object's construction. The constructors of this class are:

```
public OpenQuestion(String question, String answer, int score)
public OpenQuestion(String question, String answer)
```

Some example questions you can use can be found in appendix A.

5 Multiple-Choice Questions

A multiple-choice question has a number of possible answers. It is a subclass of `Question`. It should have the following constructors.

```
public MultipleChoiceQuestion(String question, String[] answers
    , int correctAnswer, int score)
public MultipleChoiceQuestion(String question, String[] answers
    , int correctAnswer)
```

The integer `correctAnswer` is the index of the correct answer in the array of answers. The toString function should print all possible answers, preceded by `a)`, `b)`, `c)`, and so on. The correct answer for a multiple choice question is the **letter of the respective answer**.

Here is an example of a MultipleChoiceQuestion and the outcome of its toString method.

```
MultipleChoiceQuestion("What is the best achievable complexity
    of in situ sorting?", new String[] { "O(N^2)", "O(N log N)",
     "O(N)", "O(log N)" }, 1, 4)
```

```
What is the best achievable complexity of in situ sorting?
a) O(N^2)
b) O(N log N)
c) O(N)
d) O(log N)
```

Some example questions you can use can be found in appendix A.

6 This-Or-That Questions

This-or-that questions are a special case of multiple-choice questions with short answers, such as "yes" and "no" or "right" and "wrong". Create a subclass of MultipleChoice-Question with the following constructors.

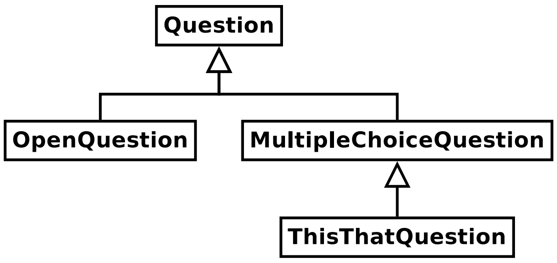

Figure 1: Class diagram for this assignment. Not everything you need is shown.

```
ThisThatQuestion(String question, String answer1, String
    answer2, int correctAnswer, int score)
ThisThatQuestion(String question, String answer1, String
    answer2, int correctAnswer)
```

For these questions, answers are not labeled, but formatted as text. Here is an example of a ThisThatQuestion and the output of its toString method.

```
ThisThatQuestion("Every class must have a constructor", "Right"
    , "Wrong", 1)
```

```
Right or Wrong: Every class definition must have a constructor.
```

The correct answer to these questions is **one of the two choices**.

7 Summary

Your class hierarchy should look like fig. 1

The functions *toString*, *isCorrect*, and *correctAnswer* are all different and must be overridden in all question classes. Read the specification carefully to understand the differences.

8 Asking a Number of Questions

Since there should not be a limit to the total number of questions, you should store them in a list instead of an array. Arrays have a fixed size, while lists can always be extended. Here is example code how to store questions in a list, and use a for-each loop to iterate over them.

```
List<Question> questions = new LinkedList<>();
questions.add(new OpenQuestion("What is the capital of France?"
    , "Paris", 3));
questions.add(new OpenQuestion("How many roads must a man walk
    down?", "42", 5));
for (Question q : questions)
  System.out.print(q);
```

This kind of for-loop can be used to conveniently iterate over all items in a collection. When possible you should always prefer it over the more cumbersome index-based iteration.

```
    for (int i = 0; i < questions.size(); i += 1)
      System.out.print(questions.get(i));
```

In the loop, your program should ask the user for an answer, read in the answer and check whether it is correct. The program should not distinguish between uppercase and lowercase letters in answers. You can use the `String` method `equalsIgnoreCase` instead of the generic `equals`.

9 Repeating Questions

Your program should ask all questions once. After that, the questions that the user has answered wrong should be repeated. There should be two rounds of questions. For this, you should add those questions to an extra list.

In the end your program should display an overview of the score from the questions that were answered correctly in the first round and the score of the second round.

10 Your Tasks

1. Implement the question class hierarchy as described above.
   - All three question classes have their own implementations of toString, correctAnswer and isCorrect.
   - ThisThatQuestion should extend MultipleChoiceQuestion.
2. Implement a class `Game` that implements the logic of asking two rounds of questions.

11 Submit Your Project

To submit your project, follow these steps.

1. Use the **project export** feature of NetBeans to create a zip file of your entire project: File → Export Project → To ZIP.
2. **Submit this zip file on Brightspace**. Do not submit individual Java files. Do not submit any other archive format. Only one person in your group has to submit it. Submit your project before the deadline, which can be found on Brightspace.

A Example Questions

These questions are also available in the project start template.

```
OpenQuestion("What is the big O complexity of binary search?",
    "O(log N)")
OpenQuestion("How would you read an integer i from scanner s in
     Java?", "i = s.nextInt();", 2)
```

```
OpenQuestion("What is the minimum amount of constructors you
    have to define for a class in Java?", "0", 2)

MultipleChoiceQuestion("What is the best achievable complexity
    of in situ sorting?", new String[] { "O(N^2)", "O(N log N)",
     "O(N)", "O(log N)" }, 1, 4)
MultipleChoiceQuestion("How do you print \"Hello world\" on a
    line in Java?", new String[] { "System.out.print(\"Hello
    world\");", "System.out.println(\"Hello world\");", "cout <<
     \"Hello world\";" }, 1)
MultipleChoiceQuestion("How do you read a non-empty word in
    Java using scanner s?", new String[] { "s.nextline()", "s.
    next(\"\\S+\")", "s.next(\"\\a*\")", "s.next(\"\\S*\")", "s.
    next(\"\\\\s+\")", "s.next(\"\\s+\")", "s.nextString(\"\\s
    *\")", "s.next(\"\\\\S+\")", "s.nextString()" }, 7, 1)

ThisThatQuestion("Every class must have a constructor", "Right"
    , "Wrong", 1)
ThisThatQuestion("Is there a difference between an interface
    and an abstract class?", "Yes", "No", 0, 5)
ThisThatQuestion("Is there a maximum to the amount of
    constructors a class can have in Java?", "Yes", "No", 1)
```