Indiana University
CSCI-C212: Introduction to Software Systems (S25) Lab 10
Joshua Crotts and Muazzam Siddiqui

Even More Object-Oriented Programming

What To Do:

Follow each step carefully. As you complete the lab, submit the source files (`.java`) problems to the autograder. After finishing, please let one of the AIs know.

You must write sufficient tests and documentation. Not writing tests will result in a zero.

Problem 1.

In this exercise, you'll be developing a `Document` interface along with its implementing classes:

- `TextDocument`
- `SpreadsheetDocument`
- `PresentationDocument`

The `Document` interface is defined as follows:

```
interface Document {

  /**
   * Returns the number of pages in this document.
   */
  int numberOfPages();

  /**
   * Returns a string representing that the Document
   * is being printed.
   */
  default String print() {
    return "Printing the document!";
  }
}
```

Notice that we have a `default` method, which is one that an implementing class does **not** have to implement. It provides "default" functionality, should the "implementee" not want to implement the method (hence the name!).

(a) Implement the other three classes with the following specifications:

- A `TextDocument` consists of 100 pages. When it is printed, it should return a message `"Printing text document!"`.
- A `SpreadsheetDocument` has 50 pages. When it is printed, it should return a message `"Printing spreadsheet document!"`.
- A `PresentationDocument` contains 20 pages. It utilizes the default implementation of the `print` method.

(b) Design the `PrintingOffice` class, which includes the following static method: `static OptionalDouble avgPages(List<Document> lodocs)`. This method calculates and returns the average number of pages across the provided list of `Document` objects. Remember why we use `Optional`: if there are no `Document` objects in the list, we would be dividing by zero if we took the average!

(c) Inside the `PrintingOffice` class, modify it to include the `static void printDocuments(List<Document> documents)` method, responsible for invoking the `print` method on each object in the list of `Document` instances.

Problem 2.

In this question you will design a series of classes to represent different question types for an exam. Consider the following class definition for a `Question`, which stores the prompt for the question as well as its answer. Notice that the class has two constructors: one for when the answer is known at the time of instantiation, and one where it is not. We can set the answer to the question via its respective setter.

```
class Question {

  private final String PROMPT;
  private String answer;

  Question(String prompt, String answer) {
    this.PROMPT = prompt;
    this.answer = answer;
  }

  Question(String prompt) {
    this(prompt, null);
  }

  /**
   * Determines whether a given answer is the correct answer.
   * @param ans - answer TO the question itself.
   * @return true if ans is correct, false otherwise.
   */
  boolean isCorrect(String ans) {
    return this.answer.equals(ans);
  }

  @Override
  public String toString() {
    return this.PROMPT;
  }

  // Getters and setters omitted.
}
```

(a) Design the `ChoiceQuestion` class, which inherits from `Question`. This class receives only the prompt for the question in its constructor. To store a collection of possible choices, we will use a `Map<String, Boolean>` whose keys are the choices and whose values are whether or not the answer is correct. Design the `addChoice(String choice, boolean isCorrect)` method, which adds the choice to the map as a pair. You should call `super.setAnswer` when `isCorrect` is `true`. Finally, override `toString` to return not only the prompt, but also the choices on separate lines. This class should contain very few lines of code, no more than 35 (and that's with lines for spaces, braces, and so forth). Hint: use a `LinkedHashMap` to preserve insertion order.

```
ChoiceQuestion q1 =
    new ChoiceQuestion("What is the capital of North Carolina?");
q1.addChoice("Charlotte", false);
q1.addChoice("Raleigh", true);
q1.addChoice("Winston Salem", false);
q1.addChoice("Columbia", false);

q1.toString() =>
What is the capital of North Carolina?
Charlotte
Raleigh
Winston-Salem
Columbia

q1.isCorrect("Raleigh")   => true
q1.isCorrect("Charlotte") => false
```

(b) First, design the `TrueFalseQuestion` class, which inherits from `ChoiceQuestion`. The constructor should invoke `super.addChoice` to add `"true"` and `"false"` as possible options. Do *not* call `this.addChoice` on instances of this class. The starter code, in fact, prevents you from doing so with an exception (which we will learn about in two weeks)! This class should contain very few lines of code, no more than 25 (and that's with lines for spaces, braces, and so forth).

```
ChoiceQuestion q2 =
    new TrueFalseQuestion("The square root of 2 is rational.",
                          false)

q2.toString() =>
The square root of 2 is rational.
true
false

q2.isCorrect("true")  => false
q2.isCorrect("false") => true
```

(c) Second, design the `FillInBlankQuestion` class, which inherits from `ChoiceQuestion`. With a "fill in the blank" style question, there are many equivalent correct answers that students could provide, as we will exemplify. Therefore, you should overload (note: **not** override) the `addChoice` method to only receive a string, since any answer marked as a possible answer is correct. Your overloaded version of `addChoice` should call `super.addChoice` with the provided answer and `true`. You should also override `isCorrect` by checking to see if the given answer exists in the map of choices (use `super.getChoices()`. In this class, we prevent you from calling `setAnswer` or `getAnswer` with an exception. This class should contain very few lines of code, no more than 25 (and that's with lines for spaces, braces, and so forth).

```
FillInBlankQuestion q3 = new FillInBlankQuestion("2 + 2 = _________");
q3.addChoice("4");
q3.addChoice("four");
q3.addChoice("FOUR");
q3.addChoice("4.0");

q3.toString() =>
2 + 2 = _________
4
four
FOUR
4.0

q3.isCorrect("4")        => true
q3.isCorrect("four")     => true
q3.isCorrect("FOUR")     => true
q3.isCorrect("4.0")      => true
q3.isCorrect("sqrt(16)") => false
```

As a follow-up question, consider why we *must* initialize `q3` as a `FillInBlankQuestion` rather than `ChoiceQuestion`. What would happen if we did the former?