CSCI 3400 :: Object-Oriented Design
Fall 2022
Lab 04: Design Principles and Another(?) Pattern
Assigned date: 09/20/2022
Due date: 09/28/2022 (10:30 PM, EST)
Total points: 50 points

- Implement/Write code as you go along with the problem description. Complete the problems as they appear.
- Keep notes on the questions asked in a file called **note.txt**.
- Submit all of the **code** and the **note.txt** file by placing them in a zipped folder.

Problem Description:

Assume that we are part of a Software Developer team. We work in collaboration with the Software Design team and the Software Testing team.

Scene 0 (Developer Team):
Last year we built a package called `**robot**` (check the **robot** package in the starter code), that has an abstract class (Robot) and several concrete child classes (MathematicianRobot, NLPResearcher).

Now, a new US client (let's call "client us1") wants us to design a new class for them that will be creating different types of Robot objects upon request/input. After creating a robot, they also want to test the functionality of the produced robots. So, we have written a class `USRobotProducer` which has two methods:

`buildARobot`: it takes a name and type clue and produces (returns) a new robot of the specified type.
`testYourRobot`: it takes a robot object and displays its expertise.

This code is provided (**lab04starter** package) and it satisfied the US client.

Scene 1 (Developer Team):

In a few days, we received a similar request from a new client from China. They want us to design a class that creates only mathematician robots. They also want a method to test the functionality of the robot. It seems similar to what we did for the US client. Maybe we should create a new class and deliver the code to our Chinese client. So, we asked one of our teammates to complete the task as it seems similar to some old projects. While writing the code for this client from China, the assigned teammate named the class `ChineseRobotIndustry` which has the following two methods:

`deliver`: it takes a name and produces (returns) a new Mathematician robot.
`test`: it takes a Robot object as input and displays its expertise.

> **(5 points) To Do:**
> Write the `ChineseRobotIndustry` class with expected methods (`deliver` and `test`) in it.

Scene 2 (Testing Team):
The testing team needs to develop code (`DemoTesting` class in the `lab04starter` package) for testing the `USRobotProducer` and `ChineseRobotIndustry`. Assume, people in the testing team had not written the code for the `USRobotProducer` and `ChineneRobotIndustry`; meaning they are not familiar with the method names and parameters.

> **(10 points) To Do:**
> What challenges do the testing team face initially? Write your answers in `note.txt` file.
> Write necessary code to test both classes (`USRobotProducer`, `ChineseRobotIndustry`).

Scene 3 (Developer Team):

In a few months, we receive a request from another US client, (let's call "client us2"), to create a Nurse robot (which should inherit from the Robot class) and we fulfill their request. The nurse robot can only greet the patient with a short message.

> **(5 points) To Do:**
> Implement the NurseRobot class to fulfill the request of client us2.
> Add necessary lines in the `DemoRobot` class to test this new type of robot. (Assume that you are working for the Testing team in this case.)

Scene 4 (Developer Team):

Now, "client us1" comes with a complaint: with the existing code they have, they can only produce MathematicianRobots or NLPResearchers, but not Nurse robots. They want us to fix the code for them. While fixing the code, you noticed that we were not following an important principle.

> **(5 points) To Do:**
> Which important design principle (out of the five stated in SOLID) did we violate?

Scene 5 (A joint meeting between Design and Developers):
Our team has a meeting today where we identify the following issue:

Different clients may ask for similar requests. Instead of coming up with different solutions for each of them, it might be beneficial to follow a particular standard. That way, it can help us retract the code later, write testing codes more efficiently, and fix future issues.

> **(20 points) To Do:**
> At this point, you identify that the problem fits a design pattern; many complications can be avoided if that pattern were used. You discuss that with the team and redesign and reimplement the code (for the US client and the Chinese client).
> Instead of changing the starter code, create a new package, **lab04solution**, and put the re-modeled code here. Import the **robot package** if necessary.

> **(5 points) To Do:**
> Before submitting the code, add JavaDoc comments for each of the classes.