CS1331 Homework 04

In this assignment, we will be testing your understanding of class design and inheritance.

Introduction

The Olympics happen every 2 years, and the Olympic commission is planning ahead for the 2020 Olympic Games! However, the Olympic commission has become wracked with corruption and as a result the event is being terribly mismanaged. This is especially evident in the Java program they're using to manage the people at the event. As a promising Georgia Tech undergrad, they have hired you to work (for free) to fix their computer system and save the Olympics!

Problem Description

The current codebase for managing the people at the Olympic event violates some of the principles of good class design, namely Single Responsibility and Open/Closed. Your job is to refactor this code base to conform to these principles. This assignment is fairly open ended, so you have freedom to create new files and name them as you like. Consequently, most of your grade on this assignment will be on your design choices, and your explanation of those design choices.

Background

Before beginning this assignment, it would be wise to familiarize yourself with the first two SOLID design principles (Single Responsibility and Open/Closed). To give you a general idea of each principle, we will summarize them briefly in this section, but don't be afraid to branch out and do some of your own research!

Single Responsibility Principle

This principle is fairly intuitive. It basically means that each class should only really be responsible for managing and manipulating its own information, and there shouldn't be that much information to manage. Essentially, this principle suggests the use of many smaller classes, each with a very specific purpose over the use of larger, broader classes. Doing this helps to make the code more modular, reusable, and extensible.

Open/Closed Principle

The short form of this principle would be, "Open for extension, but closed for modification." Essentially, it means that when you write a class, you're done. When you need new features, this means you can't go back and add code to that class, and this is for good reason. If for every time you needed more features you just tacked them onto the same class, you would soon have a very broad class that is doing too many things. Moreover, requiring everyone on a team to edit the same file makes work a bit of a hassle. Instead, it's better to close classes for modification, but leave them open for extension. This means that subclassing a class is OK when you need to add more features or change functionality.

Solution Description

Athlete.java

- Right now, due to the terrible chaos that has engulfed the Olympics, they are only supporting Tennis Players, Soccer Players, and Javelin Throwers. To represent these different athletes, they are using the file Athlete.java, and have been **modifying** it as they add support for different kinds of Athletes. Can you spot which of the SOLID principles they are violating by having done this?
- Refactor Athlete.java to make it conform to the SOLID principles. Feel free to make additional classes and redistribute the code in Athlete.java. You should not be adding additional methods including getters/setters/constructors.
  - Hint: pay attention to what the play method is doing, and what it is using that private instance field for. You should definitely be able to improve the implementation of this method.
  - Hint: what are things all Athletes should be able to do? What things are specific to specific kinds of Athletes?
- Each type of Athlete should also properly override the Object `equals` method such that two Athletes of the same type that have the same name should be considered equal.
- In the class JavaDoc for Athlete.java, briefly explain what SOLID principle(s) the original implementation was violating and how, and also how your refactoring has solved it (answer in 2-5 sentences, so don't write an essay please).
- Now, use your new implementation to add support for Hockey Players. Hockey Players should have some special instance field, a getter for it, and they should have some special method that prints something out (Use your imagination. We are grading more on your class design, and are not particularly concerned about your implementation of this method. It doesn't even need to be a Hockey Player if you don't want. It could be a StarCraft player. What a time to be alive!).

EventManager.java

- The EventManager is a really busy guy. Because of budget cuts, he's basically running the whole show and has a ton on his plate. He's in charge of training the Athletes, cooking for them, making sure they're not cheating, and organizing the games. Often times he gets home late at night, and his health and married life have been suffering. He decided he needs to reevaluate where he is in life and has turned in his two week notice.
- Do you notice which of the SOLID principles has been violated by assigning so much work to the EventManager?
- Refactor EventManager to make the code conform to the SOLID principles. Feel free to create additional classes as you please. You do not need to create any additional methods for this class.
  - Hint: You may not even need the EventManager class in the end. Submit this file regardless though, and in its class JavaDoc, write a short explanation like the one for Athlete.java.

Olympics.java

- Now, create a file called Olympics.java to test your code. This file should be the main entry point for your program.
- Make an array that has one of each kind of Athlete.

- Iterate over that array, and call the play method on each Athlete.
- Use Athletes in that array to call all the public methods that were in EventManager (and may still be in EventManager or new classes) in the original implementation.

Javadocs

You will need to write Javadoc comments and watch for checkstyle errors with your submission.

- Every class should have a class level Javadoc that includes `@author <GT Username>` and `@version <version number>`.
- Every public method should have a Javadoc explaining what the method does and includes any of the following tags if applicable:
    - `@param <parameter name> <brief description of parameter>`
    - `@return <brief description of what is returned>`

See the CS 1331 Style Guide on Canvas for details.

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error, with the points deducted being capped.

For this homework the **checkstyle cap is 50**, meaning you can lose up to 50 points on this assignment due to style errors. This limit will increase with each homework.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on Canvas that includes xml configuration file specifying our checks. To check the style of your code run `java -jar checkstyle-6.2.2.jar *.java`.
- To check your Javadocs run `java -jar checkstyle-6.2.2.jar -j *.java`.
- Note that the command for checking code and the command for checking Javadocs are different. You will have to run both commands to fully test for style errors.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the Customization Tips on Canvas for more information.

Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homework
- Parts of the homework specification you are unsure of and need more explanation

- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may not discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

OKAY: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"

BY NO MEANS OKAY: "Hey... the homework is due in like 20 minutes... Can I see your code? I *promise* won't copy it directly!"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Submission

- Submit **ALL** of your Java files for this homework as attachments to the `Homework 4` assignment on Canvas. You can submit as many times as you want, so feel free to submit as you make substantial progress on the homework. We only grade your **last** submission, meaning we will ignore any previous submissions.
- If you submit multiple times Canvas will append a number to your Java file (`Athlete.java` becomes `Athlete-1.java`). **Do not worry about this, we will fix the file name before compiling and running your code.**
- Non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted ALL of the necessary files by re-downloading your file(s) and compiling/running them.

Good Luck! \(°□°)/