OOP and Shape Representations [Due: March 30 at 11:59 PM (EST) on OAKS]

This assignment is designed to familiarize you with Object-Oriented Programming (OOP) concepts using the topic of shapes. You will become more familiar with writing Java classes and implementing instance methods that update or access instance variables. You will also apply method overloading, method overriding, and class inheritance. These OOP concepts are extremely important, practical, and fun to play around with. Enjoy!

VERY IMPORTANT: Your method headers (i.e., return type, method name, and parameter list) should match the ***exact*** description in the handout and starter code, including the ***case sensitivity*** of any method's name; otherwise, our testing scripts will fail and you risk getting 0 points.

Part 1: Complete these simple Java classes

Your first task is to complete the implementation of the following classes that represent different shapes, by filling in the missing methods in each class. For some methods you will need to either complete or define the signature of the method as well. Pay careful attention to the expected *name* of the method, the order and type of *parameters*, and the method's *return type*, since an incorrect method signature will cause our testing code to fail when running against your submission.

Note that the missing methods in the code are marked by a `TODO` prefix. Make sure to remove the `TODO` tags once you're done coding and testing your implementation. This is common practice when you are coding a problem in the real world. Complete all missing code in the following classes:

- Class **Circle** is defined inside Circle.java and is meant to represent a circle shape.
- Class **Rectangle** is defined inside Rectangle.java and is meant to represent a rectangle shape.
- Class **ShapeTester** is defined inside ShapeTester.java and is meant to compare different objects (i.e. instances) of shapes.

Part 2: Implement object inheritance in Java

It is often the case that the classes you write to represent real-world data types will inherit properties (data and methods) from other classes. In this assignment, your second task is to write a new class called **Cylinder** and save it in a file named Cylinder.java. Class Cylinder must `extend` the class Circle and satisfy the following requirements:

- The center of the cylinder should have an `x` coordinate, a `y` coordinate, and the `z` coordinate should be 0. It should also have a `height`, which is how tall the cylinder is in the z direction. All coordinates and dimensions should be of type `double`. Think carefully about which variables you will inherit from class Circle and, therefore, which new ones will you need to define in class Cylinder.
- There should be two constructors in class Cylinder. The first one takes no arguments and sets the center of the base at (0.0,0.0,0.0), the radius of the base at 1, and the height at 1. The second constructor takes as input the x and y coordinates, the radius of the cylinder, and the height of the cylinder.

- Your class should define the methods `getHeight` (which returns the cylinder height), and `setHeight` (which updates the cylinder height).
- Your class should override the `getArea` method from class Circle to make it return the area of the Cylinder object instead. Your method implementation here must utilize the getArea computation already defined in the super class Circle. (You may invoke other methods from the super class Circle as well, if needed.)
- Your class should define the method `getVolume` which returns the volume of the Cylinder object (as a `double` value).

Testing and submitting your work

You can define a main method in each class you complete or write, in order to test the different methods you have implemented in that class. You should try different test cases and troubleshoot your code using inputs you know the answers to. Your code will be assessed by a main method that you do not have access to. Remember that your methods must have signatures (headers) that **match exactly the provided descriptions**, otherwise our testing code will not be able to call your methods correctly, resulting in a zero grade.

The Circle class has the most information filled in, followed by the Rectangle class. The **ShapeTester** class only has method descriptions, and it is up to you to write the Cylinder class. To submit this assignment, you must upload four files on OAKS: **Circle.java**, **Rectangle.java**, **ShapeTester.java**, and **Cylinder.java**. Make sure all files compile and run successfully before you submit them.

Collaboration policy

This is an assignment to be completed individually. General discussion among students in this class and elsewhere, about how to use Java, how to create zip files, etc. are acceptable. But no discussion/sharing of how to solve the problem is allowed. No internet or other searching or asking for solutions is allowed. You may post general questions, but no code, to Piazza. Questions asking for clarification of the specifics of the problem are allowed.

Grading

If a program does not compile, you will get 0 points for it (NO EXCEPTION – Emails that ask for an exception will be disregarded). Programs that do compile successfully will be executed with different test cases. The breakdown of marks is as follows (detailed breakdown of the marks is available inside the starter code files):

- Class Circle correctness [20 points]
- Class Rectangle correctness [20 points]
- Class ShapeTester correctness [20 points]
- Class Cylinder correctness [35 points: 8 points for proper class and instance variables declaration, 6 points for constructors, 12 points for getArea, 9 points for remaining methods]
- Code clarity and style (add meaningful comments when appropriate!) [5 points]

Honor Code

Solve this assignment **individually**; do not collaborate with classmates or look for online solutions. **We check for code plagiarism**. The assignment is governed by the College Honor Code and Departmental Policy. Remember, any code you submit must be your own; otherwise

you risk being investigated by the Honor Council and facing the consequences of that. Finally, please remember to have the following comment included at the top of the file:

```
/*
THIS CODE WAS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING
CODE WRITTEN BY OTHER STUDENTS OR COPIED FROM ONLINE RESOURCES.
_Your_Name_Here_
*/
```

Program Submission

Create a **ZIP file** that only contains .java files and txt files (no other files types). **DO NOT** include a file hierarchy (i.e. the package structure) – just one or more .java files. For the ZIP file you must use the naming convention: <Lastname><Firstinitial>.zip (e.g., **HashemiN.zip**) If the assignment is not submitted in the correct format – **it will not be graded – no exceptions!!** Submit the ZIP file via OAKS in the Dropbox that corresponds to the assignment. Resubmit as many times as you like, the newest submission will be the graded submission.

Note – to create such a zip folder:

1. Create an empty folder with the name <Lastname><Firstinitial>
2. Move your HW1.java file and any text files into it.
3. Zip/compress it into <Lastname><Firstinitial>**.zip**
4. Upload to OAKS.

To test if you have created a proper submission file, assuming your name is Hans Nero:

1. Unzip NeroH.zip
2. You should now see a folder named NeroH
3. Look in NeroH and see ONLY java and text files you wish to submit.
4. If 1, 2, & 3 succeeded, success! Submit NeroH.zip

Late assignments will NOT be accepted (NO EXCEPTION – Please do not email us your assignment after the due date; it will be ignored).

Please feel free to setup an appointment or drop by during office hours to discuss the assigned problem. I'll be more than happy to listen to your approach and make suggestions.

Required program documentation:

Header comments at the start of each file must include:
Your name
HW #, Class Section #, Spring 2021
A one- to three-sentence description of the purpose of the code in this file.
For any file that you edit or write, include a comment describing how you completed the file. Here is an example:
This code was written by me alone, but I did discuss the problem with tutors at the Center for Student Learning.

- Note: you do not have to document that you talked to the instructor or the TA.