CS 171: Introduction to Computer Science II
Assignment #1: OOP and Shape Representations

[Due: on Gradescope]

[Late Submission: See Syllabus for policy details.]

Submission Instructions: This programming assignment should be submitted electronically on **Gradescope**. Make sure that all Java files you submit can be compiled and run against Java Development Kit 11 (or higher). You can submit *as many times as you want before the deadline.* Your assignment will be partially autograded on Gradescope once you submit so you will receive the results as soon as the autograder finishes running.

Goals: 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!

!! 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 Gradescope 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 **Sphere** and save it in a file named Sphere.java. Class Sphere must `extend` the class Circle and satisfy the following requirements:

- The center of the sphere should have an `x` coordinate, a `y` coordinate, and a `z` coordinate. It should also have a `radius`. 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 Sphere.

- There should be two constructors in class Sphere. The first one takes no arguments and sets the center of the sphere at (0.0,0.0,0.0) and the radius of the sphere to 1. The second constructor takes as input in the *exact* following order: the x, y, and z coordinates and then the radius of the sphere.
- Your class should override the `getCenter` method from class Circle so that it returns the 3-dimensional center of the Sphere object (x, y, z) as an array of doubles.
- Your class should overload the `setCenter` method from the class Circle so that it takes in x, y, and z coordinates (in that exact order), and sets the center of the Sphere object accordingly. Your method implementation here must utilize the `setCenter` method already defined in the super class Circle.
- Your class should override the `getArea` method from class Circle to make it return the (surface) area of the Sphere 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 Sphere object (as a `double` value). Your method implementation here must utilize the getArea computation already defined in the super class Circle.

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 – the Gradescope autograder only exposes a **subset** of the test cases that we will try. Remember that your methods must have signatures (headers) that **match exactly the provided descriptions**, otherwise the autograder 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 Sphere class. To submit this assignment, you must upload four files on Gradescope: **Circle.java**, **Rectangle.java**, **ShapeTester.java**, and **Sphere.java**. Make sure all files compile and run successfully before you submit them.

Discussion and asking for clarifications: Use our course Piazza page to post questions about the assignment if you need a clarification about any of the methods. Please do **not** post solutions! Also, take advantage of our daily office hours to get your questions answered—and start early!

Grading: If a program does not compile, you will get 0 points for it. 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 Sphere 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 every 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_
*/
```

Submission Checklist: We've created this checklist to help you make sure you don't miss anything important. Note that completing all these items does not guarantee full points, but at least assures you are unlikely to get a zero.

- ☐ Did your four files (i.e., Circle.java, Rectangle.java, ShapeTester.java, Sphere.java) compile on the command line using JDK 11 or above (e.g. JDK 17)?
- ☐ Did your submission on Gradescope successfully compile and pass at least one autograder test case?
- ☐ Have you included the honor code on top of every file?
- ☐ Did you remove the `TODO` prefix from the methods you needed to implement?
- ☐ Did you give your variables meaningful names (i.e., no `foo` or `bar` variables)?
- ☐ Did you add meaningful comments to the code when appropriate?