CS5004-5005-SV – Lab 7 – June 18, 2020
Abstract Classes, Polymorphism, and Packages

Last week in Lab we practiced using inheritance with a hierarchy of bird classes. The main goal of tonight's lab is to practice using similar concepts but where the parent class is abstract, meaning that not every method is defined. One reason for using abstract classes is when ideas are conceptually related but calculated in different ways. For example, we think of Rectangles and Circles as both being Shapes, whose areas can be measured. However, the method for computing the area is different for each. At the same time, other methods, such as how to compare areas, might be the same; so, we could eliminate duplicate code by capturing it in the more abstract parent class. For methods that are the same for different kinds of shapes, ***inheritance*** saves work. For methods that differ, depending on which *kind* of shape, ***polymorphism*** saves the day.

0. Create a folder for your Shapes package, called myShapes. Initially, create three .java files within that folder: Shapes.java, Rectangle.java, and Circle.java. Depending on your IDE, your myShapes folder might need to be within another folder; for example, IntelliJ prefers a folder called src. The first line of each file should read: `package myShapes;`

   (When you go to build and run, you may need to tinker with your IDE config a bit to help it find the other files and the main method, in Class Rectangle.)

1. In Shapes.java, enter this starter code:

```
public abstract class Shape {
    abstract double area();
    abstract double perimeter();
    public boolean compareTo(Shape other) {
        return true; // replace by logic
    }
}
```

   Notice that one of the methods is "abstract," meaning it must be defined in its subclasses. The other is "concrete" meaning that the subclasses can share the same method without duplicating code. In "top down" style, replace "true" by logic that returns true iff *this* Shape is larger in area than the *other* Shape, even though the area method is not yet defined.

2. In your Rectangle class, get started like this:

   ```
   public class Rectangle extends Shape {
   }
   ```

   Create private double instance variables for width and length. Create accessors and mutators for each. Create a constructor that accepts width and length parameters, a default constructor that creates a new 0x0 rectangle, and a copy constructor that duplicates an existing Rectangle.

3. Create a method, **area**, that returns a double calculated as the length times the width of the Rectangle. Be sure to use @Override in front of it, since you are overriding the abstract definition in the Shapes class.

4. Create an analogous method for the **perimeter**.

5. Begin a main method for testing your package within the Rectangle class. Initially, you should create a few Rectangles to test your constructors, getters, and setters, and print their areas and perimeters. Try using the parent **compareTo** method to compare areas. Notice that you can create new Rectangles and assign them to variables of either type Rectangle or type Shape.

6. Create your Circle class in the same way. It does not have length or width, but it does have an instance variable (double) for the radius. Create methods for the area and the perimeter (circumference), overriding the abstract methods from the abstract Shape class. Add tests to your main method for Circles, including comparing the areas of Rectangles and Circles.

7. Add a Triangle class. Instance variables should be the lengths of its 3 sides. Hint for the **area** method: https://www.mathopenref.com/heronsformula.html

8. Add **toString** methods for each Shape and test.

Submit your answers to above items to Canvas for CS5005 *(due next Thursday by midnight). Only your .java files for the above need to be uploaded. Prompt submission encouraged, but late is better than never.*

Work on any unfinished labs or CS5004 homework assignments. If finished, ***submit homeworks to Canvas CS5004***.