CS5004-5005-SV – Lab 8 – June 25, 2020
Abstraction Part 2

Last week in Lab we practiced working with abstract classes (accidentally slightly ahead of the lecture on this topic). Since then, you have heard more about **abstract classes**, Java's features for default methods, combining ***inheritance*** from abstract classes with ***implementing interfaces***, and so on. This lab provides more practice with these constructs to support abstraction in object-oriented design/programming.

0. Make a copy of your Shapes package from the previous Lab. It should contain the abstract class Shape.java and two concrete subclasses called Rectangle.java and Circle.java, hopefully with implementations of **area** and **perimeter** methods to override the abstract methods from Shape itself. Add a **toPrint** method, if not already done.

1. If you hadn't already done so, add a **Triangle** (aka Tri.java) class. Instance variables and constructor parameters should be the lengths of its 3 sides. Hint for the **area** method: https://www.mathopenref.com/heronsformula.html. Triangle's **perimeter** method should be easy in this case.

2. Add **toString** methods for each Shape, if not already done, and test.

3. Add 2 sub-types of Triangles: **RightTri** and **EquilateralTri**. Implement all appropriate methods. Naturally, try to avoid redundant code. Consider whether a **default(?) method** might help anywhere. (Hint: Heron's formula?)

   Add methods to tell you if a Triangle is a Right Triangle or an Equilateral.

4. **(a)** Add the ability to compare Shapes, based on their area. Use the **Comparable** interface (as covered in lecture).
   (b) Also provide a method to compare shapes based on their perimeter. Use the **Ordered** interface (also covered in lecture; .java file provided).

   The intent is to increase understanding of these three different approaches to comparisons, while gaining practice with implementing interfaces.

5. Find at least 5 bugs in the file, Buggy.java. You can simply add comment lines to the .java file pointing out flaws. Upload an annotated version as your solution to this part. You do *not* have to actually *fix* the bugs. (Given such extremely sloppy coding, it would probably be easier to just start over!)

6. [OPTIONAL BONUS QUESTION] Add a new optional parameter (in other words overload the constructor) for each shape, to add an int color, representing a choice from a set of colors (eg 1 = red, 2 = blue, etc.). Create an interface called **colorfulThing**. It has just one required method, it needs: **int getColor method.** Shape should implement it. It's job is to return the color instance variable of each shape. Now, we also want to be able to compare items based on their color (so in this case red < blue). We have already used Comparable (compareTo method), so we can't use that again. We have already used Ordered. So next look at the Comparator interface. So, for each of the three comparisons we are doing, we are using three different approaches to doing it, just for practice. See the following URL for how to use Comparator: https://www.geeksforgeeks.org/comparable-vs-comparator-in-java/

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. Add comments showing which code addresses each part above. 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***.