Inheritance and Visibility: Shapes
This assignment is to help you practice with:
Inheritance
Types of visibility (public, private, package/default)
Use of final for
Methods, classes, parameters, and instance fields
Implementing interfaces
Writing a factory class for building objects that are related
Checking parameters and throwing exceptions when the data in the parameters does not look right
Using CheckStyle, PMD, and SpotBugs
More practice with Javadoc comments
For this assignment you will build a simple inheritance hierarchy of shapes. The hierarchy is as follows:
A Shape class (you can rename it to AbstractShape to get rid of style error) that has the following items:
public abstract double area(): will return the area of a given shape – this method will have no code in it (it will be abstract)
public abstract String name(): will return the shape name – this method will have no code in it (it will be abstract)
NOTE: You can remove the abstract designation if you wish and implement this method and call getClass().getSimpleName() to get the name of the class.
public String toString(): will return a String containing the name of the shape along with its area as follows:

Name: <shape name goes here>, Area: <area of shape goes here>

implements the Comparable interface and provides an implementation for the public int compareTo(final Shape theOtherShape) behavior/method. This method should compare the current shape to the one passed in to the method. It must first compare by name, then by area if the names are the same. Be

sure and use generics to specify you are comparing against another Shape when you implement this interface.
overrides the equals method inherited from Object (public boolean equals(final Object theOtherObject))
it should use getClass() to see if current instance is the same class as otherObject --do not use instanceof
NOTE: it is ok (good form, actually) for equals to call the compareTo method you wrote :-)
All shape classes listed below must extend this Shape class
All shape classes listed below must provide an override for the public int hashCode() method inherited from Object
Circle class that has the following:
extends the Shape class (and thus must provide implementations of the area() and name() methods)
An instance field for the radius that is final
A package level constructor that is passed a radius
The constructor must make sure this value is greater than 0 or throw an IllegalArgumentException with specific information about what was wrong with the value (e.g. it was not greater than 0 and show the actual value)
A public get method to return the radius
The class must be final
Square class that has the following:
extends the Shape class (and thus must provide implementations of the area() and name() methods)
An instance field for the side length that is final
A package level constructor that is passed a side length
The constructor must make sure this value is greater than 0 or throw an IllegalArgumentException with specific information about what was wrong with the value (e.g. it was not greater than 0 and show the actual value)
A public get method to return the side length
The class must be final
Triangle class that has the following:

extends the Shape class (and thus must provide implementations of the area() and name() methods)
An instance fields for the base and height that are final
A package level constructor that is passed a base and height
The constructor must make sure these values are greater than 0 or throw an IllegalArgumentException with specific information about what was wrong with the value (e.g. it was not greater than 0 and show the actual value)
public get methods to return the base and height, respectively
The class must be final
Rectangle class that has the following:
extends the Shape class (and thus must provide implementations of the area() and name() methods)
An instance fields for the length and width that are final
A package level constructor that is passed a length and width
The constructor must make sure these values are greater than 0 or throw an IllegalArgumentException with specific information about what was wrong with the value (e.g. it was not greater than 0 and show the actual value)
public get methods to return the length and width, respectively
The class must be final
ShapeFactory class that has the following:
Static methods to create Circle, Square, Triangle, and Rectangle
Those static methods should accept parameters necessary to call the constructors for each of the classes
ALL THE ABOVE ITEMS MUST BE IN A PACKAGE NAMED shapes
ShapeTester class that has the following:
Is placed in a package called controller
Creates at least 5 shapes of each type (5 circles, 5 squares, 5 triangles, 5 rectangles, for a total of 20 shapes) using the ShapeFactory and places them in an ArrayList of type Shape
The 5 shapes of each type (5 circles, 5 squares, 5 triangles, 5 rectangles) should have at least two of those shapes with the same data (e.g. two Circles with the same radius, etc.)

Prints the ArrayList of shapes, preceded by a message specifying that the shapes are being printed before sorting
Make sure what you print is nicely formatted and easy to read
Sorts the ArrayList of shapes
You can use a sort method from the Java API or write one of your own
If you write your own sort method you may only use the compareTo method provided by the shapes
Prints the ArrayList of sorted shapes, preceded by a message specifying the shapes have been sorted
Make sure what you print is nicely formatted and easy to read
Additional Details/Requirements
All classes and interfaces above must conform to our class documentation (Javadoc) and naming standards.
Any method in a class that is an override of a method from a super class or interface MUST have the @OverRide tag above the method declaration
Any method in a class that should not be overridden in a child/sub class MUST be declared final
Any class that should not be inherited from MUST be declared final
Run CheckStyle, PMD, and SpotBugs and clear as many warnings as possible. If you are able to clear all of them, you can earn an extra 4 points on your assignment. However, you MUST document this in your executive summary. Also include screen shots that show you had no errors.
NOTE: You can ignore indentation complaints when you know your indents are four spaces
NOTE 2: You can ignore the complaint about the use of System.out.println instead of writing to a logger (log file)
NOTE 3: When CheckStyle and PMD contradict one another, follow CheckStyle
Capture the output from running your program and place it in a pdf file called ShapeTesterOutput.pdf. You can do a screen capture, a text capture, or whatever else you would like to get the output.
Include an executive summary as you did on Assignment 1

It should be named executive-summary.txt
See the specifications from assignment 1 for what to place in your summary
Include any CheckStyle or PMD warnings you were unable to resolve in this summary
Create a folder named with your UWNetIDassignment2 (e.g. tcapaulassignment2) and place the following items in that folder
The folder/directory that contains your IntelliJ project
Your output capture (ShapeTesterOutput.pdf)
Your executive summary (executive_summary.txt)
If you earned extra credit, be sure and include screen shots to show you have no errors/warnings from CheckStyle, PMD, and SpotBugs. Name this file ExtraCreditScreenShots.pdf
Zip the folder and submit it to Canvas
HINTS
Focus on one thing at a time
Get your Shape class written first, then maybe write your Circle class and test to see the basics of Shape and Circle work. Don't even worry about implementing the Comparable interface for Shape right away – do that after the other things in Shape and then Circle are confirmed to work
Once Circle works, the other shapes will be very similar to design and implement
Once everything in the Shape hierarchy is in place, write your ShapeFactory class. It will be comprised of static methods that will take the necessary parameters to call the constructors of the given shapes. As an example, you'll probably want a method that looks like this:

```
public static Circle createCircle(final double theRadius) {
//code goes here to create a Circle with the specified radius
  // and return it
}
```

Finally, you can create your ShapeTester and provide the items specified for that class