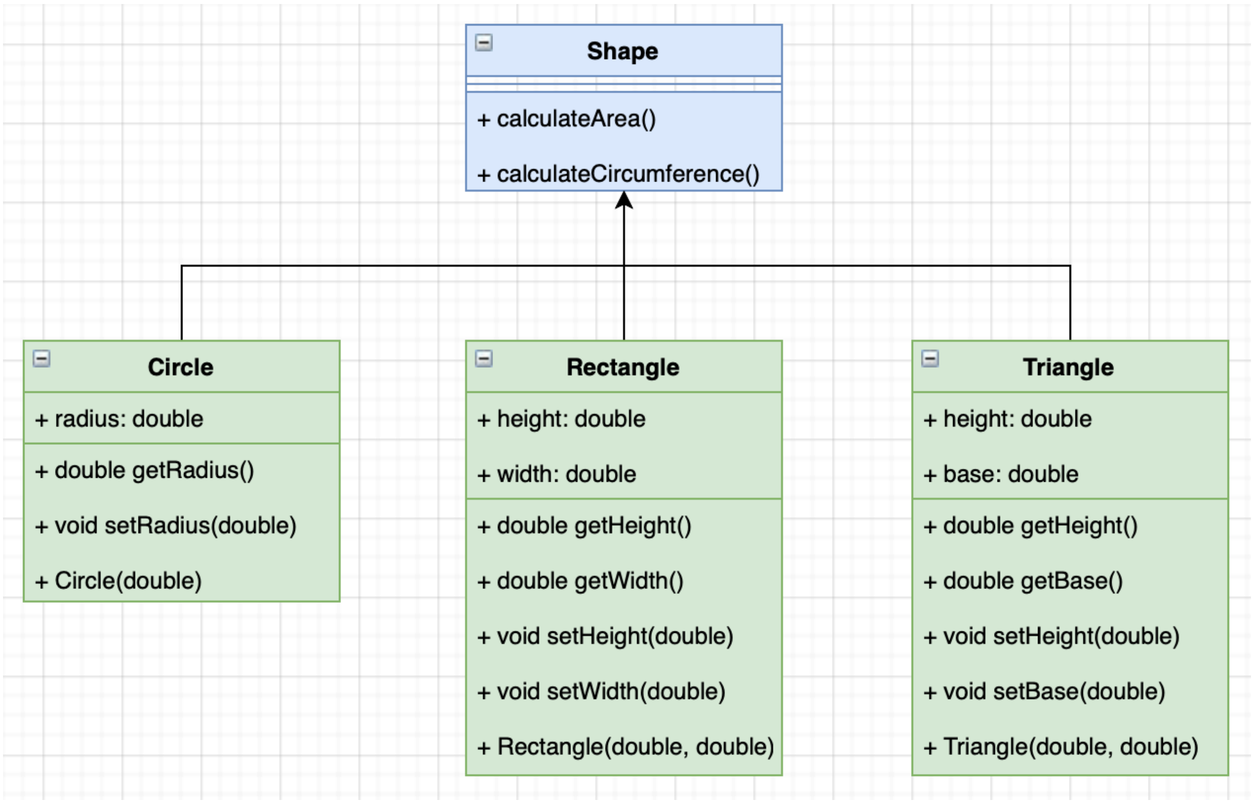
Lab 4

For the LAB

You will be completing code for a Shape Abstract Class and 3 subclasses, Triangle, Rectangle, and Circle. Shape defines two int variables: x and y, which represent the coordinates of the center of the shape. Shape also defines 2 abstract methods calculateArea() and calculateCircumference(). Both return doubles representing the area and circumference of the shapes respectively.

For each shape you will have to create the necessary values need to calculate each of these values and getters and setters for each of the values. You will also need to create constructors for each class, and the instructions for those will be included with each class.

Provided below is a UML diagram that gives a visual representation of this lab



Part 1 Shape: calculateArea() and calculateCircumference()

In the Shape Class you will be writing two abstract methods calculateArea() and calculateCircumference(). Both of these methods will take no parameters and will return a double. Since these methods are abstract, the other classes will define these methods for their specific class.

Part 2 Circle Class

For this part, you will be writing the Circle class. This class will have a constructor, accessors and mutators, and the abstract methods defined for this class.

Circle Class: Constructor, Circle(double radius)

The Circle class constructor will take in a double that represents a radius of a circle. This value should be stored in an instance variable that is private, since we will use mutators and accessors to manipulate and obtain the value of it.

Circle Class: Accessors and Mutators, getRadius()

setRadius(double radius)

With the radius instance variable made, accessors and mutators should be made for that variable.

The accessor will return the value of the radius instance variable, and should be named *getRadius()*.

The mutator should take in a double that can change the radius for a given Circle object, and should be named *setRadius()*.

Circle Class: Abstract Methods, calculateArea() and calculateCircumference()

Since this class ***extends*** the Shape class, this means that the Circle class must implement the abstract methods in the Shape class. In this case, those methods are calculateArea() and calculateCircumference(). For each method, return the area and circumference respectively of a circle with the radius specified by the object.

Testing Circle Class

In the Test Class, you can now test the code written in this class. Make sure to see if the accessors and mutators are returning correctly. Also, make sure the area and circumference are being returned correctly. For area and circumference, you may assume the results will have 2 decimal points of precision.

Part 2 Rectangle Class

For this part, you will be writing the Rectangle class. This class, like the previous, will have a constructor, accessors and mutators, and abstract methods defined for this class.

Rectangle Class: Constructor, Rectangle(double height, double width)

The Rectangle class constructor will take in two doubles, one for height and the other for width. These values should be stored in instance variables.

Rectangle Class: Accessors and Mutators, getHeight() getWidth() setHeight(double height) setWidth(double width)

With the instance variables for both height and width made, accessors and mutators for each of the variables should be made. For the accessors, the method should return the value for the appropriate instance variable. For mutators, each mutator will take in a double to change the instance variables. This means there is one accessor and mutator for each instance variable.

Rectangle Class: Abstract Methods, calculateArea() and calculateCircumference()

Like in the Circle class, the Rectangle class also extends the Shape class meaning the abstract methods in the Shape class must be defined in the Rectangle class. Again, these methods are calculateArea() and calculateCircumference(). It may be obvious to you that rectangles do not have circumferences but perimeters. For the calculateCircumference() method, return the **perimeter** of a given rectangle object. Also, return the area of a given rectangle in the calculateArea() method.

Testing Rectangle Class

In the Test Class, you can now test the code written in this class. Make sure to see if the accessors and mutators are returning correctly. Also, make sure the area and circumference are being returned correctly. For area and circumference, you may assume the results will have 2 decimal points of precision.

Part 3 Triangle Class
For this part you will be writing the Triangle class. As in the previous class, the Triangle Class has a constructor, accessors and mutators and abstract methods defined.

Triangle Class: Constructor, Triangle(double height, double base)
The constructor for the Triangle Class will take two doubles, one for base and one for height. These values should be stored in instance variables.

Triangle Class: Accessors and Mutators, getHeight() getBase() setHeight(double height) setBase(double base)
Like in the Rectangle class, there are now two instance variables, one for height and another for base. Both of these instance variables need to have accessors and mutators. The accessors should return the respective value of an instance variable. The mutators will take a double to change a respective instance variable.

Triangle Class: Abstract Methods, calculateArea() and calculateCircumference()

As seen previously in all the other subclasses, the Triangle class also extends the Shape class. This means that the abstract methods in the Shape class have to be defined in the Triangle class.

The Triangle class will be limited to performing calculations for **equilateral** triangles, where all sides are the same length. This should make calculations for area and perimeter relatively simple.

The methods that must be implemented are calculateArea() and calculateCircumference(). For calculateArea(), return the area of a triangle, which as a reminder is (base*height)/2. As before with the Rectangle Class, there is not a circumference for a triangle. Therefore, for this method return the perimeter of a triangle. As you have the base of the triangle, and because this class is limited to just equilateral triangles, you should be able to easily calculate the perimeter of the triangle..

Testing Triangle Class

In the Test Class, you can now test the code written in this class. Make sure to see if the accessors and mutators are returning correctly. Also, make sure the area and circumference are being returned correctly.