Objective: The purpose of this activity is to explore a Shape object that serves as the parent for at least 3 other shapes (i.e. classes that extend Shape or inherit visible members) in order to intentionally incorporate inheritance and polymorphism into a program.
Submission: *Shape.java, Rectangle.java, Triangle.java, Circle.java, ShapeTest.java*

The third pillar of object-oriented programming is polymorphism. This is the idea that given derived classes (created through the use of inheritance), we can use a reference variable of a superType to hold a reference to a subType object.

For this assignment, you will create at least 5 classes: Shape.java, Circle.java, Triangle.java, Rectangle, and ShapeTest.java. Shape will serve as the SuperClass, from which Circle, Triangle and Rectangle will be derived. ShapeTest.java will hold the main method used to create instances of these objects and test the code that you create.

`public class Shape{}:` This class represents a superclass, and includes only a few members (described below).

- Private Data Field(s):
    - `String color;`
    - `String name`;
    - `Static int numShapes=0;` //to hold the number of shape objects created
- Methods
    - Constructor(s): overload the constructor method at least 2 times to initialize any combination of data fields.
    - Getter/Setter methods: create methods to access and modify private data fields
    - Create `computeArea()` and `computePerimeter()` methods that simply return 0.0.
        - These methods will be overridden in the subclasses that you will define
        - *Think*: why do we have to define these methods in `Shape`, to only return 0?
    - Overridden methods: override `toString` (from the `Object` class) to return the shape name and color.

`public class Circle{}:` This is a direct child/subclass to `Shape`, and is described here.

- Private Data Field(s):
    - `double radius;`
    - `static int numCircles=0;` //to hold the number of circle objects created
- Methods
    - Constructor(s): overload the constructor method <u>at least 3 times</u> to initialize any combination of data fields from Circle and within the super class.
    - Getter/Setter methods: create methods to access and modify `radius`
    - Override `computeArea()` to calculate the area of the circle and return the value as a double
        - As a reminder, the formula to calculate the area of a circle is: $A=\pi*(radius^2)$
    - Override `computePerimeter()` to calculate the perimeter of the circle and return the value as a double
        - As a reminder, the formula to calculate the perimeter of a circle is: $P=2*\pi*radius$
    - Override `toString` (from the `Shape` class) to return super.toString() concatenated with the radius and area of the shape.

`public class Triangle{}:` This is a direct child/subclass to `Shape`, and is described here.

- Private Data Field(s):
    - `double base;`
    - `double height;`
    - `static int numTris=0;` //to hold the number of triangle objects created
- Methods
    - Constructor(s): overload the constructor method <u>at least 3 times</u> to initialize any combination of data fields from Triangle and within the super class.
    - Getter/Setter methods: create methods to access and modify the private data fields
    - Override `computeArea()` to calculate the area of the triangle and return the value as a double
        - As a reminder, the formula to calculate the area of a triangle is: $A=\frac{1}{2} *base *height$
    - Override `computePerimeter()` to calculate the perimeter of the triangle and return the value as a double

        - As a reminder, the formula to calculate the perimeter of a triangle is: P=x+y+base, where x and y are the length of the other two sides for the triangle
        - You can ask the user for input here for the length of the other two sides (since you already have the base), or you can incorporate those values as private data fields to be initialized within the constructor(s).
        - Be sure to validate these as positive doubles
    - Override `toString` (from the `Shape` class) to return super.toString() concatenated with the base, height and area of the shape.

`public class Rectangle{}:` This is a direct child/subclass to `Shape`, and is described here.

- Private Data Field(s):
    - `double length;`
    - `double width;`
    - `static int numRects=0;` //to hold the number of rectangle objects created
- Methods
    - Constructor(s): overload the constructor method <u>at least 3 times</u> to initialize any combination of data fields from Rectangle and within the super class.
    - Getter/Setter methods: create methods to access and modify the private data fields
    - Override `computeArea()` to calculate the area of the triangle and return the value as a double
        - As a reminder, the formula to calculate the area of a rectangle is: A=length * width
    - Override `computePerimeter()` to calculate the perimeter of the triangle and return the value as a double
        - As a reminder, the formula to calculate the perimeter of a triangle is: P=(2*length) + (2*width)
    - Override `toString` (from the `Shape` class) to return super.toString() concatenated with the base, height and area of the shape.

`public class ShapeTest{}:` This class will include a main method to test the Shape, Circle and Triangle classes that you previously created. The main method you create should be simple enough to test your code, but also highlight the power of polymorphism, afforded through the use of object-oriented programming and inheritance. Your program should include a modularized main class (using methods) to do the following:

- An array of shapes that includes references to all types of shape objects
- Allows the user to manually create a variety in the array
  - Be sure to validate any input, allowing the user to indicate what shape they are creating
  - Only one scanner should be used per program
- Randomly create shapes in the array (and pass in random values to constructors)
- Display each object, including the perimeter and area
- Display the total perimeter of all objects
- Display the total area of all objects
- Display the total number of each type of object created

Suggestion: throughout this class (and the others that you create), add comments explaining the various concepts of OOP that you are utilizing.