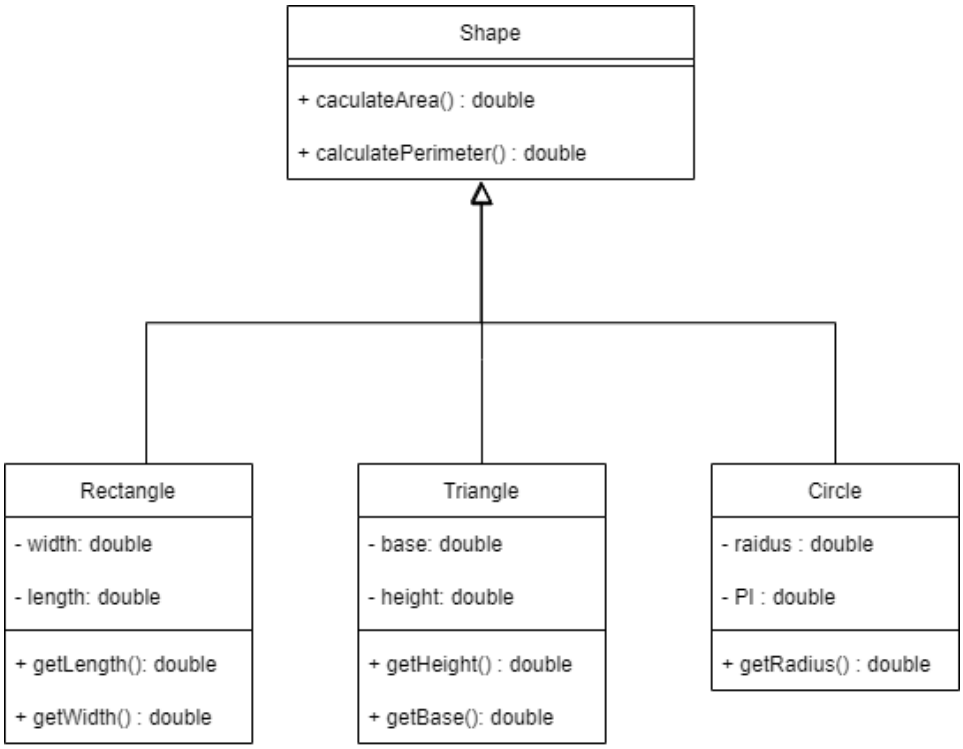
Refactoring Exercise

Given the following Codebase link perform the following

1. Create a class diagram that illustrates the given implementation
2. Identify any design flaws/code smells/SOLID principle violations in the implementation giving an argument why.
3. Fork and refactor the codebase to address the violation identified in the previous question

Solution

Question 1



Question 2

Inappropriate Abstraction

Shape is defined as a concrete class but judging by its implementation, the user would never need to create an instance of shape. Additionally, it provides no useful concrete methods that are used as-is so there is little reason for it to be an abstract class either.

Shape.java

```
1 public class Shape{
2   double calculateArea( ){
3     return 0.0;
4   };
5   double calculatePerimeter( ){
6     return 0.0;
7   };
8 }
```

Redundant Code

The intent of the user is to test the methods by printing dimensions, perimeter and area of the shapes. However, it is done in a redundant manner. To handle additional shapes a similar code block must be copied and pasted.

```
public static void ShapeTest(Rectangle rectangle, Circle circle, Triangle triangle){
    System.out.println("Rectangle width: " + rectangle.getWidth( ) + " and length: " + rectangle.getLength( )
            + "\nResulting area: " + rectangle.calculateArea( )
            + "\nResulting perimeter: " + rectangle.calculatePerimeter( ) + "\n");

    System.out.println("Triangle. base:" + triangle.getBase( ) + " and height:" + triangle.getHeight( ) +
      "\nResulting Area: " + triangle.calculateArea( )
      + "\nResulting Perimeter: " + triangle.calculatePerimeter( ) + "\n");

    System.out.println("Circle radius: " + circle.getRadius( )
      + "\nResulting Area: " + circle.calculateArea( )
      + "\nResulting Perimeter: " + circle.calculatePerimeter( ) + "\n");
}
```

Unnecessary Specificity

The ShapeTest method works with exactly 3 child shapes, rectangle, circle and triangle but does not invoke any behavior specific to the children.

For each child object the method prints 3 things:

1. Dimensions
2. Area
3. Perimeter

These are all behaviors that can be exhibited by any shape and thus should be invoked via the shape type. ShapeTest doesn't care that the objects are rectangle\circle\triangle because it only invokes general Shape behavior.

Open Closed Principle Violation

ShapeTest is coupled to specific shape child classes. If we were to add new shape child classes; ShapeTest would need to be updated to accommodate a new parameter as an instance of the new child class.

Question 3

Complete the following tasks:

- Convert shape to an interface
- Add a toString() method to the interface
- Implement toString() in the children classes return the type of shape and its dimensions as a string eg "Circle radius: 10" is returned by Circle.toString()
- Update child classes to implement the new interface instead of extend
- Update ShapeTest to work with a collection of shapes and invoke all methods of Shape

Main.java should then look as follows:

```java
class Main {

  public static void ShapeTest(Shape[] shapes){
    for (Shape shape: shapes){
          System.out.println(shape.toString()
          + "\nResulting area: " + shape.calculateArea()
          + "\nResulting perimeter: " + shape.calculatePerimeter() + "\n");
    }
  }

  public static void main(String[] args) {
    Rectangle rectangle = new Rectangle(5, 10);
    Circle circle = new Circle(7);
    Triangle triangle = new Triangle(4, 6);
    Shape[] shapes = new Shape[]{rectangle, circle, triangle};
    ShapeTest(shapes);

  }
}
```

Now ShapeTest is closed for modification for the creation of new shape children. Additionally, it can work with any amount of shapes.

https://replit.com/@Snickdx/Shape-Exercise-refactor

References

- UML Conventions