Data Structures and Algorithms

Lab 4: Object-Oriented Programming - Part 3

I. Objective

After completing this tutorial, you can:

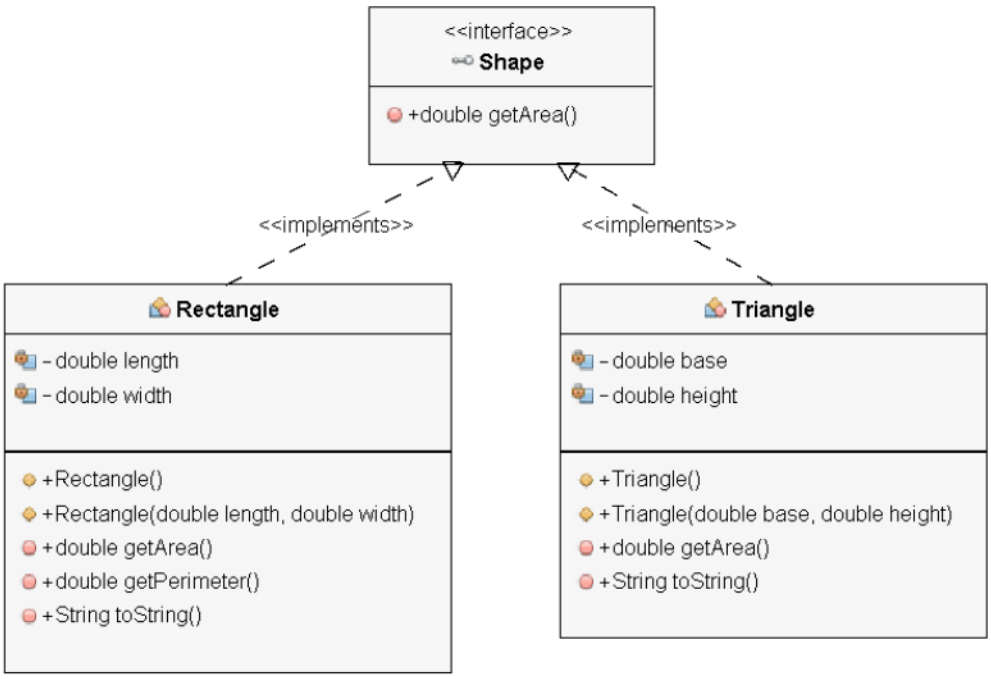
- Understand *polymorphism* and *abstraction* in OOP.

II. Polymorphism

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

In the following example, we have two real objects, which are *Rectangle* and *Triangle*, and a general object, *Shape*. In reality, we don't need to create a *Shape* object, since it does not have any behavior. The question is how can we prevent users from creating *Shape* object.

We have two approaches, the first one is taking advantage of *interface* and *abstract* class.



Shape.java

```
public interface Shape {

    public double getArea();

}
```

Rectangle.java

```
public class Rectangle implements Shape{

    private double length;
    private double width;

    public Rectangle()
    {
        this.length = 0;
        this.width = 0;
    }

    public Rectangle(double length, double width)
    {
        this.length = length;
        this.width = width;
    }

    @Override
    public double getArea()
    {
        return this.length * this.width;
    }

    public double getPerimeter()
    {
        return (this.length + this.width)*2.0;
    }

    @Override
    public String toString()
    {
        return "Rectangle{" + "length=" + length + ", width=" + width + '}';
    }
}
```

Test.java

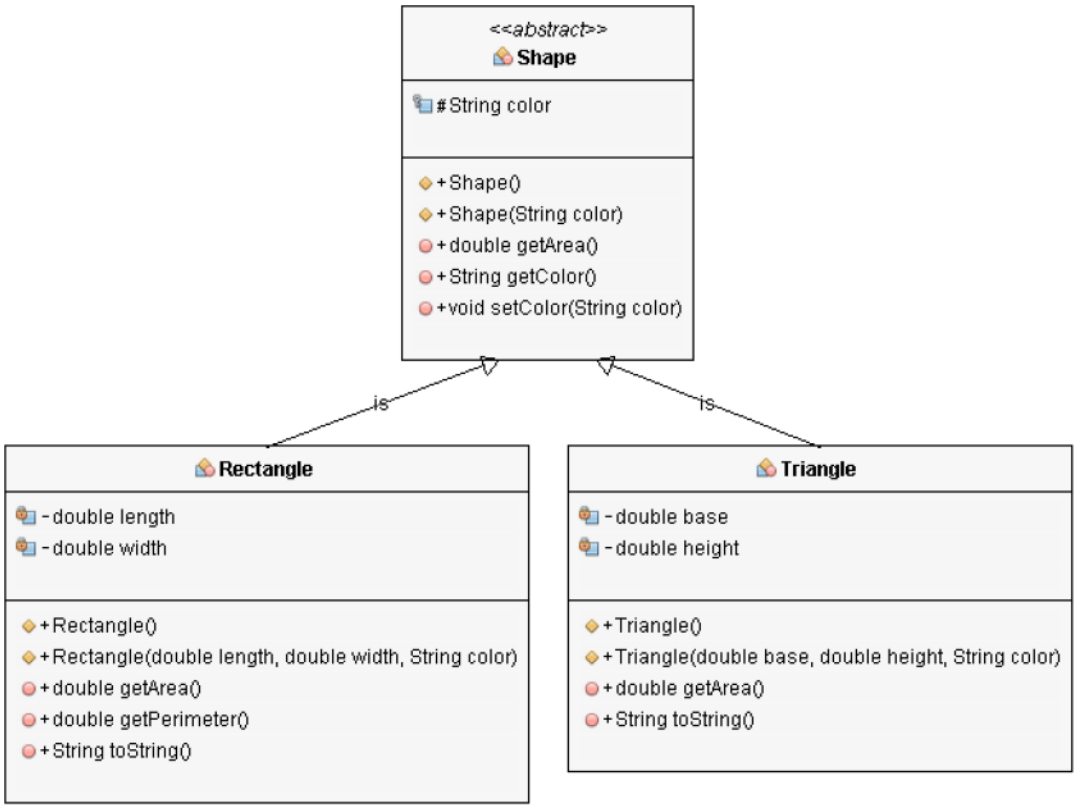
```
public class Test {

    public static void main(String[] args)
    {
        Shape s = new Rectangle(3, 4);
        System.out.println(s.toString());
        System.out.println("Area = " + s.getArea());

        s = new Triangle(4, 5);
        System.out.println(s.toString());
        System.out.println("Area = " + s.getArea());
    }
}
```

III. Abstraction

Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. For example, in email system, to send an email, a user need only to provide recipient email, the content, and click send. All implementation of the system is hidden.

Java provides a mechanism to allow a program achieve abstraction, by using *abstract* keyword. The abstract keyword can be used for class and method definition. For example, in the following diagram, we'll define an abstract class, *Shape*, which contains *color* attribute and *getArea()* behavior. That means, every derived object from *Shape* will have *color* information and *getArea()* method.

You should notice that if you define an abstract class, the method must be either an abstract method or an implemented method.



Shape.java

```
public abstract class Shape {

    protected String color;

    public Shape()
    {
        this.color = "";
    }

    public Shape(String color)
    {
        this.color = color;
    }

    public abstract double getArea();

    public String getColor()
    {
        return color;
    }
```

```
    public void setColor(String color)
    {
        this.color = color;
    }

}
```

Rectangle.java

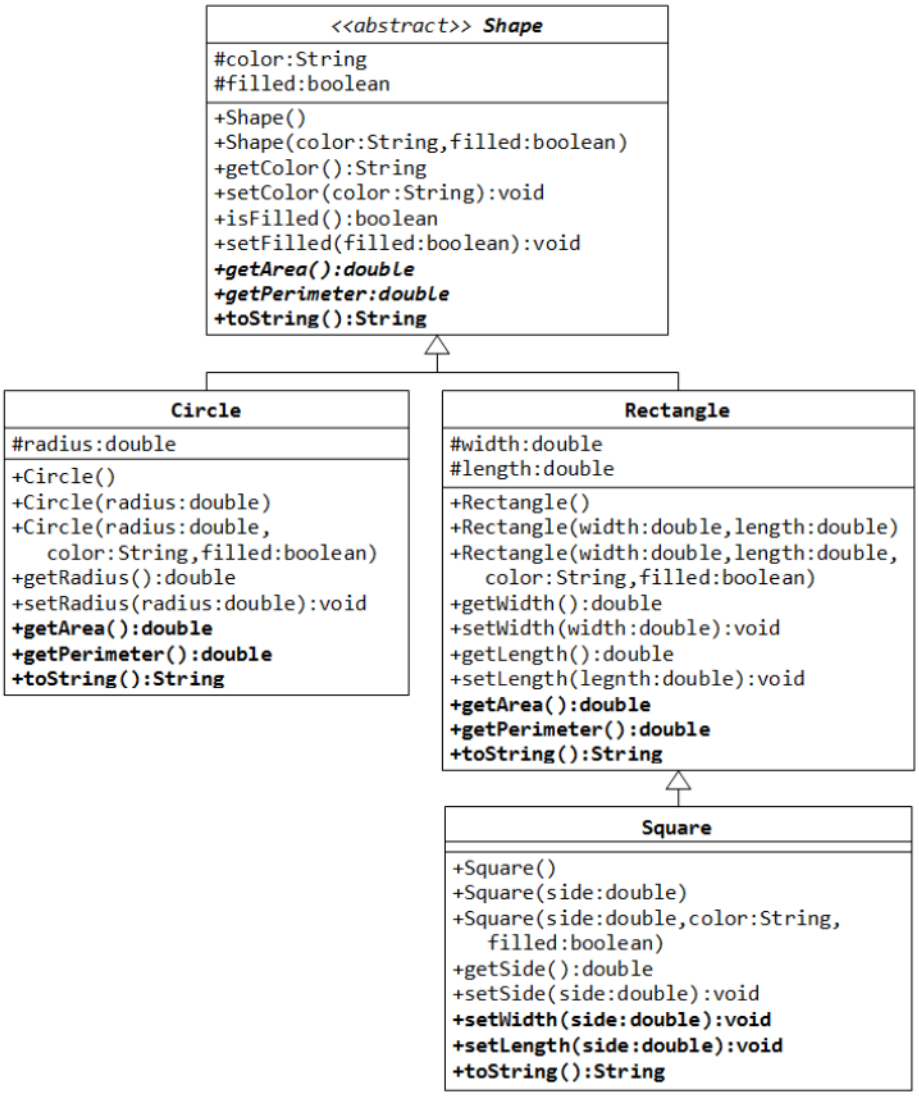
```
public class Rectangle extends Shape{

    private double length;
    private double width;

    public Rectangle()
    {
        super();
        this.length = 0;
        this.width = 0;
    }

    public Rectangle(double length, double width, String color)
    {
        super(color);
        this.length = length;
        this.width = width;
    }

    @Override
    public double getArea()
    {
        return this.length * this.width;
    }

    public double getPerimeter()
    {
        return (this.length + this.width)*2.0;
    }

    @Override
    public String toString()
    {
        return "Rectangle{" + "length=" + length +
                ", width=" + width +
                ", color=" + color + '}';
    }
}
```

Test.java

```
public class Test {

    public static void main(String[] args)
    {
        Shape s = new Rectangle(3, 4, "white");
        System.out.println(s.toString());
        System.out.println("Area = " + s.getArea());

        s = new Triangle(4, 5, "black");
        System.out.println(s.toString());
        System.out.println("Area = " + s.getArea());
    }
}
```

IV. Exercises

1. What is the difference between the two above diagrams?
2. Continue the above examples, implement the *Triangle* class.
3. Abstract superclass *Shape* and its concrete subclasses.

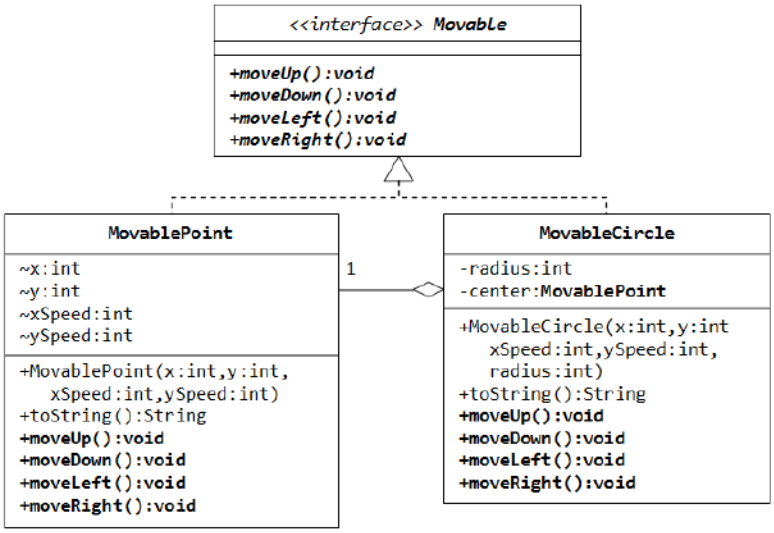


4. Examine the following program and draw the diagram.

```
abstract public class Animal {
   abstract public void greeting();
}

public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}

public class Dog extends Animal {
   @Override
   public void greeting() {
```

```
        System.out.println("Woof!");
    }

    public void greeting(Dog another) {
        System.out.println("Wooooooooof!");
    }
}

public class BigDog extends Dog {
    @Override
    public void greeting() {
        System.out.println("Woow!");
    }

    @Override
    public void greeting(Dog another) {
        System.out.println("Wooooowwwww!");
    }
}
```

5. Interface *Movable* and its implementations *MovablePoint* and *MovableCircle*.



6. (*) Interfaces *GeometricObject* and *Resizable*.