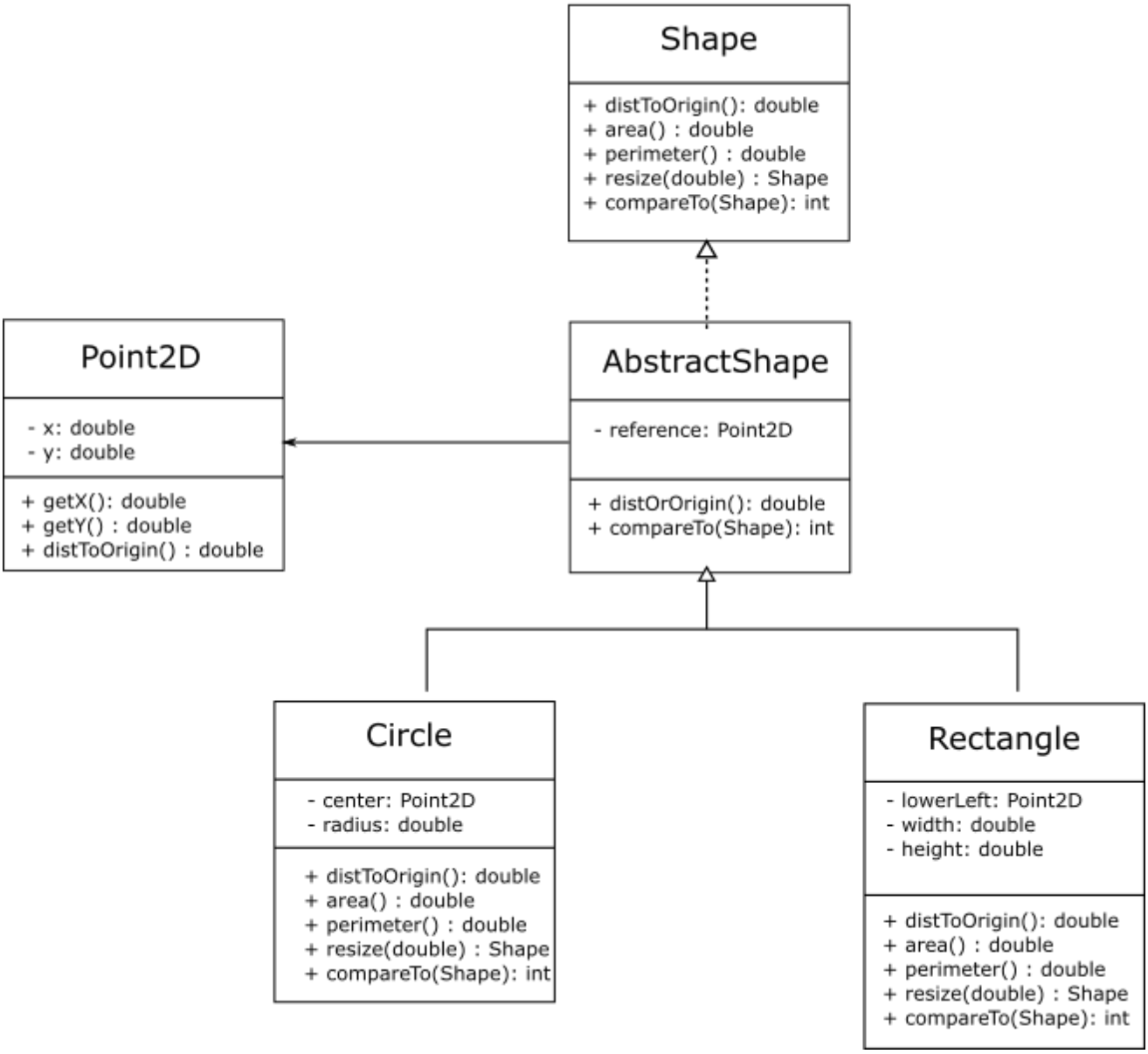
CS5010 P8

Situation 1



- Single Responsibility Principle: Each of the above classes does exactly one thing. For example, the Circle and the

Rectangle classes are only responsible for the properties and actions related to shape.

- Open/Close Principle: If there are more shapes to be implemented, we could create more subtypes of the `AbstractShape` class. However, We are not able to modify the `AbstractShape` class.
- Liskov Substitution Principle: The work of `AbstractShape` classes can be replaced by its children `Circle` and `Rectangle` without causing any problem.
- Interface Segregation Principle: The `Shape` interface typically defines only the methods relevant to all shapes.
- Dependency Inversion Principle: Instead of directly communicating with each other, the `Circle` class and the `Rectangle` class use the `Shape` interface for communication.

Situation 2

We have the following **parent** class:

```
1 public class Animal {
2   public void makeNoise() {
3     System.out.println("I am making noise");
4   }
5 }
```

And we also have the following **sub-classes**:

```
1 public class Dog extends Animal {
```

```
  @Override
  public void makeNoise() {
    System.out.println("bow wow");
  }
}

public class Cat extends Animal {
  @Override
  public void makeNoise() {
    System.out.println("meow meow");
  }
}

public class Fish extends Animal {
  @Override
  public void makeNoise() {
    throw new RuntimeException("I can't make noise");
  }
}
```

Q1) Do you think this implementation follows SOLID principles?

Except for violating the Liskov Substitution Principle, it follows the rest of the SOLID principles.

- SRP: Each class does one job related to itself
- OCP: Every above class is open to access and close to modification

- LSP: The `Animal` class could not be substituted for the `Fish` class, thus violating the LSP.
- ISP: Each of the subclasses overrides the `makeNoise()` method, so each subclass 'implements' a method that is only related to itself.
- DIP: Each subclass uses the parent class to communicate instead of directly having a new `makeNosie()` method inside.

Q2) If not, a) can you identify the principle(s) we are breaking and b) a way to improve the design?

(a) LSP: The `Animal` class could not be substituted for the `Fish` class, thus violating the LSP.

(b) Instead of directly throwing errors, still using `println()` method.

```
1 public class Fish extends Animal {
2     @Override
3     public void makeNoise() {
4         System.out.println("...I cannot make
  noise");
5     }
6 }
```

Situation 3

We have an **interface** named **Shape,** which has the methods `area()`, `perimeter()`, `resize()` and `distanceToOrigin()`. We have already implemented several **concrete classes** with this interface: **Circle, Rectangle, Ellipse,** and **Square.** The product manager says **we now need to include two new methods** named `setColor()` and `getColor()`.

Q1) How would you redesign your current implementation so that you a) meet the requirements and b) do not break any SOLID principles?

Create a new subclass like:

```
public class ShapeColor extends AbstactShape{
    private String color;

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

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

And, then make the `Circle` class and the `Rectangle` class inherit this `ShapeColor` class instead of `AbstractShape` class.