

# Solution to Problems(methods)

## Interfaces

**Definition:** In its most common form, an interface is a group of related methods with empty bodies. Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile. To implement these interfaces, you'd use the **implements** keyword in the class declaration.

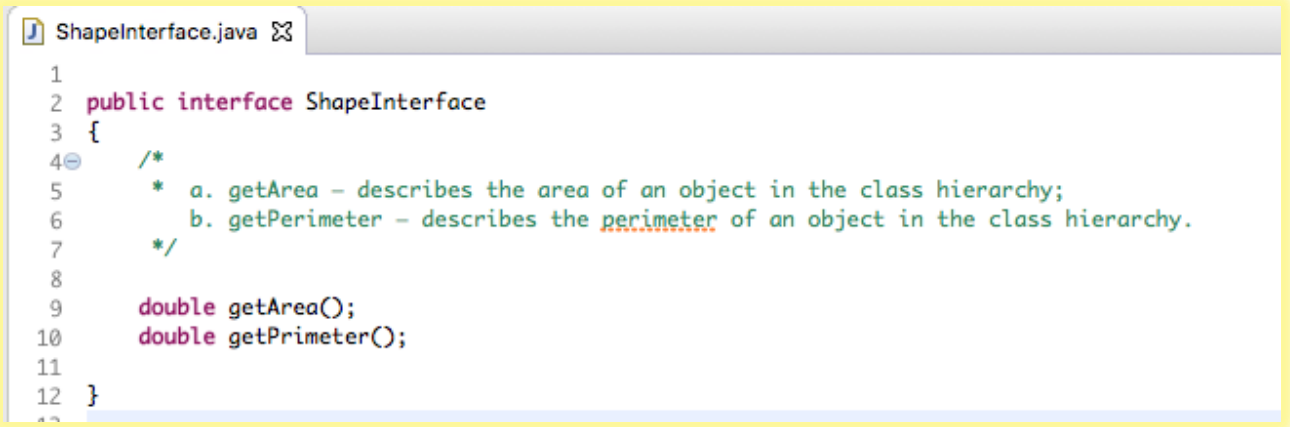
## My Interfaces

- [Interface ShapeInterface](#)
- [Interface PositionInterface](#)
- [Interface ShapePositionInterface](#)

## Interface ShapeInterface

Interface ShapeInterface includes appropriate abstract, static, and/or default methods that describe the intrinsic functions and behaviors of the specific object types of the class hierarchy, including:

- a. **getArea** — describes the area of an object in the class hierarchy;
- b. **getPerimeter** — describes the perimeter of an object in the class hierarchy.



```

1
2 public interface ShapeInterface
3 {
4     /*
5      * a. getArea - describes the area of an object in the class hierarchy;
6      * b. getPerimeter - describes the perimeter of an object in the class hierarchy.
7      */
8
9     double getArea();
10    double getPrimeter();
11
12 }

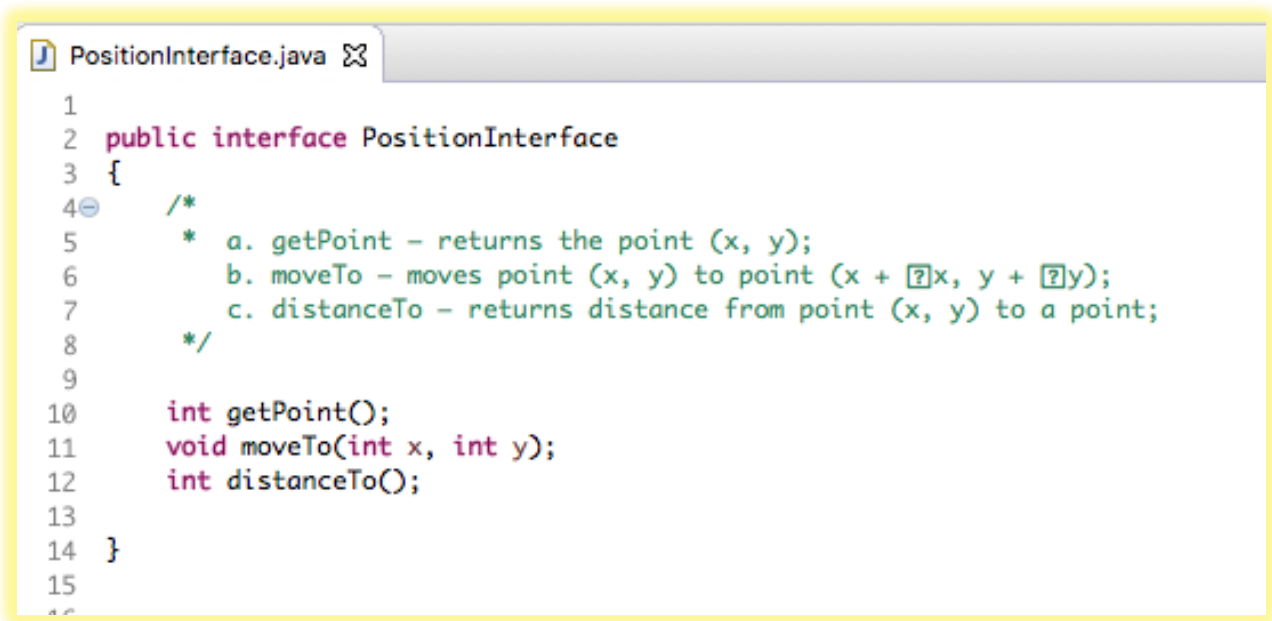
```

Figure 1 Code Developed

## Interface PositionInterface

Interface **PositionInterface** includes appropriate abstract, static, and/or default methods that describe the positional functions and behaviors of the specific object types of the class hierarchy, including:

- a. **getPoint** – returns the point (x, y);
- b. **moveTo** – moves point (x, y) to point (x +  $\Delta x$ , y +  $\Delta y$ );
- c. **distanceTo** – returns distance from point (x, y) to a point;



```
1
2 public interface PositionInterface
3 {
4     /*
5      * a. getPoint - returns the point (x, y);
6      * b. moveTo - moves point (x, y) to point (x + ?x, y + ?y);
7      * c. distanceTo - returns distance from point (x, y) to a point;
8      */
9
10    int getPoint();
11    void moveTo(int x, int y);
12    int distanceTo();
13
14 }
15
16
```

Figure 2 Code Developed

## Interface ShapePositionInterface

Interface ShapePositionInterface extends interface ShapeInterface and interface PositionInterface. Interface ShapePositionInterface includes appropriate abstract, static, and/or default methods that describe the functions and behaviors of the specific object types of the class hierarchy, including:

- a. **getBoundingBox** — returns the bounding rectangle of an object in the class hierarchy;
- b. **doOverlap** — returns true if two objects in the class hierarchy overlap.

```
*ShapePositionInterface.java
1
2 public interface ShapePositionInterface extends ShapeInterface, PositionInterface
3 {
4
5     /*
6      * a. getBoundingBox - returns the bounding rectangle of an object in the class hierarchy;
7      * b. doOverlap - returns true if two objects in the class hierarchy overlap.
8      */
9     void getBoundingBox();
10    boolean doOverlap();
11 }
12
```

Figure 3 Code Developed

## Solution to my Methods

### getArea() method for Rectangle class

To get the area of a rectangle is to multiply its length by its width. The formula is:

$A = L * W$  where A is the area, L is the length, W is the width, and \* means multiply.

In our program we are using the getArea() method that is implemented from the interface we have created. For that reason, when coding this, we must use the keyword, **override** and describe our method field.

```
@Override
public double getArea()
{ return height * width; }
```

Figure 2 Code Implementation

### getPerimeter() method for Rectangle class

To get the perimeter of a rectangle, simply add up all the sides of the rectangle or add the length and height of the rectangle and then multiply them by two. The formula is:  $P =$

$2*(L+W)$  where P is perimeter, L is length and W is width. When coding, we use the override annotation because we are using this method from the interface we have created.

```
@Override
public double getPerimeter()
{   return (2*(height + width));   }
```

*Figure 3 Code Implementation*

### **getArea()** method for **Oval** class

To get the area of an oval, we use the formula :  $A = r_1 * r_2 * \text{Pi}$ . where  $r_1$  is the larger radius and  $r_2$  is the smaller radius and we use Pi from the Java Math Library. When coding, we use the override annotation because we are using this method from the interface we have created.

```
@Override
public double getArea()
{   return (width/2)*(height/2)*Math.PI;   }
```

*Figure 4 Code Implementation*

### **getPerimeter()** method for **Oval** class

To get the perimeter of an oval, we use the formula:  $P = 2*\text{Pi} * (r_1^2 + r_2^2)/2$  where P is perimeter,  $r_1$  is the larger radius and  $r_2$  is the smaller radius, and we use Pi and sqrt from the Java Math Library. When coding, we use the override annotation because we are using this method from the interface we have created.

```
@Override
public double getPerimeter()
{ return 2*Math.PI*(Math.sqrt((width*width + height*height)/2)); }
```

*Figure 5 Code Implementation*

## moveTo() method

**moveTo()** method is declared in the interface **PositionInterface**. The purpose of this method is to take two parameters, dx and dy and move the object to another place in the screen. We can achieve this by adding dx to the current x value and dy to the current y value of the objects location.

```
@Override
public void moveTo(int dx, int dy)
{
    dx += getX();
    dy += getY();
}
```

*Figure 1 Code Implementation*

## draw() method

The purpose of the **draw()** method is to draw the shape using Graphics g as parameter. In the superclass, shape class has draw() method is declared as an abstract method and in each subclasses, this method is overridden by the **override** keyword.

```
public abstract void draw(Graphics g);
```

*Figure 6 Code Implementation*

### **getPoint() method**

the getPoint() method returns the points x and y. This method declared in the interface PositionInterface and implemented on the Shape class.

### **distanceTo() method**

in distanceTo() method we use the formula below to get the distance of a point from a coordinate(x,y). . This method declared in the interface PositionInterface and implemented on the Shape class.

### **getBoudingBox() method**

Here is how the boudingBox method is implemented. Since we have 4 points, x1, x2, y1 and y2, the method will return these points. An array is used in this method. This method may vary in different class.

### **doOverlap() method**

The function of doOverLap() method is to make sure if two shapes overlap each other. . This method declared in the interface PositionInterface and implemented on the Shape class.