# Guided LAB - 303.10.4
# How to use Java Interfaces

## Lab Overview

In the previous lab (GLAB - 303.10.3), we demonstrated that an abstract class has both methods with bodies, and methods with no bodies (abstract methods). You learned that abstract methods must be overridden in a subclass.

## Learning Objective:

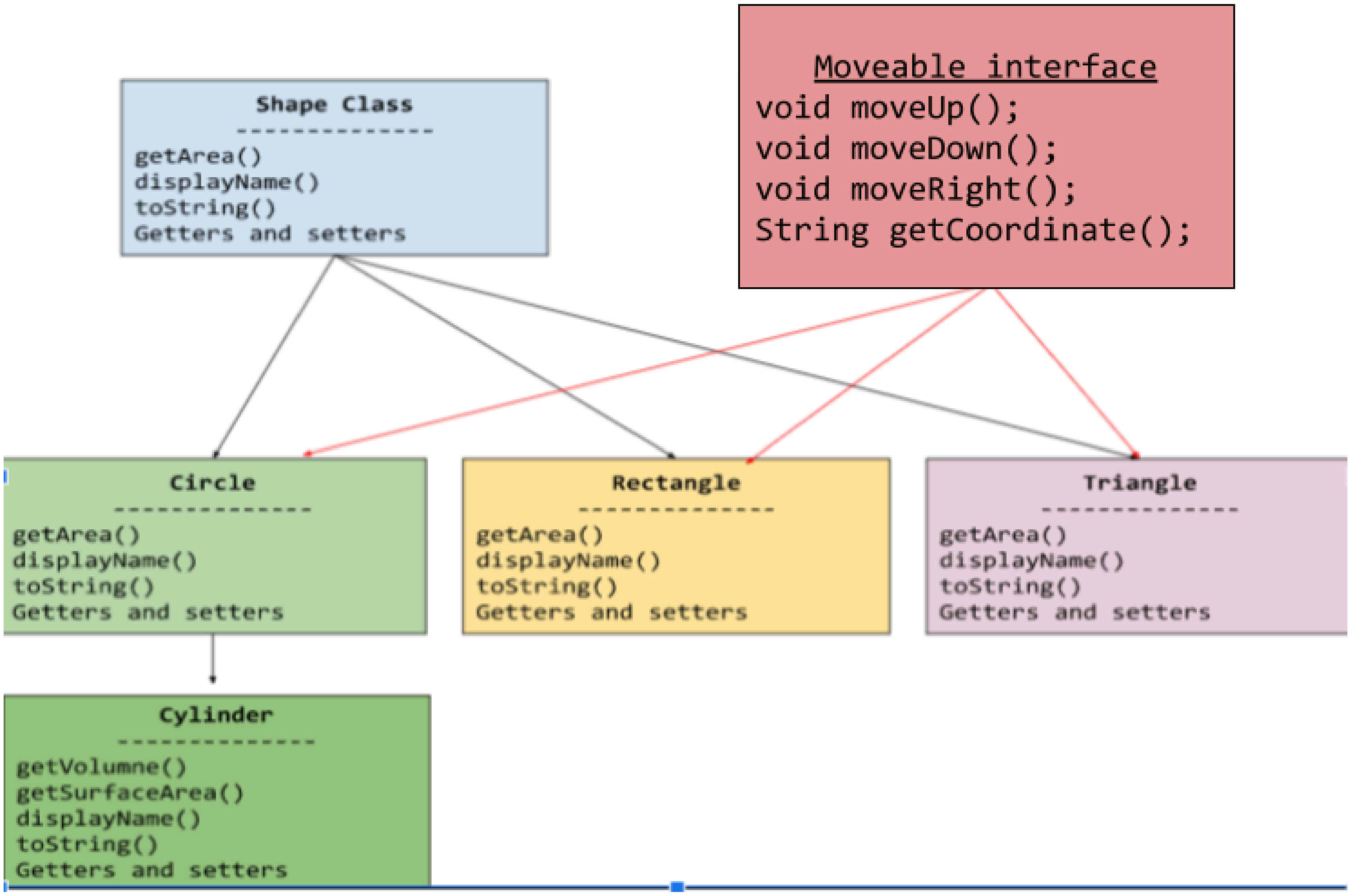
By the end of this lesson, learners will be able to use Java interfaces.

An Interface is similar to an abstract class with no fields and all abstract methods. Interfaces cannot be instantiated — they can only be *implemented* by classes. The purpose of an Interface is to specify behavior for a class.

In other words, we can say that an Interface is a design contract. It specifies methods and classes that can "implement" the Interface; and thereby, sign the contract.

We will use the ***Shapes*** example in this lab.

> Suppose that our application involves many shapes that can move. We could define an ***interface*** as ***movable***, containing the signatures of the various movement methods.



## Begin

Create a class named **Shape**. This will be an Abstract class and a Super class. Write the code below:

```
public abstract class Shape {
  protected String color;
  protected double height;  // To hold height.
  protected double width;  //To hold width
  protected double base;  //To  hold base

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

  public void setWidth(double width) {
     this.width = width;
  }

  public void setHeight(double height) {
     this.height = height;
  }

  public void setBase(double base) {
     this.base = base;
  }

// The getArea method is abstract.
  // It must be overridden in a subclass.
  /** All shapes must provide a method called getArea() */
  public abstract double getArea();
  /** Returns a self-descriptive string */

  public String toString() {
     return "Shape[color=" + color + "]";
  }

  public void displayshapName()
  {
     System.out.println("I am a Shape.");
  }
}
```

Create an Interface named **Movable.** It is similar to creating a new class, as shown below:

```
public interface Movable {// An interface defines a list of public
abstract methods to be implemented by the subclasses

   void moveUp();    // "public" and "abstract" by default
   void moveDown();
   void moveLeft();
   void moveRight();
   String getCoordinate();
}
```

Similar to an **abstract class, an Interface cannot be instantiated** because it is incomplete (the abstract method's body is missing). To use an interface, you must derive subclasses and provide ***implementation*** to all of the ***abstract methods*** declared in the interface. The subclasses are now complete and can be instantiated.

To derive subclasses from an interface, a new keyboard **"implement"** is to be used instead of **"extends"** for deriving subclasses from an ordinary class or an abstract class. It is important to note that the subclass implementing an interface needs to override ALL abstract methods defined in the interface; otherwise, the subclass cannot be compiled.

Create a class named **`Circle`**. This will be a Child class. Write the code below.

*The new constructor will add in the Circle class for coordinates and radius.*

```
public class Circle extends Shape implements Movable {
   protected double radius;
   private int x, y;    // x and y coordinates of the point
   private final double PI = Math.PI;

   /** Constructs a MovablePoint instance at the given x and y */
   public Circle(int x, int y, double radius) {
       this.x = x;
       this.y = y;
       this.radius = radius;
   }

   public Circle(double radius) {
       this.radius = radius;
   }

   public Circle(double radius, double height) {
       this.radius = radius;
```

```
        super.height = height;
    }
@Override
    public double getArea() {
        //double area = PI * this.radius * this.radius;
        double area = PI * Math.pow(this.radius, 2); // initializing value
in parent class variable
        return area; //reference to  parent class variable
    }
    @Override
    public void displayshapName() {
        System.out.println("Drawing a Circle of radius " + this.radius);
    }
    /** Returns a self-descriptive string */
    @Override
    public String toString() {
        return "Circle[ radius = " + radius + super.toString() +  "] ";
    }

    public String getCoordinate()
    {
      return  "(" + x + "," + y + ")";
    }

    // Need to implement all the abstract methods defined in the interface
Movable
    @Override
    public void moveUp() {
        y++;
    }
    @Override
    public void moveDown() {
        y--;
    }
    @Override
    public void moveLeft() {
        x--;
    }
    @Override
    public void moveRight() {
        x++;
    }
}
```

Create a class named **myRunner**. This will be the Main class or **entry point** for the application. Write the code below.

```
public class myRunner {
   public static void main(String[] args) {

     Circle c1 = new Circle(1, 2, 2);
     System.out.println("Area of Circle " + c1.getArea());
     System.out.println("Coordinates are " + c1.getCoordinate());

     c1.moveDown();
     System.out.println("After move Down, Coordinates are " + c1.getCoordinate());

       c1.moveRight();
       System.out.println("After move right, Coordinates are " + c1.getCoordinate());

       c1.moveUp();
       System.out.println("After move Up, Coordinates are " + c1.getCoordinate());

       c1.moveLeft();
       System.out.println("After move left, Coordinates are " + c1.getCoordinate());


       System.out.println("--------Test Polymorphism-------");
       Movable c2 = new Circle(5, 10, 200);  // upcast
       c2.moveUp();
       System.out.println("After move up , Coordinates are " + c2.getCoordinate());

       c2.moveLeft();
       System.out.println("After move Left , Coordinates are " + c2.getCoordinate());
        }
}
```

We can also upcast subclass instances to the **Movable** interface via Polymorphism, similar to an abstract class.

Output:
```
Area of Circle 12.566370614359172
Coordinates are (1,2)
After move Down, Coordinates are (1,3)
After move right, Coordinates are (2,3)
After move Up, Coordinates are (2,2)
After move left, Coordinates are (1,2)
--------Test Polymorphism-------
After move up , Coordinates are (5,9)
After move Left , Coordinates are (4,9)
```

## Practice Task:

Create two classes: **`Rectangle`** and **`Triangle.`** Extend both classes from the Shape class, and give an implementation of the Movable interface.

---

## Submission Instructions:

Include the following deliverables in your submission -

- Submit your source code using the Start Assignment button in the top-right corner of the assignment page in Canvas.

PER SCHOLAS

## CANVAS STAFF USE ONLY: Canvas Submission Guideline:

| Instructions for Canvas Assignment Creation |
|---|
| Assignment Name: GLAB - 303.10.4 - How to use Interface<br><br>Points: 100<br>Assignment Group: Module 303: Java SE Review (Not Graded)<br>Display Grade As: Complete/Incomplete<br>Do not count this assignment towards the final grade: Checked<br>Submission Types: Document File or Source Code Files<br><br>Everything else is the default. |