COMP2511

Object Oriented Programming (OOP) in Java

Prepared by

Dr. Ashesh Mahidadia

OOP in Java

❖ Object Oriented Programming (OOP)

❖ Inheritance in OOP

❖ Introduction to Classes and Objects

❖ Subclasses and Inheritance

❖ Abstract Classes

❖ Single Inheritance versus Multiple Inheritance

❖ Interfaces

❖ Method Forwarding (Has-a relationship)

❖ Method Overriding (Polymorphism)

❖ Method Overloading

❖ Constructors

Interfaces in Java

❖ Interfaces are like abstract classes, but with few important differences.

❖ All the methods defined within an interface are implicitly abstract. (We don't need to use abstract keyword, however, to improve clarity one can use abstract keyword).

❖ Variables declared in an interface must be static and final, that means, they must be constants.

❖ Just like a class extends its superclass, it also can optionally implements an interface.

❖ In order to implement an interface, a class must first declare the interface in an implements clause, and then it must provide an implementation for all of the abstract methods of the interface.

❖ A class can "implements" more than one interfaces.

❖ More discussions on "interfaces" later in the course.

Interfaces in Java: Example
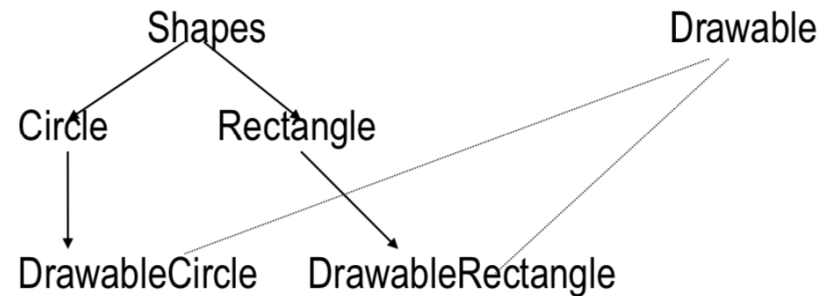

```java
public  interface Drawable {
    public void setColor(Color c);
    public void setPosition(double x, double y);
    public void draw(Graphics  g);
}

public class DrawableRectangle
                extends Rectangle
                implements Drawable  {

    private  Color c;
    private double x, y;


        ……

        ……
    // Here are implementations of the
    // mehtods in Drawable
    // we also inherit all public methods
    // of Rectangle

    public void setColor(Color c) { this.c = c;}
    public void setPosition(double x, double y){
        this.x = x; this.y = y;}
    public void draw(Graphics g) {
        g.drawRect(x,y,w,h,c); }
}
```

Using Interfaces: Example

❖ When a class implements an interface, instance of that class can also be assigned to variables of the interface type.

```java
Shape[] shapes = new Shape[3];
Drawable[] drawables = new Drawable[3];


DrawableCircle dc = new DrawableCircle(1.1);
DrawableSquare ds = new DrawableSquare(2.5);
DrawableRectangle dr = new DrawableRectangle(2.3,
4.5);

// The shapes can be assigned to both arrays
shapes[0] = dc; drawables[0] = dc;
shapes[1] = ds; drawables[1] = ds;
shapes[2] = dr; drawables[2] = dr;

// We can invoke abstract method
// in Drawable and Shapes

double total_area = 0;
for(int i=0; i< shapes.length; i++) {

    total_area += shapes[i].area();

    drawables[i].setPosition(i*10.0, i*10.0);

    // assume that graphic area 'g' is
    // defined somewhere
    drawables[i].draw(g);
}
```
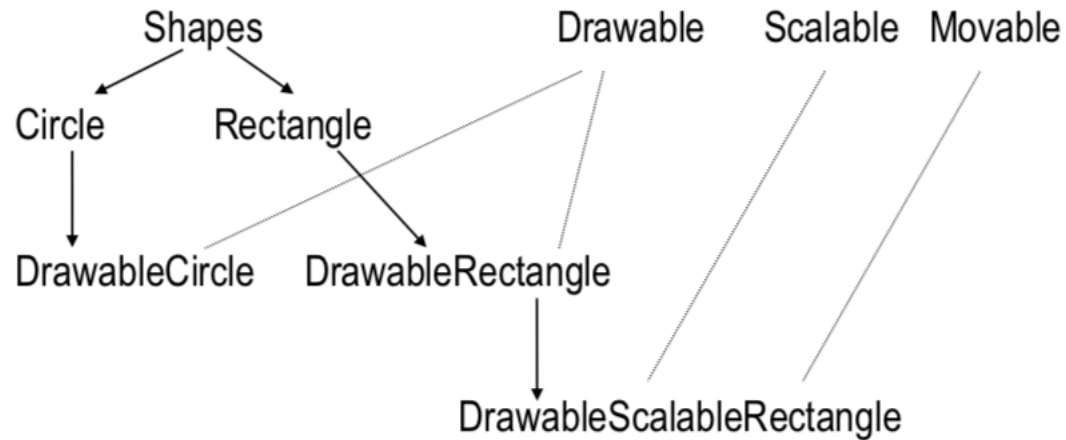
Implementing Multiple Interfaces

A class can implements more than one interfaces. For example,


```
public class DrawableScalableRectangle
        extends DrawableRectangle
        implements  Movable,  Scalable {

//   methods go here ....

}
```

Extending Interfaces

❖ Interfaces can have sub-interfaces, just like classes can have subclasses.

❖ A sub-interface inherits all the abstract methods and constants of its super-interface, and may define new abstract methods and constants.

❖ Interfaces can extend more than one interface at a time. For example,

```
public interface Transformable
        extends Scalable,Rotable,Reflectable {}

public interface DrawingObject
        extends Drawable, Transformable{}

public class Shape implements DrawingObject {
        ... }
```