CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Syllabus

MODULE III JAVA OBJECTS – 2	L	T	P	EL
	3	0	4	3
Inheritance and Polymorphism – Super classes and sub classes, overriding, object class and its methods, casting, instance of, Array list, Abstract Classes, Interfaces, Packages, Exception Handling				
SUGGESTED ACTIVITIES : • flipped classroom • Practical - implementation of Java programs – use Inheritance, polymorphism, abstract classes and interfaces, creating user defined exceptions • EL – dynamic binding, need for inheritance, polymorphism, abstract classes and interfaces				
SUGGESTED EVALUATION METHODS: • Assignment problems • Quizzes				

A Superclass Variable Can Reference a Subclass Object

- A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass.
- The type of the reference variable—not the type of the object that it refers to—that determines what members can be accessed.
  - That is, when a reference to a subclass object is assigned to a superclass reference variable, will have access only to those parts of the object defined by the superclass.

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use  -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

```
// Here, Box is extended to include weight.
class BoxWeight extends Box {

    double weight; // weight of box

    // constructor for BoxWeight
    BoxWeight(double w, double h, double d, double m) {
      width = w;
      height = h;
      depth = d;
      weight = m;
    }
  }
```

```
class DemoBoxWeight {
  public static void main(String args[]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
    double vol;

    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();

    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);
    System.out.println("Weight of mybox2 is " + mybox2.weight);
  }
}
```

The output from this program is shown here:

```
Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3

Volume of mybox2 is 24.0
Weight of mybox2 is 0.076
```

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use  -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

```
// Here, Box is extended to include weight.
class BoxWeight extends Box {

    double weight; // weight of box

    // constructor for BoxWeight
    BoxWeight(double w, double h, double d, double m) {
      width = w;
      height = h;
      depth = d;
      weight = m;
    }
  }
```

```
// Here, Box is extended to include color.
class ColorBox extends Box {
  int color; // color of box

  ColorBox(double w, double h, double d, int c) {
    width = w;
    height = h;
    depth = d;
    color = c;
  }
}
```

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use  -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

```
// Here, Box is extended to include weight.
class BoxWeight extends Box {

    double weight; // weight of box

    // constructor for BoxWeight
    BoxWeight(double w, double h, double d, double m) {
      width = w;
      height = h;
      depth = d;
      weight = m;
    }
  }
```

```
class RefDemo {
  public static void main(String args[]) {
    BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
    Box plainbox = new Box();
    double vol;

    vol = weightbox.volume();
    System.out.println("Volume of weightbox is " + vol);
    System.out.println("Weight of weightbox is " +
                       weightbox.weight);
    System.out.println();

    // assign BoxWeight reference to Box reference
    plainbox = weightbox;

    vol = plainbox.volume(); // OK, volume() defined in Box
    System.out.println("Volume of plainbox is " + vol);

    /* The following statement is invalid because plainbox
       does not define a weight member. */
//  System.out.println("Weight of plainbox is " + plainbox.weight);
  }
}
```

Using super

- super has two general forms. The first calls the superclass' constructor. The second is used to access a member of the superclass that has been hidden by a member of a subclass.
- Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.
- Using super to Call Superclass Constructors
  - super(arg-list);
  - Here, arg-list specifies any arguments needed by the constructor in the superclass.
  - super( ) must always be the first statement executed inside a subclass' constructor.

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use  -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}
```

```
// BoxWeight now uses super to initialize its Box attributes.
class BoxWeight extends Box {
  double weight; // weight of box

  // initialize width, height, and depth using super()
  BoxWeight(double w, double h, double d, double m) {
    super(w, h, d); // call superclass constructor
    weight = m;
  }
}
```

```
// Here, Box is extended to include color.
class ColorBox extends Box {
  int color; // color of box

  ColorBox(double w, double h, double d, int c) {
    width = w;
    height = h;
    depth = d;
    color = c;
  }
}
```

```
// A complete implementation of BoxWeight.
class Box {
  private double width;
  private double height;
  private double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
  }

  // constructor used when all dimensions specified
  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  // constructor used when no dimensions specified
  Box() {
    width = -1;  // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  // constructor used when cube is created
  Box(double len) {
    width = height = depth = len;
  }

  // compute and return volume
  double volume() {
    return width * height * depth;
  }
}

// BoxWeight now fully implements all constructors.
class BoxWeight extends Box {
  double weight; // weight of box

  // construct clone of an object
  BoxWeight(BoxWeight ob) { // pass object to constructor
    super(ob);
    weight = ob.weight;
  }

  // constructor when all parameters are specified
  BoxWeight(double w, double h, double d, double m) {
    super(w, h, d); // call superclass constructor
    weight = m;
  }

  // default constructor
  BoxWeight() {
    super();
    weight = -1;
  }

  // constructor used when cube is created
  BoxWeight(double len, double m) {
    super(len);
    weight = m;
  }
}

class DemoSuper {
  public static void main(String args[]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
    BoxWeight mybox3 = new BoxWeight(); // default
    BoxWeight mycube = new BoxWeight(3, 2);
    BoxWeight myclone = new BoxWeight(mybox1);
    double vol;

    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();

    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);
    System.out.println("Weight of mybox2 is " + mybox2.weight);
    System.out.println();

    vol = mybox3.volume();
    System.out.println("Volume of mybox3 is " + vol);
    System.out.println("Weight of mybox3 is " + mybox3.weight);
    System.out.println();

    vol = myclone.volume();
    System.out.println("Volume of myclone is " + vol);
    System.out.println("Weight of myclone is " + myclone.weight);
    System.out.println();

    vol = mycube.volume();
    System.out.println("Volume of mycube is " + vol);
    System.out.println("Weight of mycube is " + mycube.weight);
    System.out.println();
  }
}
```

This program generates the following output:

```
Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3

Volume of mybox2 is 24.0
Weight of mybox2 is 0.076

Volume of mybox3 is -1.0
Weight of mybox3 is -1.0

Volume of myclone is 3000.0
Weight of myclone is 34.3

Volume of mycube is 27.0
Weight of mycube is 2.0
```

Polymorphism

- Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class.

```
public class TestBikes {
  public static void main(String[] args){
    Bicycle bike01, bike02, bike03;

    bike01 = new Bicycle(20, 10, 1);
    bike02 = new MountainBike(20, 10, 5, "Dual");
    bike03 = new RoadBike(40, 20, 8, 23);

    bike01.printDescription();
    bike02.printDescription();
    bike03.printDescription();
  }
}
```

The Java virtual machine (JVM) calls the appropriate method for the object that is referred to in each variable. It does not call the method that is defined by the variable's type. This behavior is referred to as *virtual method invocation* and demonstrates an aspect of the important polymorphism features in the Java language.

Overriding and Hiding Methods

- **Instance Methods**
  - An instance method in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass *overrides* the superclass's method.
- **Static Methods**
  - a subclass defines a static method with the same signature as a static method in the superclass, then the method in the subclass *hides* the one in the superclass.
- The distinction between hiding a static method and overriding an instance method has important implications:
  - The version of the overridden instance method that gets invoked is the one in the subclass.
  - The version of the hidden static method that gets invoked depends on whether it is invoked from the superclass or the subclass.

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

```
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }

    public static void main(String[] args) {
        Cat myCat = new Cat();
        Animal myAnimal = myCat;
        Animal.testClassMethod();
        myAnimal.testInstanceMethod();
    }
}
```

The output from this program is as follows:

The static method in Animal
The instance method in Cat

External learning

- Refer the programs(inheritance, constructors, this, super, polymorphism-overloading vs overriding) in the given link.
- https://cse.iitkgp.ac.in/~dsamanta/java/ch3.htm#Inheritance