faheem Akhtar Rajputt

Object Oriented Programming (JAVA)

Lecture 16


Sukkur
IBA
Chapter # 08
Inheritance

Inheritance

- Cornerstone of OOP
- Allows the creation of hierarchical classification
- The class that's inherited is called *Superclass*
- And the class that does the inheritance is called *subclass*.
- Contains all of instance variables and methods of superclass plus its own unique elements

extends

- Word **extends** is used to inherit the class

SYNTAX:

```
-----------------------------------------------
class superclass-name {
      //body of class;
}
-----------------------------------------------
class subclass-name extends superclass-name {
     //body of class;
}
-----------------------------------------------
```

Example

```
OUTPUT
Contents of superOb:
i and j: 10 20

Contents of subOb:
i and j: 7 8
k: 9

Sum of i, j and k in subOb:
i+j+k: 24
```

```
// Create a superclass.
class A {
  int i, j;

  void showij() {
    System.out.println("i and j: " + i + " " + j);
  }
}

// Create a subclass by extending class A.
class B extends A {
  int k;

  void showk() {
    System.out.println("k: " + k);
  }
  void sum() {
    System.out.println("i+j+k: " + (i+j+k));
  }
}
```

```
class SimpleInheritance {
  public static void main(String args[]) {
    A superOb = new A();
    B subOb = new B();

    // The superclass may be used by itself.
    superOb.i = 10;
    superOb.j = 20;
    System.out.println("Contents of superOb: ");
    superOb.showij();
    System.out.println();

    /* The subclass has access to all public members of
       its superclass. */
    subOb.i = 7;
    subOb.j = 8;
    subOb.k = 9;
    System.out.println("Contents of subOb: ");
    subOb.showij();
    subOb.showk();
    System.out.println();

    System.out.println("Sum of i, j and k in subOb:");
    subOb.sum();
  }
}
```

Supper & Subclass

- Only single inheritance is supported with java.
- A superclass can be inherited in more than one java programs.
- Superclass is still an independent working class
- Can't inherit class itself

Member Access

- It can't access those methods/functions which are **private**

```
class Access {
public static void main(String args[]) {
  B subOb = new B();

  subOb.setij(10, 12);

  subOb.sum();
  System.out.println("Total is " + subOb.total);
 }
}
```

```
class A {
  int i; // public be default
  private int j; // private to A

  void setij(int x, int y) {
    i = x;
    j = y;
  }
}

// A's j is not accessible here.
class B extends A {
  int total;

  void sum() {
    total = i + j; // ERROR, j is not accessible here
  }
}
```

```
class Box {
  double width;
  double height;
  double depth;

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

  Box(double w, double h, double d) {
    width = w;
    height = h;
    depth = d;
  }

  Box() {
    width = -1;  // use -1 to indicate
    height = -1; // an uninitialized
    depth = -1;  // box
  }

  Box(double len) {
    width = height = depth = len;
  }
  double volume() {
    return width * height * depth;
  }
}
```

```
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;
  }
}
```

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
```

```
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);
  }
}
```

Advantages

- It is not necessary for BoxWeight to re-create all of the features found in Box. It can simply extend Box to meet its own purposes.
- Once you have created a superclass that defines common set of attributes, these can be used to create more specific subclasses

Extension…

```
// 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;
  }
```

Reference a Subclass Object

- A reference variable of a superclass can be assigned a reference to any subclass derived from that superclass.

Extending previous Example

```
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* [1/4]

- Subclass can call a constructor defined by its superclass by use of the following form of super:

  super(*arg-list*);

- improved version of the BoxWeight( ) class:

```
// 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;
  }
}
```

1 USAGE

Using *super*[2/4]

- In the preceding example, super( ) was called with three arguments. Since constructors can be overloaded
- super( ) can be called using any form defined by the superclass.
- In each case, super( ) is called using the appropriate arguments.

```
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();
  }
}
```

Using *super*[3/4]

- The second form of super acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used.

super.member

- Here, member can be either a method or an instance variable.

```
// Using super to overcome name hiding.
class A {
  int i;
}

// Create a subclass by extending class A.
class B extends A {
  int i; // this i hides the i in A

  B(int a, int b) {
    super.i = a; // i in A
    i = b; // i in B
  }

  void show() {
    System.out.println("i in superclass: " + super.i);
    System.out.println("i in subclass: " + i);
  }
}

class UseSuper {
  public static void main(String args[]) {
    B subOb = new B(1, 2);

    subOb.show();
  }
}
```

2 USAGE

OUTPUT

```
i in superclass: 1
i in subclass: 2
```

Questions?


Sukkur
IBA
Thanks…