OBJECT ORIENTED PROGRAMMING

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

USING SUPER

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

- The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

What if the superclass data is kept private?

- **super** gives the solution. Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword **super**.

USING SUPER

- **super has two general forms:**
  - The 1st calls the superclass constructor.
  - The 2nd is used to access a method of the superclass that has been hidden by a method of a subclass

USING SUPER TO INVOKE SUPERCLASS CONSTRUCTOR

- A subclass can call a constructor method defined by its superclass by use of the following form of **super**:

  super(parameter-list);

- *super()* must **always be the first statement** executed inside a subclass' constructor.

USING SUPER TO INVOKE SUPERCLASS CONSTRUCTOR

```
// 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;
  }
}
```

USING SUPER TO INVOKE SUPERCLASS CONSTRUCTOR

- Since constructors can be overloaded, super() can be called using any form defined by the superclass.
- The constructor executed will be the one that matches the arguments.

USAGE OF JAVA SUPER KEYWORD

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

USING SUPER TO INVOKE SUPERCLASS CONSTRUCTOR

```
// 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;
 }
```

Program continues on next slide …

```
// 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
 }
```

Program continues on next slide …

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

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

Program continues on next slide …

```
// 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;
 }
```

When invoking Box(Box ob), super() is called with an object of type BoxWeight – not of type Box. This is because a superclass variable can be used to reference any object derived from that class.

Program continues on next slide …

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

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

Program continues on next slide …

```
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(); // no aurgment constructor // default
  BoxWeight mycube = new BoxWeight(3, 2);
  BoxWeight myclone = new BoxWeight(mybox1);
  double vol = mybox1.volume();
  System.out.println("Volume of mybox1 is " + vol);
  System.out.println("Weight of mybox1 is " + mybox1.weight);
  vol = mybox2.volume();
  System.out.println("Volume of mybox2 is " + vol);
  System.out.println("Weight of mybox2 is " + mybox2.weight);
  vol = mybox3.volume();
  System.out.println("Volume of mybox3 is " + vol);
  System.out.println("Weight of mybox3 is " + mybox3.weight);
  vol = myclone.volume();
  System.out.println("Volume of myclone is " + vol);
  System.out.println("Weight of myclone is " + myclone.weight);
  vol = mycube.volume();
  System.out.println("Volume of mycube is " + vol);
  System.out.println("Weight of mycube is " + mycube.weight);   } }
```

USING SUPER TO ACCESS HIDDEN MEMBERS OF SUPERCLASS

- 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.
- It has the following general form

  super.member

- This form of **super** is most applicable to situations in which member names of a subclass hide members by the same name in the superclass.

USING SUPER TO ACCESS HIDDEN MEMBERS OF SUPERCLASS

```
// 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
 }
```

USING SUPER TO ACCESS HIDDEN MEMBERS OF SUPERCLASS

```
 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();
 }
}
```

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