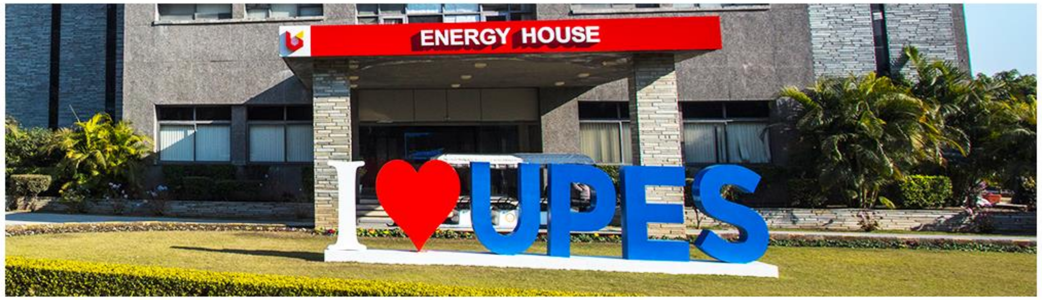

UPES

UNIVERSITY WITH A PURPOSE

Object Oriented Programming

Inheritance

- To inherit a class, we simply incorporate the definition of one class into another by using the **extends** keyword.
- The general form of a **class** declaration that inherits a superclass is shown here:

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

- You can only specify one superclass for any subclass that you create.
- Java does not support the inheritance of multiple super classes into a single subclass.
- You can, as stated, create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass. However, no class can be a superclass of itself.

Inheritance

```
Example: // 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();
}
}
```

Member Access and Inheritance

- Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as **private**.

```
// Create a superclass.
class A {
int i; // public by 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 Access {
public static void main(String args[]) {
B subOb = new B();
subOb.setij(10, 12);
subOb.sum();
System.out.println("Total is "+subOb.total);}}
```

Note: A class member that has been declared as **private** will remain private to its class. It is not accessible by any code outside its class, including subclasses.

Member Access and Inheritance

```
// 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); } }
```

Member Access and Inheritance

Note: A major advantage of inheritance is that once you have created a superclass that defines the attributes common to a set of objects, it can be used to create any number of more specific subclasses. Each subclass can precisely tailor its own classification. For example, the following class inherits **Box** and adds a color attribute:

```
// 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; } }
```

- Remember, once you have created a superclass that defines the general aspects of an object, that superclass can be inherited to form specialized classes.
- Each subclass simply adds its own unique attributes. This is the essence of inheritance.

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. This aspect of inheritance quite useful in a variety of situations. For 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); }
}
```

A Superclass Variable Can Reference a Subclass Object

Observation:

- Here, **weightbox** is a reference to **BoxWeight** objects, and **plainbox** is a reference to **Box** objects.
- Since **BoxWeight** is a subclass of **Box**, it is permissible to assign **plainbox** a reference

to the **weightbox** object.

- It is important to understand that it is 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, you will have access only to those parts of the object defined by the superclass.
- This is why **plainbox** can't access **weight** even when it refers to a **BoxWeight** object.
- If you think about it, this makes sense, because the superclass has no knowledge of what a subclass adds to it.
- This is why the last line of code in the preceding fragment is commented out. It is not possible for a Box reference to access the weight field, because Box does not define one.

Using super keyword

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

1. Using super to Call Superclass Constructors:

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

```
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.

Using super keyword (cont..)

```
// 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;
}
}
```

Observation:

1. Here, **BoxWeight( )** calls **super( )** with the arguments **w**, **h**, and **d**. This causes the **Box** constructor to be called, which initializes **width**, **height**, and **depth** using these values. **BoxWeight** no longer initializes these values itself. It only needs to initialize the value unique to it: **weight**.
2. 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. The constructor executed will be the one that matches the arguments.

Using super keyword (cont..)

```
// 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;
}
```

Using super keyword (cont..)

```
// 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 keyword (cont..)

Note:

1. Notice that **super( )** is passed an object of type **BoxWeight**—not of type **Box**. This still invokes the constructor **Box(Box ob)**. As mentioned earlier, a superclass variable can be used to reference any object derived from that class. Thus, we are able to pass a **BoxWeight** object to the **Box** constructor. Of course, **Box** only has knowledge of its own members.

2. When a subclass calls **super( )**, it is calling the constructor of its immediate superclass. Thus, **super()** always refers to the superclass immediately above the calling class. This is true even in a multileveled hierarchy. Also, **super( )** must always be the first statement executed inside a subclass constructor.

Using super keyword (cont..)

2. A Second Use for super:

- 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. This usage has the following general form:

```
super.member
```

-- Here, *member* can be either a method or an instance variable. This second 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 keyword (cont..)

```
// 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();
}
}
```

This program displays the following:

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

Although the instance variable **i** in **B** hides the **i** in **A**, **super** allows access to the **I** defined in the superclass. As you will see, **super** can also be used to call methods that are hidden by a subclass.

References

Schildt, H. (2014). *Java: the complete reference*. McGraw-Hill Education Group.