Inheritance

- Inheritance allows the creation of hierarchical classifications.

- Using inheritance, you can create a general class that defines traits common to a set of related items.

- This class can then be inherited by other, more specific classes, each adding those things that are unique to it.

- In the terminology of Java, a class that is inherited is called a *superclass*. The class that does the inheriting is called a *subclass*.

- Therefore, a subclass is a specialized version of a superclass.

- It inherits all of the instance variables and methods defined by the superclass and adds its own, unique elements.

Syntax

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

```
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( );
}
}
```

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
```

Member Access and Inheritance

```
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;
}
}
```

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

```
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);
}
}
```

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
```

A Superclass Variable Can Reference a Subclass Object

```
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

Used to refer immediate super class.
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.

Using super to Call Superclass Constructors

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

super(*parameter-list*);

- *parameter-list* specifies any parameters needed by the constructor in the superclass.

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

To see how **super( )** is used, consider this 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;
}
}
```

Inside main( ) method:

```
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
```

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*

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();
}
}
```

Output:

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

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

When Constructors Are Called

```
// Demonstrate when constructors are called.
// Create a super class.
class A {
A( ) {
System.out.println("Inside A's constructor.");
}
}
// Create a subclass by extending class A.
class B extends A {
B( ) {
System.out.println("Inside B's constructor.");
}
}
// Create another subclass by extending B.
class C extends B {
C( ) {
System.out.println("Inside C's constructor.");
}
}
```

```
class CallingCons {
public static void main(String args[]) {
C c = new C( );
}
}
```

Output:

```
Inside A’s constructor
Inside B’s constructor
Inside C’s constructor
```

Output:

Inside A’s constructor

Inside B’s constructor

Inside C’s constructor

ie., constructors are called in order of derivation, from superclass to subclass.