Programming in Java

Lecture 10: Inheritance

Mahesh Kumar
Assistant Professor (Adhoc)

Department of Computer Science
Acharya Narendra Dev College
University of Delhi

Course webpage
[http://www.mkbhandari.com/mkwiki]

Outline

1. Inheritance Basics

2. Using super

3. Creating a Multilevel Hierarchy

4. When Constructors are Executed

Inheritance

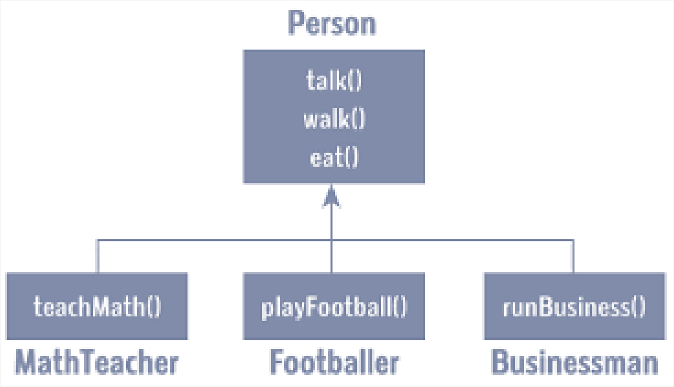
- One of the important concept/feature of Object Oriented Programming.
- It allows/facilitates Reusability through the Hierarchical Classification.

1 *Superclass*

- *Defines the general aspects of an object (attributes common to a set of objects) .*
- *It can be used to create any number of more specific subclasses.*
- *Also known as base class or parent class.*

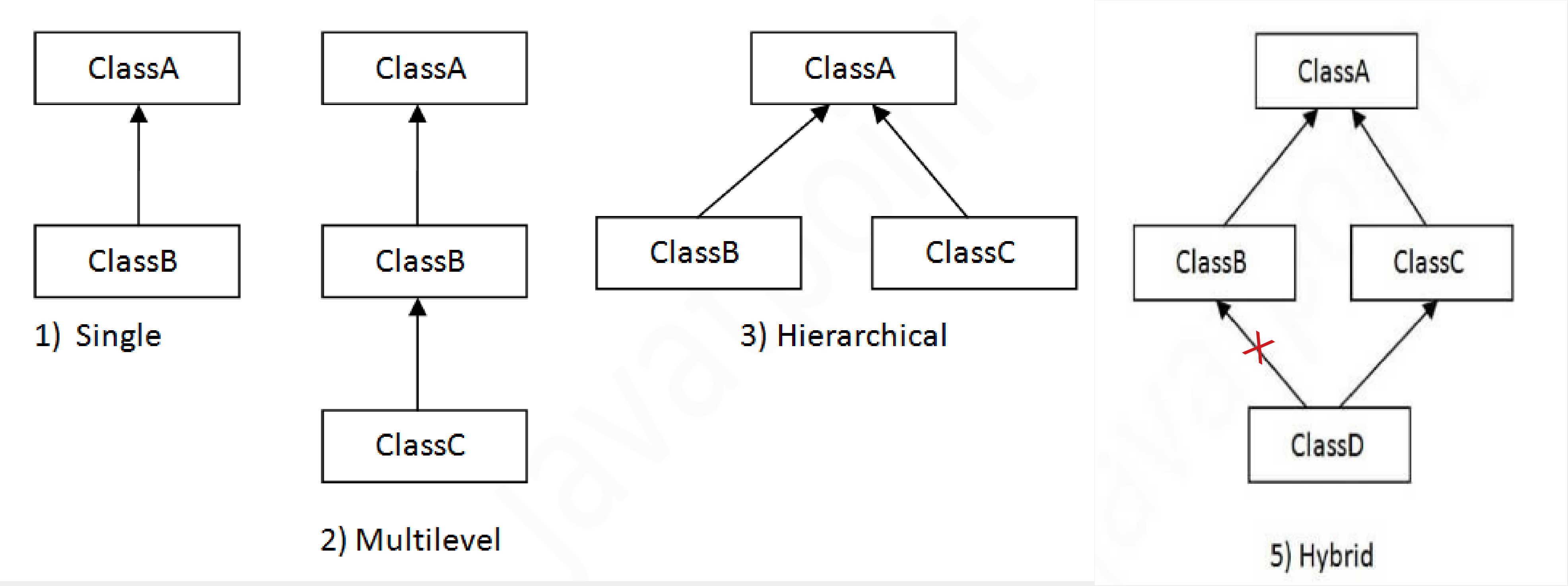
2 *Subclass*

- *Specialized version of a Superclass.*
- *Inherits the Superclass (common traits/properties).*
- *Adds things that are unique to it (its own, unique elements).*
- *Also known as derived class or child class.*



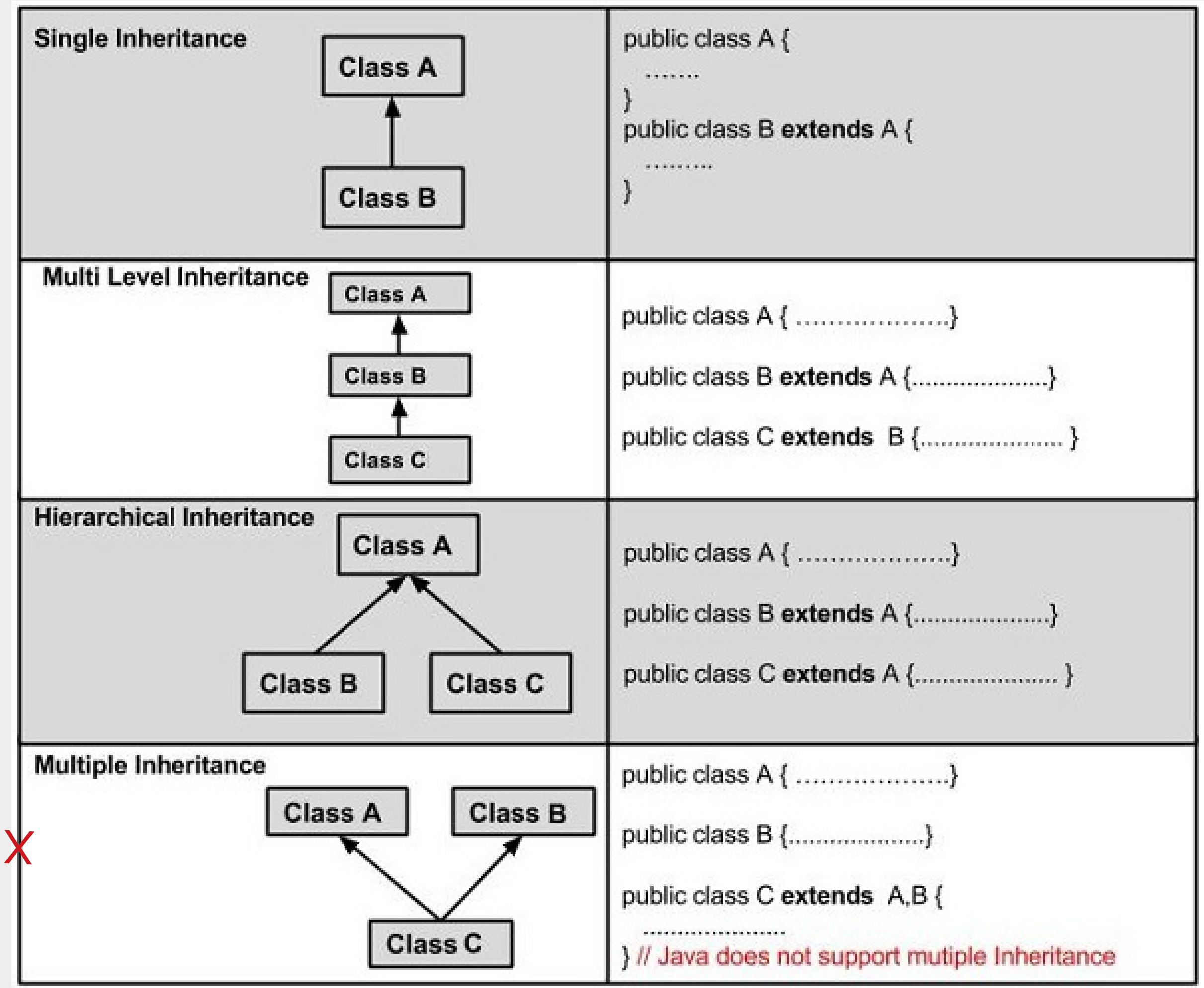
[Example of Inheritance]

Types of Inheritance in Java



[Types of Inheritance]

Types of Inheritance in Java



Single Inheritance
Class A
Class B
	public class A {
	…….
	}
	public class B **extends** A {
	………
	}

Multi Level Inheritance
Class A
Class B
Class C
	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** B {………………. }

Hierarchical Inheritance
Class A
Class B
Class C
	public class A { ……………….}
	public class B **extends** A {……………….}
	public class C **extends** A {………………. }

X **Multiple Inheritance**
Class A
Class B
Class C
	public class A { ……………….}
	public class B {……………….}
	public class C **extends** A,B {
	……………………
	} // Java does not support mutiple Inheritance

Inheritance Basics

- The extends keyword is used to inherit a class.
- The general form of a class declaration that inherits a Superclass is shown here:

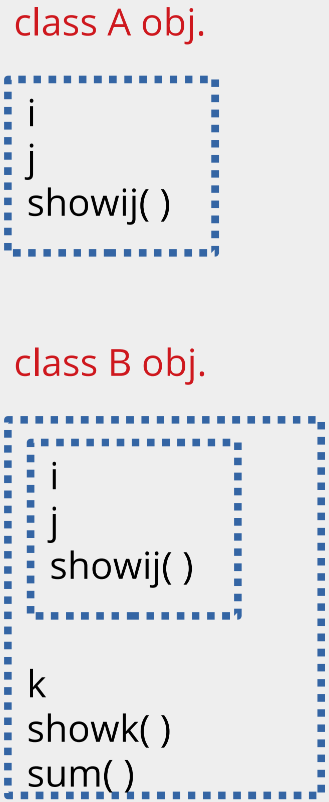
```
class  subclass-name extends superclass-name {

        // body of class

}
```

#Note:
superclass is also a completely independent, stand-alone class, can be used by itself.

1. You can only specify one superclass for any subclass that you create.
2. Multiple inheritance is not supported in Java.
3. You can create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass.
4. However, no class can be a superclass of itself.

Inheritance Basics – A simple example of Inheritance

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



Inheritance Basics – A simple example of Inheritance

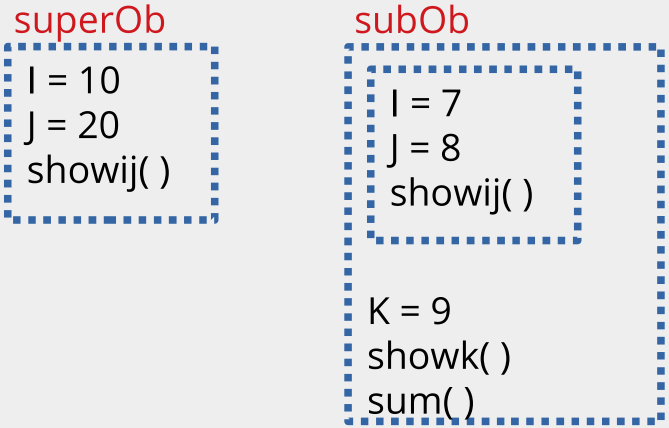
```
// Create a superclass.
class A {
    int i, j;
    void showij( ) {
        System.out.println("i and j: " + i + " " + j);
    }
}

// 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( );
    }
}
```

Inheritance Basics – A simple example of Inheritance

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
```



```
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

```
// Create a superclass.
class A {
        int i;                  // default access
        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);
        }
}
```

1. 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.
2. In a class hierarchy, private members remain private to their class.
3. #REMEMBER 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.

A More Practical Example

```
// 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;
}
}
```

A More Practical Example

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

A More Practical Example

OUTPU
T

```
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);
        }
}
```

A More Practical Example

- 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.

```
// 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 Superclass Variable Can Reference a Subclass Object

```
class RefDemo {
        public static void main(String args[ ]) {
                BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);         // weightbox is a reference to BoxWeight objects
                Box plainbox = new Box( );                                  // plainbox is a reference to Box objects.
                double vol;
                vol = weightbox.volume( );
                System.out.println("Volume of weightbox is " + vol);
                System.out.println("Weight of weightbox is " + weightbox.weight);

                // assign BoxWeight reference to Box reference, since BoxWeight is a subclass of Box
                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.
                1. 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.
                2. Because the superclass has no knowledge of what a subclass adds to it   */

        //      System.out.println("Weight of plainbox is " + plainbox.weight);
        }
}
```

Using Super

- So far inheritance were not implemented as efficiently or as robustly as they could have been. For example:

```
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;
        }
}
```

Using Super

- So far inheritance were not implemented as efficiently or as robustly as they could have been. For example:

```
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;
        }
}
```

1 The constructor for **BoxWeight** explicitly initializes the **width**, **height**, and **depth** fields of **Box.**

2 Two issues of concern:

- ***Duplicate code*** *in its superclass (inefficient)*
- *But it implies that a subclass must be* ***granted access*** *to these members*

3 However, there will be times when you will want to create a superclass that keeps the details of its implementation to itself (that is, that keeps its data members private).

4 In this case, there would be no way for a subclass to directly access or initialize these variables on its own.

Using Super

- So far inheritance were not implemented as efficiently or as robustly as they could have been. For example:

```
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;
        }
}
```

5 Since ***encapsulation*** is a primary attribute of OOP, it is not surprising that Java provides a solution to this problem.

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

7 **super** has two general forms:

- *Can be used to **Call the superclass' constructor***
- *Can be used to **access a member** of the superclass (private members)*

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);
```

```
// 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;
    }
}
```

0 Here, arg-list specifies any arguments needed by the constructor in the superclass

1 When a subclass calls super( ), it is calling the constructor of its immediate superclass.

2 Thus, super( ) always refers to the superclass immediately above the calling class.

3 This is true even in a multileveled hierarchy.

4 Also, super( ) must always be the first statement executed inside a subclass constructor.

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);
```

```
// 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;
        }
}
```

5 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.

6 BoxWeight no longer initializes these values itself. It only needs to initialize the value unique to it: **weight**.

7 This leaves Box free to make these values private if desired.

8 Since constructors can be overloaded, super( ) can be called using any form defined by the superclass.

Using Super to Call Superclass Constructors

```
// 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;
    }
}
```

Using Super to Call Superclass Constructors

```
// 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;
        }
}
```

Using Super to Call Superclass Constructors

```
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);  // clone

                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( );
```

Using Super to Call Superclass Constructors

```
        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 to Call Superclass Constructors

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
```

Using Super to Call Superclass Constructors

```
// construct clone of an object
BoxWeight(BoxWeight ob) {

        super(ob);
        weight = ob.weight;
}
```

- 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.

Using Super to access member 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.
- 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 to access member of Superclass

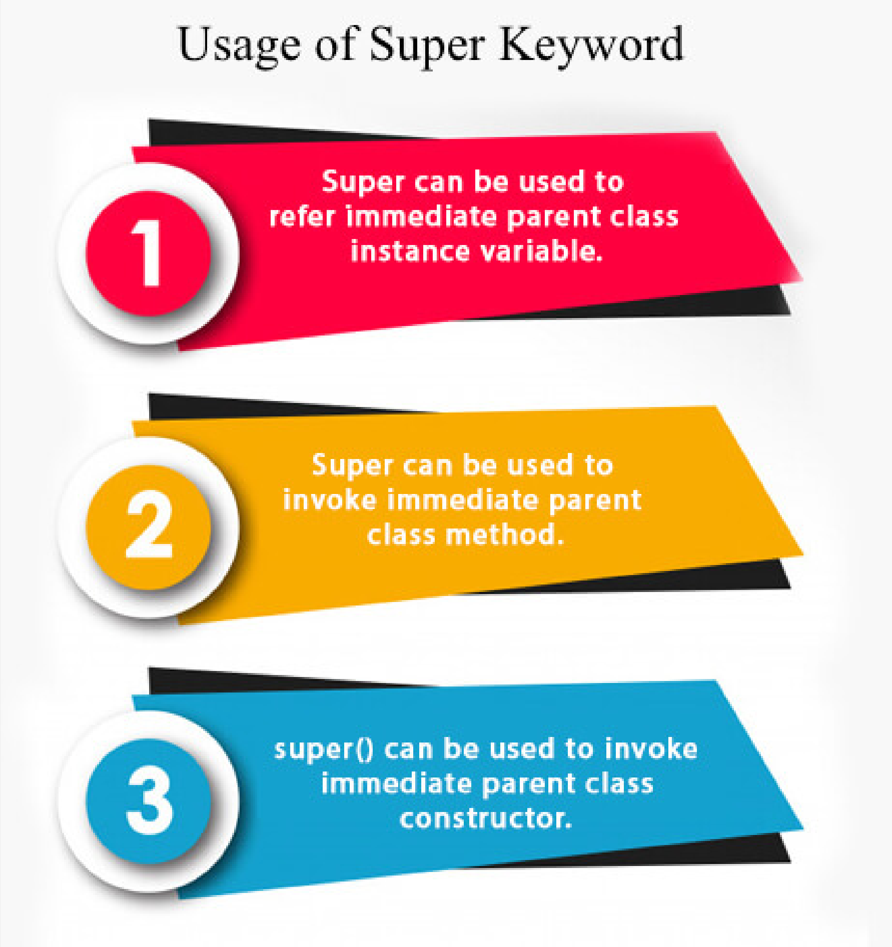
```
// Using super to overcome name hiding.
class A {
        int i;
}

// 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
        }
        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
```

Using Super - Summary



Usage of Super Keyword

1 Super can be used to refer immediate parent class instance variable.

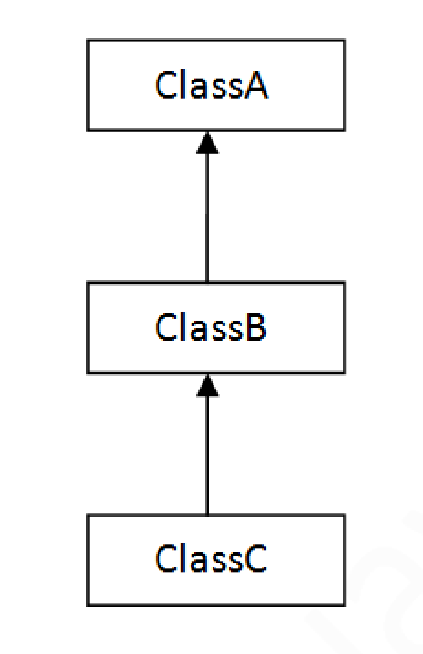
2 Super can be used to invoke immediate parent class method.

3 super() can be used to invoke immediate parent class constructor.

[Source: (3)]

Creating a Multilevel Hierarchy

- You can build hierarchies that contain as many layers of inheritance as you like.
- As mentioned, it is perfectly acceptable to use a subclass as a superclass of another.
- For example, given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A.
- When this type of situation occurs, each subclass inherits all of the traits found in all of its superclasses.
- In this case, C inherits all aspects of B and A.
- NOTE: The class hierarchy, including A, B, and C, can be in one file. In Java, all three classes can be placed into their own files and compiled separately. In fact, using separate files is the norm, not the exception, in creating class hierarchies.



Creating a Multilevel Hierarchy

```
// Extend BoxWeight to include shipping costs.
// 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;
    }
}
```

Creating a Multilevel Hierarchy

```
// Add weight
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;
        }
}
```

Creating a Multilevel Hierarchy

```
// Add shipping costs.
class Shipment extends BoxWeight {
        double cost;
        // construct clone of an object
        Shipment(Shipment ob) {    // pass object to constructor
                super(ob);
                cost = ob.cost;
        }

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

        // default constructor
        Shipment( ) {
                super( );
                cost = -1;
        }

        // constructor used when cube is created
        Shipment(double len, double m, double c) {
                super(len, m);
                cost = c;
        }
}
```

Creating a Multilevel Hierarchy

```
class DemoShipment {
        public static void main(String args[ ]) {
                Shipment shipment1 = new Shipment(10, 20, 15, 10, 3.41);
                Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.28);

                double vol;

                vol = shipment1.volume( );
                System.out.println("Volume of shipment1 is " + vol);
                System.out.println("Weight of shipment1 is " + shipment1.weight);
                System.out.println("Shipping cost: $" + shipment1.cost);
                System.out.println( );

                vol = shipment2.volume( );
                System.out.println("Volume of shipment2 is " + vol);
                System.out.println("Weight of shipment2 is "+ shipment2.weight);
                System.out.println("Shipping cost: $" + shipment2.cost);
        }
}
```

The output of this program is:

```
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41

Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
```

When Constructors are Executed?

- When a class hierarchy is created, in what order are the constructors for the classes that make up the hierarchy executed?
- For example, given a subclass called B and a superclass called A, is A's constructor executed before B's, or vice versa?
- The answer is that in a class hierarchy, constructors complete their execution **in order of derivation**, from superclass to subclass.
- Further, since super( ) must be the first statement executed in a subclass' constructor, this order is the same whether or not super( ) is used.
- If super( ) is not used, then the default or parameterless constructor of each superclass will be executed.

When Constructors are Executed?

```
// Demonstrate when constructors are executed.
// 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();
        }
}
```

The output of this program is:

```
Inside A's constructor
Inside B's constructor
Inside C's constructor
```

References

R Reference for this topic

- [Book: Java: The Complete Reference, Ninth Edition: Herbert Schildt]
  https://www.amazon.in/Java-Complete-Reference-Herbert-Schildt/dp/0071808558

- [Web: GeeksforGeeks]
  https://www.geeksforgeeks.org/java/

- [Web: Java T Point tutorial]
  https://www.javatpoint.com/java-tutorial