Using super

- In the preceding examples, classes derived from Box were not implemented as efficiently
- or as robustly as they could have been.
- For example, the constructor for BoxWeight explicitly initializes the width, height, and depth fields of Box.
- Not only does this duplicate code found in its superclass, which is inefficient, but it implies that a subclass must be granted access to these members.
  - 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).
  - In this case, there would be no way for a subclass to directly access or initialize these variables on its own.
  - 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.**

```
// 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;
}
}

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();
}
```

}

A Second Use for super

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. Consider this simple class hierarchy:

```
// Using super to overcome name hiding.

class A {

int i;

}

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();

}

}
```

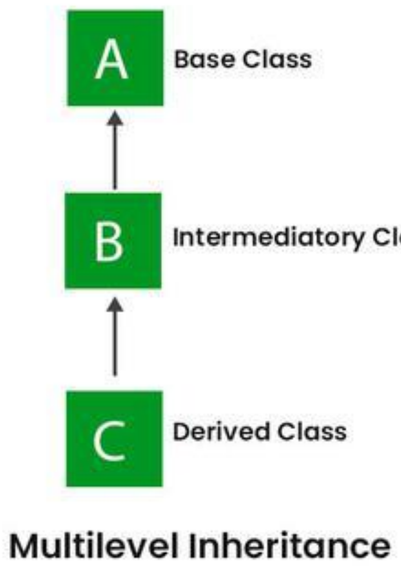
This program displays the following:

i in superclass: 1

i in subclass: 2

Multilevel Inheritance

- In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes.
- In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



Multilevel Inheritance

```
import java.io.*;
import java.lang.*;
import java.util.*;

class one {
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}
```

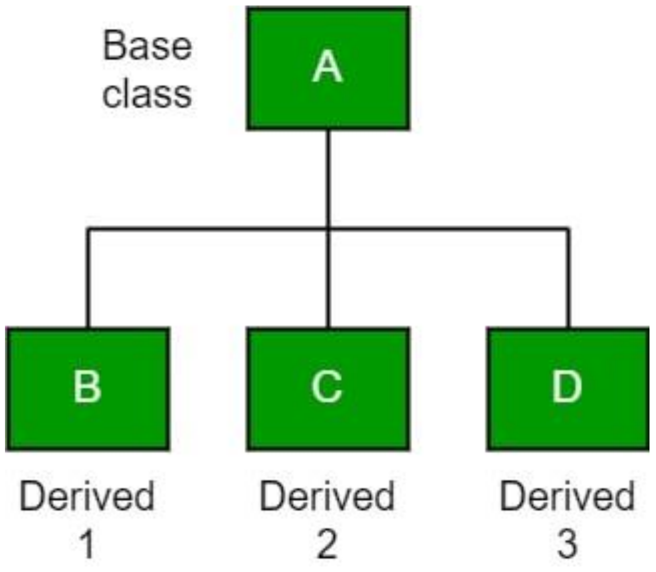
```
class two extends one {
    public void print_for() { System.out.println("for"); }
}

class three extends two {
    public void print_geek()
    {
        System.out.println("Geeks");
    }
}

// Drived class
public class Main {
    public static void main(String[] args)
    {
        three g = new three();
        g.print_geek();
        g.print_for();
        g.print_geek();
    }
}
```

Hierarchical Inheritance

- In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass.
- In the below image, class A serves as a base class for the derived classes B, C, and D.



```java
// Java program to illustrate the
// concept of Hierarchical  inheritance

class A {
    public void print_A() { System.out.println("Class A"); }
}

class B extends A {
    public void print_B() { System.out.println("Class B"); }
}

class C extends A {
    public void print_C() { System.out.println("Class C"); }
}

class D extends A {
    public void print_D() { System.out.println("Class D"); }
}

// Driver Class
public class Test {
    public static void main(String[] args)
    {
        B obj_B = new B();
        obj_B.print_A();
        obj_B.print_B();

        C obj_C = new C();
        obj_C.print_A();
        obj_C.print_C();

        D obj_D = new D();
        obj_D.print_A();
        obj_D.print_D();
    }
}
```

Output

```
Class A

Class B

Class A

Class C

Class A

Class D
```

Exercise---

Create Student class having student name and roll number, Test having marks of subject 1&2 and Result class will compute the average of marks and display all the information using multilevel inheritance. Write a program taking relevant data.

Method Overriding

- In a class hierarchy, when a method in a subclass has the **same name and type signature** as a method in its superclass, then the method in the subclass is said to override the method in the superclass.
- When an overridden method is called from within its subclass, it will always refer to the version of that method defined by the subclass.
- The version of the method defined by the superclass will be hidden.

```
// Method overriding.
class A {
int i, j;
A(int a, int b) {
i = a;
j = b;
}
// display i and j
void show() {
System.out.println("i and j: " + i + " " + j);
}
}

class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
// display k – this overrides show() in A
void show() {
System.out.println("k: " + k);
}
}
class Override {
public static void main(String args[]) {
B subOb = new B(1, 2, 3);
subOb.show(); // this calls show() in B
}
}
```

The output produced by this program is shown here:

```
k: 3
```

```
class B extends A {
int k;
B(int a, int b, int c) {
super(a, b);
k = c;
}
void show() {
super.show(); // this calls A's show()
System.out.println("k: " + k);
}
}
```

If you substitute this version of A into the previous program, you will see the following output:

```
i and j: 1 2
k: 3
```

Here, super.show( ) calls the superclass version of show( ).

Dynamic Method Dispatch