Inheritance

In Java, a class that is inherited is called superclass.

A class, does the inheriting called a subclass.

Subclass is specialized version of superclass.

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

To Inherit a class, you simply incorporate the definition of one class into another by using the extends keyword.

Example:

Creates a superkass called A and a subclass B.

Keyword extend is used to create a subclass of A

```
Class A { // create a superclass
int i,j;
void showij() {
System.out.println("i and j" + i +" "+j);
} }.
class B extends A { // create a subclass by extending 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();
// Superclass may be used by itself
  superOb.i = 10;
  superOb.j = 20;
  System.out.println("Contents of superOb:");
  superOb.showij();
  System.out.println();                    10 20
// 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();                          7 8
  subOb.showk();                           9
  System.out.println();
  System.out.println("Sum of i, j & k in subOb:");
     subOb.sum();
  }}
```

O/P

Contents of superOb:
i and j: 10 20

Contents of subOb:
i and j : 7 8
k : 9
Sum of i, j & k in subOb: i+j+k : 24

Java does not support the inheritance of multiple superclasses into single subclass.

You can create a hierarchy of inheritance in which a subclass becomes a superclass of another subclass.

Member Access and Inheritance.

We know a subclass can access all the members of superclass, it can't access those members of superclass which are declared private. Consider following example:

```
class A {    // create superclass
int i;       // By default public
private int j; // private.
void setij (int x, int y) { // Parameters
                               Declared.
     i = x
     j = y
} }
class B extends A {
int total;
void sum() { total = i+j; } } // j is not accessibe.
                                 (Error).
 class Access {
public static void main(string args()) {
    B subOb = new B();
```

```
        subOb.setij(10,20);
        subOb.sum();
      System.out.println("Total is "+subOb.total);
      }
      }
```

This program will not compile because Reference to j inside the sum() method of B causes an access violation. Since j is declared as Private it is only accessible by other members of its own class.
Subclass has no access to it.

★ A superclass variable can Reference a Subclass Object ★

Reference variable of a superclass can be assigned a reference to any subclass derived from that superclass. 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 Ref to Box Ref
plainbox = weightbox;
vol = plainbox.volume();
System.out.println("Vol of plainbox "+vol);
// following stmt is invalid b'cs plainbox doesn't
   define a weight number.
System.out.println("Weight of plainbox is"
                   + plainbox.weight);
}
}
```

Above module of Program can be seen in context of following program

```
// This program uses inheritance to extend Box
class Box {
double ~~weight~~ width;
double height;
double depth;
// Construct a 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 dimension 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("Vol of mybox1 is "+vol);
 System.out.println("Weight of mybox1 is "+mybox1.weight);
 System.out.println();

 vol = mybox2.volume();
System.out.println("Vol of mybox2 is "+vol);
System.out.println("Weight of mybox2 is "+mybox2.weight);
}}
```

O/P

```
Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3
Volume of mybox2 is 24.0
Weight of mybox2 is 0.076
```

BoxWeight inherits all the characteristics of Box and adds to them the weight component.
It is not necessary for BoxWeight to re-create all of the features found in Box.
It can simply extends Box to meet up its own purposes.
A major advantage of inheritance is that once you have created a superclass that defines attributes common to set of objects, it can be used to create any number of more specific subclasses.
Each subclass can precisely tailor its own classification. For Ex:
A class inherits Box and adds a color attribute.

```
class ColorBox extends Box {
    int color;
    ColorBox( double w, double h, double d, ~~double~~ int c)
    {
        width = w;
        height = h;
        Depth = d;
        color = c;
    }
}
```

Using Super

Many times we want to create a class (super) that keeps the details of its implementation to itself (means, keep its data member private) In this case there would be, no way for subclass to directly access or initialize these variables on its own.

Since Encapsulation is a primary attribute/feature of OOP, it is not surprising that Java provides a solution to this problem.

Whenever a subclass needs to refer to its immediate superclass, it can use **super** keyword

Super has 2 General form.

- First calls superclass' constructor
- Second use 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 defined by superclass by the use of following form of super.

```
super (arg-list);
```

— Any Arg needed by constructor in superclass

super() must be always a first statement executed in a subclass' constructor.

How super() is used ?

```
class BoxWeight extends Box {
  double weight; // weight of Box.
  BoxWeight (double w, double h, double d, double m)
  {
  super (w, h, d); // call super class Constructor
  weight = m;
  }
}
```

First use

→ Here BoxWeight calls super() with arguments w, h and d.

This causes Box() constructor to be called. which initializes weight, height and depth BoxWeight no longer initializes these values itself. It only needs to initialize the value unique to it : weight.

Second use of super

Second form of super acts somewhat like this, except that it always refer to the superclass of the subclass in which it is used.

General form ⟹ super.member

member can be method or an instance variable.

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

Consider Simple class hierarchy

```
// using super to overcome name hiding
class A {
  int i;
}
// creates a subclass by extending A.
class B extends A {
  int i;  // this i hides 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();
      }
      }.
```

O/P

i in superclass : 1 -

i in subclass : 2. -

Although the instance variable i in B hides i in A, super allows access to i defined in superclass.

super() is used to call methods that are hidden by a subclass.

Method Overriding

In a class hierarchy when a method in a subclass has the name (same) as in method of superclass. then method in subclass. said to be override the method in superclass. When a overridden method is called from within a 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.

Consider Example :-

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

```
// display k - this overrides show() in A
void show() {
System.out.println("k:"+k);
}}

Class OverRide {}

public static void main(String args[]){
    A a1 = new A();
    B subOb = new B(1,2,3);

subOb.show(); // this calls show() in B
}
}
```

O/P

k=3

If you wish to access the superclass version of an overriden method you can use super.

```
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);
}
}
```

Output

i and j : 1 and 2
k : 3

Why Overridden Methods ?

Overridden method allows Java to support Run-time polymorphism.

Polymorphism : Allows a General class to specify methods that will be common to all of its derivatives while allowing subclasses to define specific implementation of some or all of those methods.

Overridden methods are another way that Java implements the "One interface, multiple methods" aspect of polymorphism.

By combining inheritance with Overridden methods, a superclass can define the general form of the methods that will be used by all of its subclasses.

Dynamic Run time polymorphism brings to bear on code reuse and robustness.

Using final with Inheritance :

Final has 3 uses — used to create the equivalent of a named constant

Inheritance { Use final to Prevent Overriding
" " " " Inheritance

Using final to Prevent Overriding

While Overriding is Java's most powerful feature, but there will be times when u will want to prevent it from occurring

To Disallow a method ~~there will be~~ being overridden. specify final as a modifier at a start of its Declaration

Methods Declared as final cannot be overridden

Ex:

```
class A {
final void meth () {
System.out.println ("This is final method");
}}
class B extends A {
void meth () // Error Can't Override
{ System.out.println ("Illegal");
}
}
```

Using final to prevent Inheritance :

Sometimes you will want to prevent a class from being inherited.

```
final Class A {.
    ===
    ===
    ===
  }.
class B extends A
  {
    ===    // Error.
  }
```