MODULE-3 contd..

Chapter 2: INHERITANCE

Inheritance:

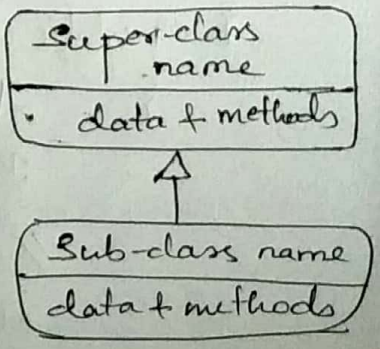
- It is one of the feature of OOP.
- It is the mechanism in java by which one class is allowed to inherit the features (properties & behavior) of another class.

Terminology:

- Super class: The class whose features are inherited is known as super class (or base class or parent class).
- Sub class: The class that inherits the other class is known as sub class. (or derived class / child class).
- Reusability: is a mechanism which facilitates to reuse the fields and methods of existing class when you create a new class. You can use the same fields (properties) & methods (behavior) already defined in the previous class.

Syntax of Java Inheritance

```
class Superclass-name
{   // data & methods
}
class Subclass-name extends Superclass-name
{   // data & methods
}
```



here, extends keyword indicates that you are making a new class that derives from an existing class. & meaning of "extends" is to increase the functionality.

* In Java, a class which is inherited is called a parent (or) super class, & the new class is called child or subclass.

* Let us consider a simple Example for demonstrating Inheritance concept: The program creates a superclass called A & a subclass B.

```
class A
{   int i, j;
    void showij()
    {  System.out.println(" i & j:" +i+ "\t" +j);
    }
}
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 ob = new A();
      B subob = new B();
      //superclass may be used itself.
      ob.i = 10;
      ob.j = 20;
      System.out.println("Contents of Superclass A");
        ob.showij();
      /*Subclass has access to all public members of
        the superclass */
      subob.i = 3;
      subob.j = 4;
      subob.k = 5;
```

```
        System.out.println("Contents of Sub-class B");
          subob.showij();
          subob.showk();
        System.out.println();
        System.out.println("Sum of i,j,k in Subob");
          subob.sum();
    } //End of main()
} //End of class
```

Output : Contents of Super class A
i and j : 10 20

Contents of Sub-class B
i and j : 3 4
k : 5

Sum of i, j, k in Subob
i+j+k : 12.

* Note: In the above program, the subclass B includes all of the members of its superclass A.

Member Access and Inheritance:

In Java, we have three access specifier : Public, Protected and Private.

* Public : When the access specifier for a base class is public during inheritance, all public members of base become public members of the derived class.
* In base class if the members are protected then it becomes protected for derived class.
* In base class if the members are private elements, it remain private for base & not accessible by members of derived class.

Example : The following program illustrates public access specifier.

```
class Base
{   public int x,y;
    Protected int a, b;
    private int p, q;
    void set()
    {   x = 2; y = 3;
        a = 4; b = 5;
        p = 6; q = 7;
    }
    void show()
    {   System.out.println("Values of a,b,x,y,p,q");
        System.out.println("A:"+a+"\t"+"B:"+b+"\t"
                +"x:"+x+"\t"+"y:"+y+"\t"+"P:"+p+"\t"
                +"Q:"+q);
    }
}
class derived extends Base
{   int z;
    void show()
    {   z=10;
        S.O.p("Value of z:"+z);
        S.O.p("A:"+a); // Accessible : it is public
        S.O.p("B:"+b); // Accessible : it is public
        S.O.p("X:"+x+"\t"+"Y:"+y); // Accessible it
                                   is public
        S.O.p("P:"+p+"\t"+"Q:"+q); // Error, it is
                                   private.
    }
}
class Main
{   P s v m(String args[])
    {   Base b = new Base();
```

```
        b.set();
        b.show();
    // derived class object
        derived d = new derived();
            d.show();
    } //close main
} //End of class.
```

Protected Access specifier: when the members of the class is protected then, it members of base can be accessible in derived as shown in previous example. In other words, protected members in base class can be accessed in its derived class.

Private Access Specifier: when the members of any class is private, then it is accessible only in its own class, not accessible inside its derived class. In other words, private members can be accessed only in its own class as shown in the previous example.

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.

Example:

```
class Base
{
    void Msg()
    { System.out.print("Hello");
    }
}
class Derived extends Base
{ public void Msg()
    { System.out.println("Hi");
    }
}
```

```
class ReferenceObject
{ psvm(String args[])
    { Base ref = new Base();
      Derived ob = new
                   Derived();
      ref = ob;
      ref.Msg();
    }
}
```

O/P: Hi

* In the previous example Base/Super class variable can refer a subclass object i.e.,
→ ref → is a object reference of Superclass & ob is a sub class variable/object.
→ ob is assigned to super class object i.e.,
ref = obj;
* Now user can access subclass method through ref object of Base class. So the o/p is "Hi"

Using super

* Sometime superclass that is created has to keep the details of its implementation to itself. (ie, data members private). Then there would be no way for a subclass to directly access or initialize these variables on its own.
* Java provides a way to this, whenever a subclass needs to refer to its immediate superclass, it can use 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.

Using super to call Superclass Constructors:

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

super(arg-list);

here, arg-list specifies any arguments needed by the constructor in the superclass. super() constructor must always be the first statement executed inside a subclass' constructor.

* Consider an example

```
class A
{   int x, y;
    A()
    {   x=10;
        y=10;
    }
}
class B extends A
{   super(); // invokes parent class constructor
    void show()
    {   System.out.println("X & Y:");
        System.out.println(x+"\t"+y);
    }
}
class AB
{   public static void main(String args[])
    {   B ob = new B();
        ob.show();
    }
}
```

A Second use for super:

→ The second form of super acts like "this" keyword.
→ super always refer to the superclass of the subclass in which it is used.
→ The general form is

super.member

here, member can be either method or an instance variable.

→ This second form of super is most applicable to situations in which member name of a subclass hide members by the same name in the superclass.
→ Consider the following example.

// Using super to overcome name hiding

```
class A
{   int i;
}
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()
   { S.O.p("i in superclass:"+super.i);
     S.O.p("i in subclass:"+i);
   }
}
class Use Super
{
   P S V m(String args[])
     {
       B ob = new B(1,2);
         ob.show();
     }
}
```

O/P: i in superclass: 1
i in subclass: 2

* In the above example 'i' is variable defined in both base class & derived. But i variable when declared in derived is going to hides 'i' in Base class. And Base 'i' is accessible thro' Super keyword as super.i.

// using super to overcome method hiding

```
class Person
{
   void msg()
    { S.O.p("This is
      person class");
    }
}
class Ram extends Person
{  void msg()
    { S.O.p("This is
      Ram/derived class");
    }
   void display()
   { super.msg(); //calls Person msg();
     msg(); //calls derived msg()
   }
}
class Test
{ P S V m(String args[])
   { Ram r = new Ram();
     r.display();
   }
}
```

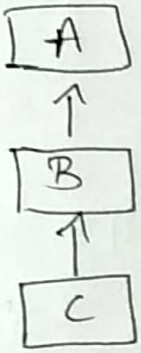
O/P: This is person class
This is Ram/derived class

*In the above example msg() method is defined both in Base & derived. But the base class msg() can be invoked only thro' Super keyword as super.msg(). & derived msg() is called normally.

Creating a Multilevel Hierarchy.

Multilevel inheritance refers to a mechanism in OOP where one can inherit from a derived class, thereby making this derived class the base class for the new class.

Example: If 'A' is superclass 'B' then B can be a base class for C.



* here each subclass inherits all of the features found in all of its superclass.
* here, C inherits all aspects of B+A.

fig: Multilevel inheritance.

Program to illustrate multilevel inheritance:

```
import java.util.*;
import java.lang.*;
import java.io.*;
class A
{   public void print_a()
      {  s.o.p("A base class");
      }
}
class B extends A
{   public void print_b()
      {  s.o.p("B's derived class");
      }
}
class C extends B
{   public void print_c()
      {  s.o.p("C's derived class");
      }
}
```

```
class ABC
{   p s vm (String args[])
    {
        C ob = new C();
        ob.print_a();
        ob.print_b();
        ob.print_c();
    }
}
```

o/p:

```
A base class
B's derived class
C's derived class
```