8.5 Method Overriding

If the method name and signature (number of arguments, arguments types, arguments sequence and return type) in both the base and derived classes are same then the derived class method is said to **override** the method in the base class.

When an overridden method is called by using object of derived class, it will always refer to the version of that method defined by the derived class. The version of the method defined by the base class will be hidden. Therefore, method overriding must be avoided while inheriting a class.

Usage of Method Overriding :

(i) Method overriding is used to provide specific implementation of a method that is already provided by its base or super class.

(ii) Method overriding is used for runtime polymorphism.

Example 8.6
Following program illustrate the concept of method overriding.

```
class base
{   void area ( double r )
    {       double circle = 3.14 * r * r;
            System.out.println ("Area of circle is" + circle);

    }
}
class derived extends base
{   void area (double r)      // overriding base class method
    {       double square = r * r;
            System.out.println ("Area of Square " + square);
    }
}
class override
{
    public static void main (String args[])
    {
            derived obj = new derived( );
            obj.area (2.5);
    }
}
```

Output :
Area of Square 6.25

In this example, derived class object always invoke derived class method and always we get the area of square because base class method **area( )** is overriden in derived class.

Note :

We cannot override **static** method because static method is bound with class whereas instance method is bound with object.

8.5.1 Overloading VS Overriding

Some important differences between overloading and overriding in Java are :

(i) In case of method overloading in Java, **signature** of method changes while in case of method overriding it remain same.

(ii) You can overload method in one class but overriding can only be done on subclass.

(iii) You cannot override **static, final** and **private** method in Java but you can overload static, final or private method in Java.

(iv) Overloading happens at **compile-time** while Overriding happens at **runtime**.

(v) Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.

(vi) Return type of method does not matter in case of method overloading, it can be same or different. However in case of method overriding the return type must be same.

8.6 Super Keyword (solution of overridding)

The **super** keyword is used to access the members of the super class (or base class). There are two purposes of super keyword.

(a) Accessing the superclass hidden member in the subclass (or derived class).

(b) To call superclass constructor in the subclass.

(a) Accessing the Superclass hidden (or overridden) member in the subclass

Here hidden members may be **instance variable** or a **method** of superclass.

(i) We can access the super class instance variables inside the subclass by using the following **syntax.**

super.variable ;

where **super** is the keyword and
variable is the hidden variable of superclass.

Example 8.7

Study the following program

```
class vehicle_twowheeler    // superclass
{
 int speed=60;
}
```

```
class bike extends vehicle_twowheeler // subclass

{
 int speed = 80;
 void display( )

  {
    System.out.println("Two wheelervehicle speed is " + super.speed);
    // use of super keyword
    System.out.println("Bike speed is " + speed);
  }
}

class demosuper

{
 public static void main(String args[])

  {
    bike obj = new bike( );
    obj.display( );
  }
}
```

Output :
Two wheeler vehicle speed is 60
Bike speed is 80

(ii) Similarly, we can invoke or called the **overridden method** inside subclass methods through the use of **super** keyword.

Syntax is :

super.methodname (arguments list);

Where

super is the keyword and

methodname( ) is the overridden method

Example 8.8

Following program illustrate the concept of method overriding.

```
class base

{
   void area ( double r )

   {
      double circle = 3.14 * r * r;
      System.out.println("Area of circle is " + circle);
   }
}

class derived extends base

{
   void area (double r)              // overriding base class method

   {
      super.area(r);    // accessing base class overridden method
      double square= r * r;
      System.out.println("Area of square is " + square);
   }
}

class override

{
   public static void main(String args[])

   {
      derived obj = new derived( );
      obj.area (2.5);
   }
}
```

Output :

Area of circle is 19.625
Area of square is 6.25

(b) **To call superclass constructor in the subclass**

A subclass constructor is used to initialize, the instance variables of both the subclass and the superclass. The subclass constructor uses the **super** keyword to invoke the constructor of the superclass.

Example 8.9

Study the following program

```
class Roomarea          // Superclass

{
    int length;
    int breadth;
    Roomarea(int x, int y)

    {
        length = x;
        breadth = y;
    }

    int area( )

    {
        return (length * breadth);
    }
}

class Roomvolume extends Roomarea          // Subclass

{
    int height;
    Roomvolume(int x, int y, int z)

    {
        super(x, y);                       // pass values to superclass
        height = z;
    }

    int volume( )
    {
        return (length * breadth * height);
    }
}
```

```
class demosuper

{
    public static void main (String args[])

    {
        Roomvolume rv = new Roomvolume (10, 15, 20);
        int area1 = rv.area( );              // superclass method
        int volume1 = rv.volume ( );                  // subclass method
        System.out.println("Area = " + area1);
        System.out.println("Volume = " + volume1);
    }
}
```

Output :

Area = 150

Volume = 3000