Java

## 3.2 Inheritance and Constructors

⇒ In java, when a subclass / child class is instantiated, the constructor of the superclass is called first. If no explicit call to the superclass constructor is made, the default constructor of the superclass is automatically invoked.

When an object is created of the derived class, it will call the both of the constructor method from superclass as well as child class. but always call the constructor of the parent class first.

Example:-

```
class Base
{
  Base()
  {System.out.println("Base class Constructor called.");
  }
}
class Derived extends Base
{Derived()
  {System.out.println("Derived class Constructor called.");
}}
public class inherit{
   public static void main(String[]args){
    Derived d = new Derived();
}}
```

output:
Base class Constructor called.
Derived class constructor called

In this example, creating an instance of 'Derived' results in the 'Base' constructor being called first, followed by the 'Derived' constructor.

## 3.3 'Super' keyword

The 'super' keyword in java is used within a subclass to refer to its immediate superclass. Super keyword can be used to call

1. Superclass methods
2. Access superclass fields, and
3. Call superclass constructors.

### 1. Superclass methods

```
Class Base{
  void B()
  {
    System.out.println("Base class method called.");
  }
}
class Derived extends Base{
  void B()
  {
    System.out.println("Derived class method called.");
  }
  void display()
  {
    super.B(); // call the superclass method
    B();
  }
}
public class inherit{
  Derived d = new Derived();
  d.display();
}
```

output:

Base class method called.
Derived class method called.

② Accessing Superclass Constructors

```
class Animal{
    Animal()
    {
      System.out.println("Animal constructor called");
    }
}
class Dog extends Animal
{
  Dog()
  {
    super(); // call the superclass constructor
    System.out.println("Dog constructor called");
  }
}
public class Inherit{
  public static void main(String[] args)
  {
    Dog dog = new dog();
  }
}
```

output:
Animal Constructor called
Dog Constructor called

## ③. Accessing Superclass fields

```
Class A
{
  String name = "Rajesh parajuli";
}
Class B extend A
{
  String name = "Rajesh";
  Void Display_Names()
  {
  System.out.println("class A Name:  " + super.name);
  System.out.println("class B Name: " + name);
  }
  public class inherit{
  public static void main(String[] args)
  {
     B bb = new B();
     bb.Display_Names();
  }
  }
```

output:

```
Class A Name: Rajesh parajuli
Class B Name: Rajesh
```

## 34 Method overriding :- Member function overriding

The process of redefining the inherited method of the super class in the derived class is called method overriding. The return type and the parameters must be same in both the super class and the subclass. The '@override' annotation is often used to indicate that a method is being overridden.

The overridden method in the subclass should have same access modifier or less restrictive than that of super class. for example, if the overridden method in super class is protected, then it must be either protected or public in the subclass but not private.

example

```
class A {
String name = "class A";
void display() {
 System.out.println("Display from "+name);
}
}
class B extends A {
 String name = "class B";
 @Override
  void display()
  {
   System.out.println("Display from "+name);
  }
  void displayBoth()
  {
   System.out.println("Superclass name:"+super.name);
   System.out.println("Subclass name:"+name);
  }
}
```

```
public class Inherit{
  public static void main(String args[])
  {
      B b = new B();
      A a = new A();

      a.display(); // Display from class A
      b.display(); // Display from class B
      b.displayBoth(); // Displays from Both class
  }
```

output:

```
Display from class A
Display from class B
Superclass name: class A
Subclass name: class B
```