What is Inheritance in Java?

Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes.

A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

The keyword "**extends**" is used to derive a subclass from the superclass.

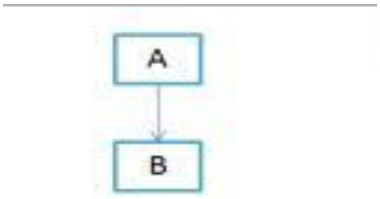
Syntax:-

```
class Super
{
   .....
   .....
}
class Sub extends Super
{
   .....
   .....
}
```

Types of inheritance in Java

1) Single Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



(a) Single Inheritance

```
class A
{
   public void methodA()
   {
     System.out.println("Base class method");
   }
}
class B extends A
{
   public void methodB()
   {
     System.out.println("Child class method");
   }
   public static void main(String args[])
   {
     B obj = new B();
     obj.methodA(); //calling super class method
     obj.methodB(); //calling local method
  }
}
```

Output:

```
Base class method
Child class method
```

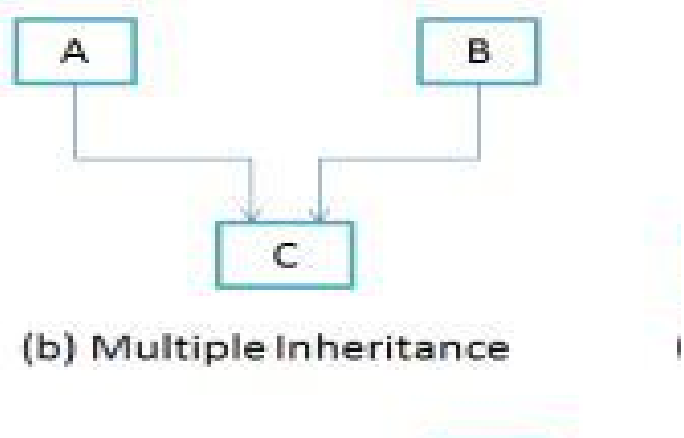
```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

```
class Teacher {
   private String collegeName = "IBSAR";
   private String designation = "Teacher";

   public String getDesignation()
   {
       return designation;
   }
   protected void setDesignation(String designation)
   {
       this.designation = designation;
   }
   protected String getCollegeName()
   {
       return collegeName;
   }
   protected void setCollegeName(String collegeName)
   {
       this.collegeName = collegeName;
   }
   public void does()
   {
       System.out.println("Teaching");
   }
}
```

```
public class JavaProgarm extends Teacher
{
   String mainSubject = "ASP.NET";
   public static void main(String args[])
   {
       JavaProgarm obj = new JavaProgarm();
       System.out.println("College Name:
"+obj.getCollegeName());
       System.out.println("Designation:
"+obj.getDesignation());
       System.out.println("Main Subject "+obj.mainSubject);
       obj.does();

   }
}
```

2) Multiple Inheritance

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



(b) Multiple Inheritance

Note [Multiple inheritance is not supported in Java, C#, Smalltalk etc. Multiple Inheritance is Supported only in C++, **"if we want to do Multiple Inheritance, then we have to use interface, with the help of Interface We can perform Multiple Inheritance in Java".**

Example:-

```
interface X
{
   public void myMethod();
}
interface Y
{
   public void myMethod();
}
class JavaExample implements X, Y
{
   public void myMethod()
   {
      System.out.println("Implementing more than one interfaces");
   }
   public static void main(String args[]){
         JavaExample obj = new JavaExample();
         obj.myMethod();
   }
}
```

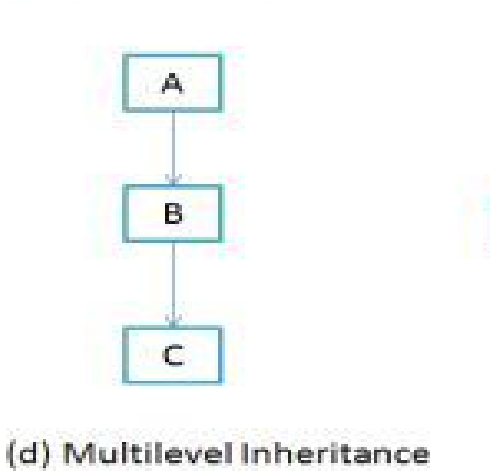
Output:

Implementing more than one interfaces

Note [As you can see that the class implemented two interfaces. A class can implement any number of interfaces.**]**

3) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.



(d) Multilevel Inheritance

```
class X
{
   public void methodX()
   {
     System.out.println("Class X method Called");
   }
}
```

```
class Y extends X
{
public void methodY()
{
System.out.println("Class Y method Called ");
}
}
class Z extends Y
{
   public void methodZ()
   {
     System.out.println("Class Z method Called ");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

Output:

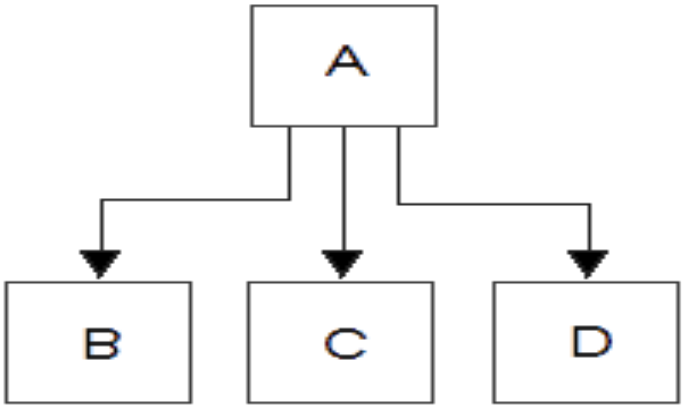
```
Class X method Called
Class Y method Called
Class Z method Called
```

```
class Animal
{
       void eat()
      {
      System.out.println("eating...");
     }
}
class Dog extends Animal
{
       void bark()
      {
     System.out.println("barking...");
```

```
    }
}
    class BabyDog extends Dog
    {
    void weep()
    {
    System.out.println("weeping...");
     }
}
    class TestInheritance2
   {
    public static void main(String args[])
    {
    BabyDog d=new BabyDog();
    d.weep();
    d.bark();
    d.eat();
    }
}
```

Hierarchical Inheritance in java

When more than one classes inherit a same class then this is called hierarchical inheritance. For example class B, C and D extends a same class A.



As you can see in the above diagram that when a class has more than one child classes (sub classes) or in other words more than one child classes have the same parent class then this type of inheritance is known as **hierarchical inheritance**.

```
class A
{
   public void methodA()
   {
      System.out.println("method of Class A");
   }
}
class B extends A
{
   public void methodB()
   {
      System.out.println("method of Class B");
   }
}
class C extends A
{
  public void methodC()
  {
     System.out.println("method of Class C");
  }
}
class D extends A
{
  public void methodD()
```

```
  {
     System.out.println("method of Class D");
  }
}
class JavaExample
{
  public static void main(String args[])
  {
     B obj1 = new B();
     C obj2 = new C();
     D obj3 = new D();
     //All classes can access the method of class
A
     obj1.methodA();
     obj2.methodA();
     obj3.methodA();
  }
}
```

Output:

```
method of Class A
method of Class A
method of Class A
```