Inheritance in Java

Inheritance is one of the key features of **OOP** that allows us to create a **new class** from an **existing class**.

Inheritance is the process of creating new class from an existing class. The **new class** is often called as (**child/sub/derived class**) and an **existing class** is often called as (**parent/base/super class**).

Inheritance is one of the important mechanism in java by which one class inherits the features(field/method) of another class.

Inheritance represents the **IS-A** relationship which is also known as a parent-child relationship.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

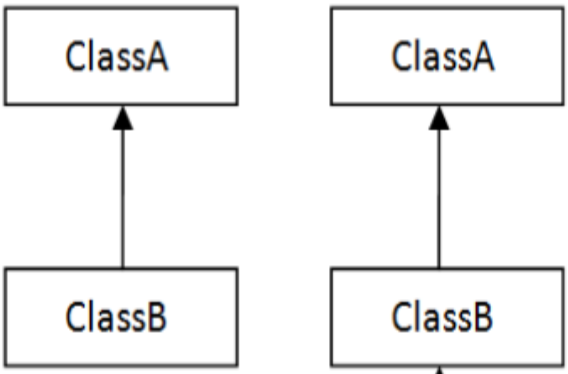
The **extends** keyword indicates that you are making a new class that derives from an existing class. The meaning of **"extends"** is to increase the functionality.

Types of inheritance in java

On the basis of class, there can be three types of inheritance in java:

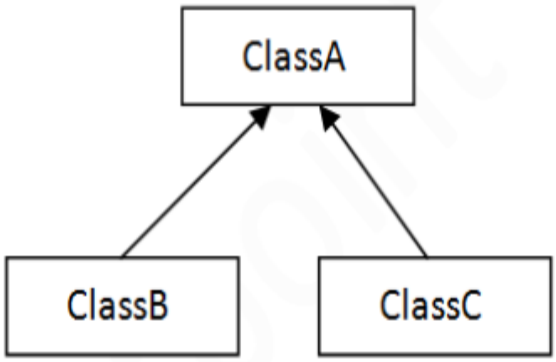
- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance

In java programming, multiple and hybrid inheritance is not supported through class but it can be supported/achieved through interface.

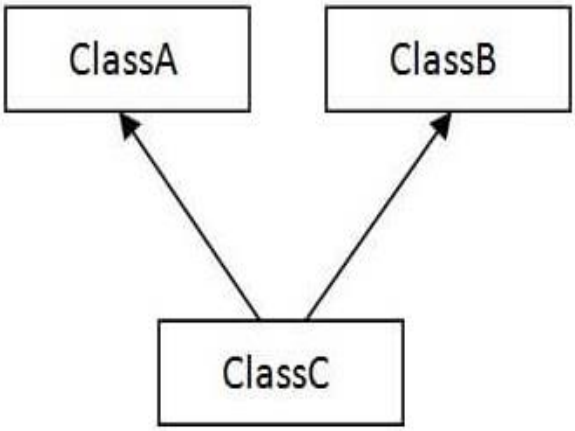


1) Single

2) Multilevel

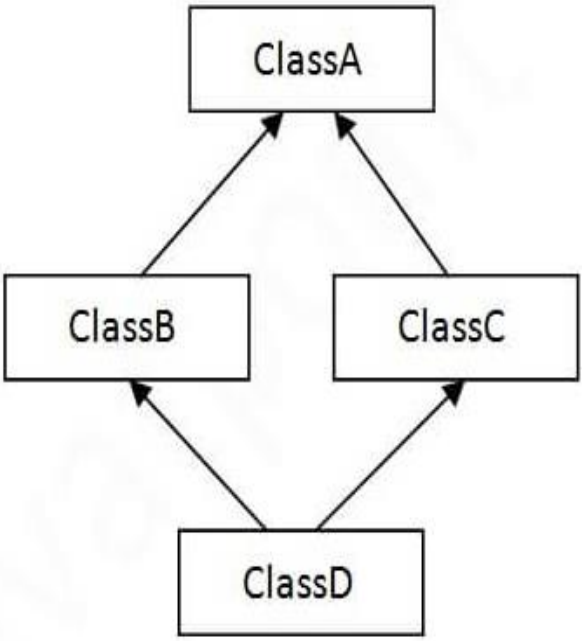


3) Hierarchical



4) Multiple

Not supported through class



5) Hybrid

Not Supported through class

Single Inheritance Example

When a class inherits another class, it is known as a single inheritance.

```
class Animal{
    void eat()
    {
      System.out.println("eating...");
    }
}
class Dog extends Animal{
      void bark()
      {
        System.out.println("barking...");
      }
}
class TestInheritance{
      public static void main(String args[]){
          Dog d=new Dog();
          d.bark();
          d.eat();
      }
}
```

Multilevel Inheritance

When there is a chain of inheritance, it is known as multilevel inheritance.

```
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

When two or more classes inherits a single class, it is known as hierarchical inheritance.

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

```
class Cat extends Animal{
        void meow()
        {
          System.out.println("meowing...");
        }
}
class TestInheritance3{
      public static void main(String args[]){
          Cat c=new Cat();
          c.meow();
          c.eat();
         //c.bark();//C.T.Error
      }
}
```

Q) Why multiple inheritance is not supported in java?

To reduce the **complexity** and **simplify** the language, multiple inheritance is not supported in java.

Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be **ambiguity** to call the method of A or B class.

```
class A{
    void msg()
    {
      System.out.println("Hello");
    }
}
class B{
     void msg()
     {
       System.out.println("Welcome");
     }
}
```

```
class C extends A,B{       //suppose if it were
   public static void main(String args[]){
      C obj=new C();
      obj.msg();       //Now which msg() method would be invoked?
   }
}
```

Polymorphism in Java

Polymorphism is derived from two Greek words **poly** and **morphs**.

poly => which means **"many"**

morphs => which means **"forms"**.

So as a collectively we can say polymorphism means many forms.

Polymorphism in Java is a concept by which we can perform a single action in different ways.

There are two types of polymorphism in java

- Compile time/ Static binding/ Early binding Polymorphism
- Runtime/ Dynamic binding/ Late binding Polymorphism

Compile time/ Static binding/ Early binding Polymorphism

Compile-time polymorphism is a polymorphism that is resolved during the compilation process. Overloading of methods is called through the reference variable of a class. Compile-time polymorphism is achieved by method overloading overloading.

Eg:- **method overloading**

Method overloading means If the same method is present in the both parent and child class but only different in its signature, we called it as method overloading.

Note: *Method overloading can be achieved by Inheritance and normal method.*

```
void show () {...}
void show (int num1) {...}
void show (float num1) {...}
void show (int num1, float num2) {...}
```

By Inheritance

```
class M
{
  public void mul (int x, int y)
  {
    int c1=x*y;
    System.out.println("Sum is:"+ c1);
  }
}
 class M1 extends M
 {
  public void mul (int x, float y)
  {
    float c2=x*y;
    System.out.println("Sum is:"+ c2);
  }
 }
 class Polymorphism1
 {
  public static void main (String args[])
  {
      M1 m = new M1();
           m.mul (50,60);
           m.mul (25,20.5f);
  }
}
```

By Normal Method

```
class M
{
  public void mul (int x, int y)
  {
    int c1=x*y;
    System.out.println("Sum is:"+ c1);
  }
  public void mul (int x, float y)
  {
    float c2=x*y;
    System.out.println("Sum is:"+ c2);
  }
}
```

```
class Polymorphism1
 {
   public static void main (String args [])
   {
       M1 m = new M1();
             m.mul (50,60);
             m.mul (25,20.5f);
   }
}
```

Area of different shape by using method overloading

Volume of different shape using method overloading

```
class Shape
{
  int base, height;
  float A;
   public void area ()
   {
     A=base*height/2;
     System.out.println("Area of Triangle: " + A);
   }
   public void area (int l, int b)
   {
     A=l*b;
     System.out.println("Area of Rectangle is: " + A);
   }
   public void area (int a)
   {
     A=a*a;
     System.out.println("Area of Square is: " + A);
   }
}
public class Area1
{
   public static void main (String args [])
   {
     Shape s=new Shape ();
           s. base=10;
```

```
        s. height=2;
        s. area ();

        s. area (4);
        s. area (10, 20);
  }
}
```

Runtime/ Dynamic binding/ Late binding Polymorphism

Runtime polymorphism or **Dynamic Method Dispatch** is a polymorphism in which a call to a method is resolved at runtime rather than compile-time.

Eg: - **method overriding**

If **sub class** (child class) has the same method as declared in the **parent class**, it is known as **method overriding** in Java.

In other words, if a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as **method overriding.**

Note: *Method overridden is only achieved by inheritance.*

Why we use override?

When a child class is not satisfied with the implementation of its parent class then the child class can implement the parent class methods according to their need. And this is and that is why we use override method. (Eg property, marriage)

Rules for Java Method Overriding

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.

```
class Vehicle {
  void run () {
    System.out.println("Vehicle is running");
  }
}
class Bike2 extends Vehicle {
  void run () {
```

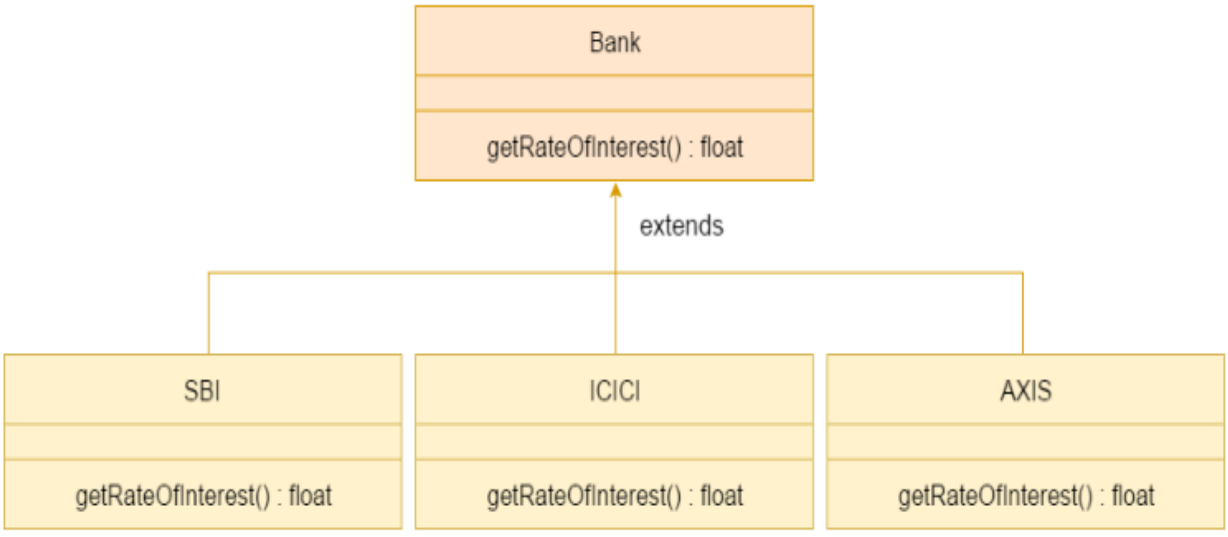
```
    System.out.println("Bike is running safely");
  }
  public static void main (String args []) {
  Bike2 obj = new Bike2();
  obj.run ();
  }
}
```

If we run this program child class run () method override the parent class run () method and child class run method execute with child class object.

If we want to access parent class run () method also then we call it by parent class object.

A real example of Java Method Overriding

Consider a scenario where Bank is a class that provides functionality to get the rate of interest. However, the rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7%, and 9% rate of interest.



Can we override static method?

No, a static method cannot be overridden.

Why can we not override static method?

It is because the static method is bound with class whereas instance method is bound with an object. Static belongs to the class area, and an instance belongs to the heap area.

Can we override java main method?

No, because the main is a static method.

How to prevent method overridden in Java?

Inheritance is a substantial rule of any Object-Oriented Programming (OOP) language but still, there are ways to prevent method overriding in child classes which are as follows:

3 Ways to Prevent Method Overriding in Java

- Using a **static** method.
- Using **private** access modifier.
- Using the **final** keyword method.

Super Keyword in Java

The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- Super () can be used to invoke immediate parent class constructor.

super can be used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

```
class Animal {
String color="white";
}
class Dog extends Animal {
   String color="black";
   void printColor () {
       System.out.println(color);
       System.out.println(super. color);
   }
}
class TestSuper1{
   public static void main(String args[]){
        Dog d=new Dog();
        d.printColor ();
    }
}
```

super can be used to invoke immediate parent class method.

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

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

```
  }
}
class TestSuper2{
    public static void main(String args[]){
        Dog d=new Dog();
         d.eat();
          d.bark();
     }
}
```

Super () can be used to invoke immediate parent class constructor.

The super keyword can also be used to invoke the parent class constructor. Super () must be the first statement in the child class constructor.

```
class Animal{
    Animal(){System.out.println("animal is created");}
}
class Dog extends Animal
 {
    Dog()
    {
      super();
      System.out.println("dog is created");
    }
}
class TestSuper3{
    public static void main(String args[]){
       Dog d=new Dog();
    }
}
```

Final Keyword in Java

Final Keyword is used in three scenarios in java.

- Variable
- Method
- Class

If you make any variable as final, you cannot change the value of final variable (It will be constant).

If you make any method as final, you cannot override it.

If you make any class as final, you cannot extend it.