

# Inheritance in Java

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

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.

## Terms used in Inheritance

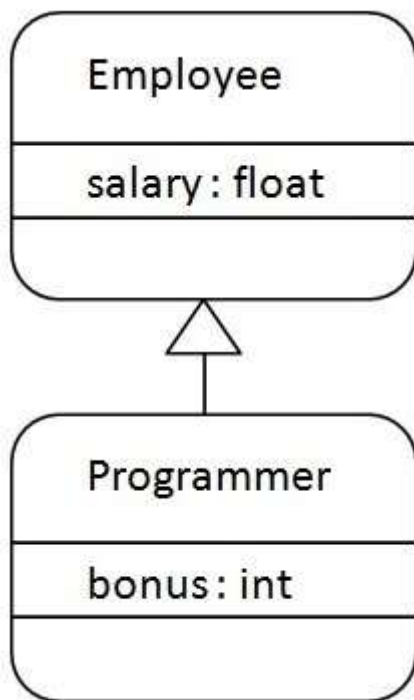
- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

## The syntax of Java Inheritance

1. **class** Subclass-name **extends** Superclass-name
2. {
3.   //methods and fields
4. }

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.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.



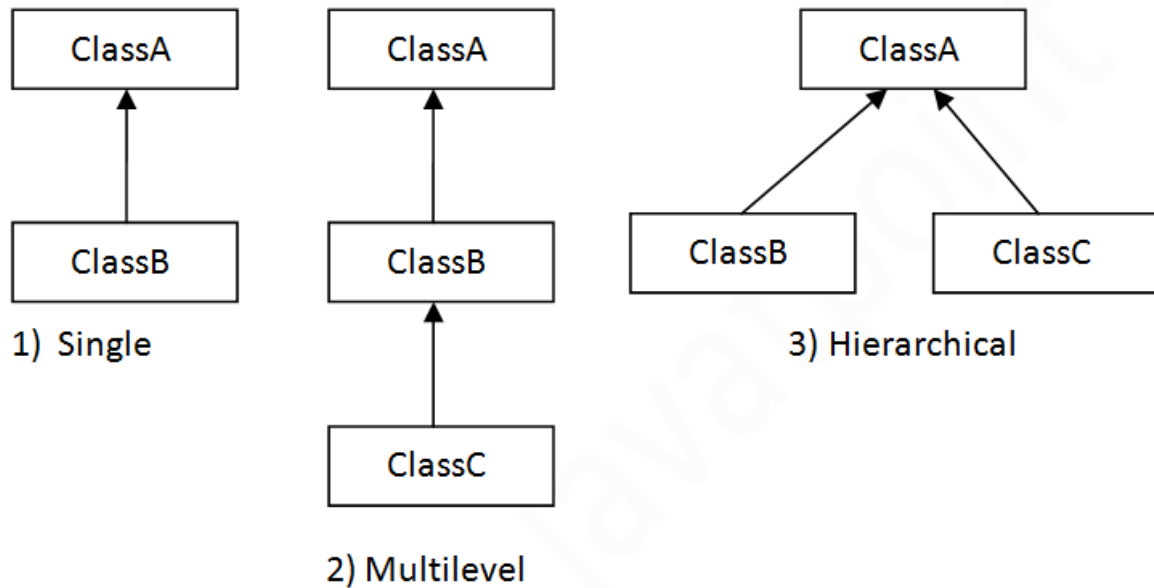
As displayed in the above figure, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is **Programmer IS-A Employee**. It means that Programmer is a type of Employee.

```
1. class Employee{
2.     float salary=40000;
3. }
4. class Programmer extends Employee{
5.     int bonus=10000;
6.     public static void main(String args[]){
7.         Programmer p=new Programmer();
8.         System.out.println("Programmer salary is:"+p.salary);
9.         System.out.println("Bonus of Programmer is:"+p.bonus);
10. }
11. }
```

## Types of inheritance in java

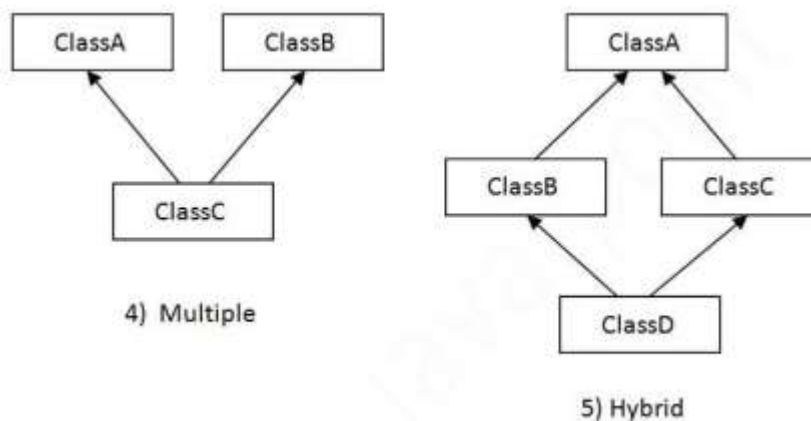
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.

In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



*Note: Multiple inheritance is not supported in Java through class.*

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



## Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class TestInheritance{
8. public static void main(String args[]){
9. Dog d=new Dog();
10. d.bark();
11. d.eat();
12. }}
```

## Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class BabyDog extends Dog{
8. void weep(){System.out.println("weeping...");}
9. }
10. class TestInheritance2{
11. public static void main(String args[]){
12. BabyDog d=new BabyDog();
```

```
13. d.weep();
14. d.bark();
15. d.eat();
16. }}
```

## Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

```
1. class Animal{
2. void eat(){System.out.println("eating...");}
3. }
4. class Dog extends Animal{
5. void bark(){System.out.println("barking...");}
6. }
7. class Cat extends Animal{
8. void meow(){System.out.println("meowing...");}
9. }
10. class TestInheritance3{
11. public static void main(String args[]){
12. Cat c=new Cat();
13. c.meow();
14. c.eat();
15. //c.bark();//C.T.Error
16. }}
```

## 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.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

```
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4. class B{
5. void msg(){System.out.println("Welcome");}
6. }
7. class C extends A,B{//suppose if it were
8.
9. public static void main(String args[]){
10. C obj=new C();
11. obj.msg();//Now which msg() method would be invoked?
12. }
13. }
```

COMPILATION ERROR

# Method Overloading in Java

If a **class** has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the **program**.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as `a(int,int)` for two parameters, and `b(int,int,int)` for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

## Advantage of method overloading

Method overloading *increases the readability of the program*.

## Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

*In Java, Method Overloading is not possible by changing the return type of the method only.*

## 1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first `add()` method performs addition of two numbers and second `add` method performs addition of three numbers.

In this example, we are creating static methods so that we don't need to create instance for calling methods.

1. **class** Adder{
2. **static int** add(**int** a,**int** b){**return** a+b;}
3. **static int** add(**int** a,**int** b,**int** c){**return** a+b+c;}
4. }
5. **class** TestOverloading1{

```

6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(11,11,11));
9. }}

```

## 2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type. The first add method receives two integer arguments and second add method receives two double arguments.

```

1. class Adder{
2. static int add(int a, int b){return a+b;}
3. static double add(double a, double b){return a+b;}
4. }
5. class TestOverloading2{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));
8. System.out.println(Adder.add(12.3,12.6));
9. }}

```

### Q) Why Method Overloading is not possible by changing the return type of method only?

In java, method overloading is not possible by changing the return type of the method only because of ambiguity. Let's see how ambiguity may occur:

```

1. class Adder{
2. static int add(int a,int b){return a+b;}
3. static double add(int a,int b){return a+b;}
4. }
5. class TestOverloading3{
6. public static void main(String[] args){
7. System.out.println(Adder.add(11,11));//ambiguity
8. }}

```



*Note: Compile Time Error is better than Run Time Error. So, java compiler renders compiler time error if you declare the same method having same parameters.*

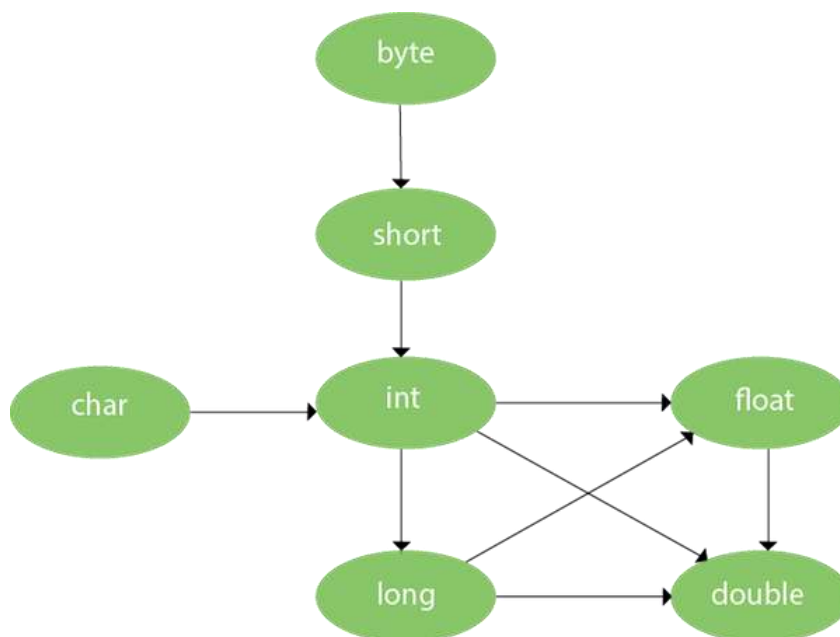
## Can we overload java main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. But [JVM](#) calls main() method which receives string array as arguments only. Let's see the simple example:

1. **class** TestOverloading4{
2. **public static void** main(String[] args){System.out.println("main with String[]");}
3. **public static void** main(String args){System.out.println("main with String");}
4. **public static void** main(){System.out.println("main without args");}
5. }

## Method Overloading and Type Promotion

One type is promoted to another implicitly if no matching datatype is found. Let's understand the concept by the figure given below:



As displayed in the above diagram, byte can be promoted to short, int, long, float or double. The short datatype can be promoted to int, long, float or double. The char datatype can be promoted to int, long, float or double and so on.

## Example of Method Overloading with TypePromotion

```
class OverloadingCalculation1{
    void sum(int a,long b){System.out.println(a+b);}
    void sum(int a,int b,int c){System.out.println(a+b+c);}

    public static void main(String args[]){
        OverloadingCalculation1 obj=new OverloadingCalculation1();
        obj.sum(20,20);//now second int literal will be promoted to long
        obj.sum(20,20,20);
    }
}
```

## Example of Method Overloading with Type Promotion if matching found

If there are matching type arguments in the method, type promotion is not performed.

```
1. class OverloadingCalculation2{
2.     void sum(int a,int b){System.out.println("int arg method invoked");}
3.     void sum(long a,long b){System.out.println("long arg method invoked");}
4.
5.     public static void main(String args[]){
6.         OverloadingCalculation2 obj=new OverloadingCalculation2();
7.         obj.sum(20,20);//now int arg sum() method gets invoked
8.     }
9. }
```

## Example of Method Overloading with Type Promotion in case of ambiguity

If there are no matching type arguments in the method, and each method promotes similar number of arguments, there will be ambiguity.

```
class OverloadingCalculation3{
    void sum(int a,long b){System.out.println("a method invoked");}
    void sum(long a,int b){System.out.println("b method invoked");}

    public static void main(String args[]){
        OverloadingCalculation3 obj=new OverloadingCalculation3();
        obj.sum(20,20);//now ambiguity
    }
}
```

*One type is not de-promoted implicitly for example double cannot be depromoted to any type implicitly.*

## Method Overriding in Java

If subclass (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.

### Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

### Rules for Java Method Overriding

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

3. There must be an IS-A relationship (inheritance).

## Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

```
1. class Vehicle{
2.     void run(){System.out.println("Vehicle is running");}
3. }
4. //Creating a child class
5. class Bike extends Vehicle{
6.     public static void main(String args[]){
7.         //creating an instance of child class
8.         Bike obj = new Bike();
9.         //calling the method with child class instance
10.        obj.run();
11.    }
12.}
```

## Example of method overriding

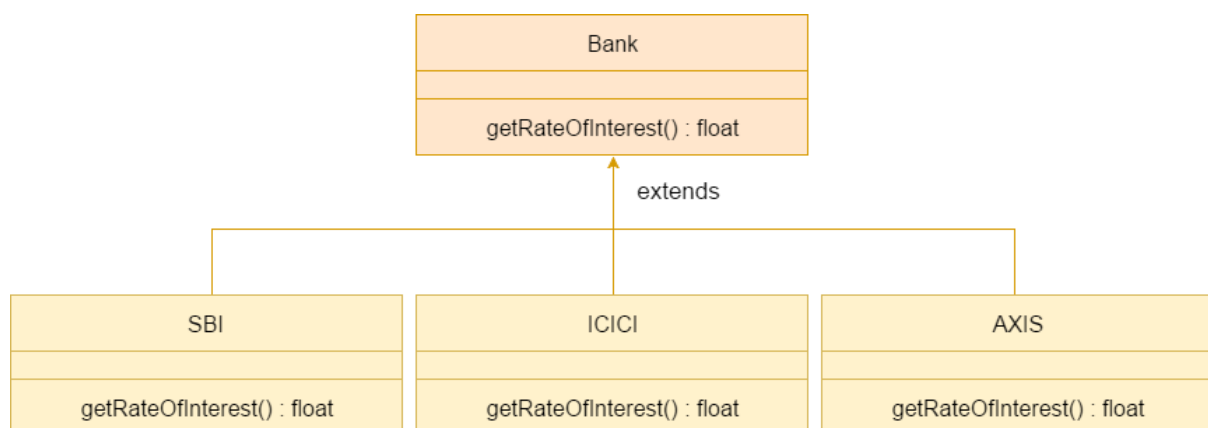
In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method are the same, and there is IS-A relationship between the classes, so there is method overriding.

```
1. class Vehicle{
2.     //defining a method
3.     void run(){System.out.println("Vehicle is running");}
4. }
5. //Creating a child class
6. class Bike2 extends Vehicle{
7.     //defining the same method as in the parent class
8.     void run(){System.out.println("Bike is running safely");}
9.
10.    public static void main(String args[]){
```

```
11. Bike2 obj = new Bike2();//creating object
12. obj.run();//calling method
13. }
14. }
```

## 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



*Java method overriding is mostly used in Runtime Polymorphism which we will learn in next pages.*

```
class Bank{
int getRateOfInterest(){return 0;}
}
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}
```

```
class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

## Can we override static method?

No, a static method cannot be overridden. It can be proved by runtime polymorphism, so we will learn it later.

---

## 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.

# Difference between method overloading and method overriding in java

No.	Method Overloading	Method Overriding
1)	Method overloading is used to <i>increase the readability</i> of the program.	Method overriding is used to <i>provide the specific implementation</i> of the method that is already provided by its super class.
2)	Method overloading is performed <i>within class</i> .	Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship.
3)	In case of method overloading, <i>parameter must be different</i> .	In case of method overriding, <i>parameter must be same</i> .
4)	Method overloading is the example of <i>compile time polymorphism</i> .	Method overriding is the example of <i>run time polymorphism</i> .
5)	In java, method overloading can't be performed by changing return type of the method only. <i>Return type can be same or different</i> in method overloading. But you must have to change the parameter.	<i>Return type must be same or covariant</i> in method overriding.