**Inheritance in Java**

1. [Inheritance](http://www.javatpoint.com/inheritance-in-java)
2. [Types of Inheritance](http://www.javatpoint.com/inheritance-in-java#inheritancetypes)
3. [Why multiple inheritance is not possible in java in case of class?](http://www.javatpoint.com/inheritance-in-java#inheritancenotmultiple)

**Inheritance** is a mechanism in which one object acquires all the properties and behaviours of parent object.

The idea behind inheritance is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you reuse (or inherit) methods and fields, and you add new methods and fields to adapt your new class to new situations.

Inheritance represents the **IS-A relationship**.

**Why use Inheritance?**

* For Method Overriding (So Runtime Polymorphism).
* For Code Reusability.

**Syntax of Inheritance**

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

|  |
| --- |
| The keyword extends indicates that you are making a new class that derives from an existing class. In the terminology of Java, a class that is inherited is called a superclass. The new class is called a subclass.  |

**Understanding the simple example of inheritance**



As displayed in the above figure, Programmer is the subclass and Employee is the superclass. Relationship between 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.
5. class Programmer extends Employee{
6. int bonus=10000;
7.
8. public static void main(String args[]){
9. Programmer p=new Programmer();
10. System.out.println("Programmer salary is:"+p.salary);
11. System.out.println("Bonus of Programmer is:"+p.bonus);
12. }
13. }

Output:Programmer salary is:40000.0

 Bonus of programmer is:10000

In the above example,Programmer object can access the field of own class as well as of Employee class i.e. code reusability.

### Types of Inheritance

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

Multiple and Hybrid is supported through interface only. We will learn about interfaces later.



#### Multiple inheritance is not supported in java in case of class.

When a class extends multiple classes i.e. known as multiple inheritance. For Example:



### Que) Why multiple inheritance is not supported in java?

* To reduce the complexity and simplify the language, multiple inheritance is not supported in java. For example:
1. class A{
2. void msg(){System.out.println("Hello");}
3. }
4.
5. class B{
6. void msg(){System.out.println("Welcome");}
7. }
8.
9. class C extends A,B{//suppose if it were
10.
11. Public Static void main(String args[]){
12. C obj=new C();
13. obj.msg();//Now which msg() method would be invoked?
14. }
15. }

**Inheritance can be classified to 5 types.**

1. Single Inheritance
2. Hierarchical Inheritance
3. Multi Level Inheritance
4. Hybrid Inheritance
5. Multiple Inheritance
	1. **Single Inheritance**

	when a single derived class is created from a single base class then the inheritance is called as single inheritance.

	

	**2. Hierarchical Inheritance**
	when more than one derived class are created from a single base class, then that inheritance is called as hierarchical inheritance.

	

	**3. Multi Level Inheritance**
	when a derived class is created from another derived class, then that inheritance is called as multi level inheritance.

	

	**4. Hybrid Inheritance**
	Any combination of single, hierarchical and multi level inheritances is called as hybrid inheritance.

	

	**5. Multiple Inheritance**
	when a derived class is created from more than one base class then that inheritance is called as multiple inheritance. But multiple inheritance is not supported by .net using classes and can be done using interfaces.

	

	Handling the complexity that causes due to multiple inheritance is very complex. Hence it was not supported in java with class and it can be done with interfaces.

### 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.

### Single Inheritance Types of inheritance in Java: Single,Multiple,Multilevel & Hybrid

**Single Inheritance example program in Java**

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

 }

}

### 2) Multiple Inheritance

“**Multiple Inheritance**” refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



Note 1: Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

Note 2: Most of the new OO languages like **Small Talk, Java, C# do not support Multiple inheritance**. Multiple Inheritance is supported in C++.

### 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. For more details and example refer 

**Multilevel Inheritance example program in Java**

Class X

{

 public void methodX()

 {

 System.out.println("Class X method");

 }

}

Class Y extends X

{

public void methodY()

{

System.out.println("class Y method");

}

}

Class Z extends Y

{

 public void methodZ()

 {

 System.out.println("class Z method");

 }

 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

 }

}

### 4) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D. 

### 5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple inheritance.** A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.

# Does Java support Multiple inheritance?

### Why Java doesn’t support multiple inheritance?

C++ , Common lisp and few other languages supports multiple inheritance while java doesn’t support it. It is just to **remove ambiguity**, because **multiple inheritance** can cause ambiguity in few scenarios. One of the most common scenario is **Diamond problem.**

**What is diamond problem?**
Consider the below diagram which shows multiple inheritance as Class D extends both Class B & C. Now lets assume we have a method in class A and class B & C overrides that method in their own way. **Wait!! here the problem comes**- Because D is extending both B & C so if D wants to use the same method which method would be called (the overridden method of B or the overridden method of C). Ambiguity. That’s the main reason why Java doesn’t support multiple inheritance.



### How to achieve multiple inheritance in Java using interfaces?

interface X

{

 public void myMethod();

}

interface Y

{

 public void myMethod();

}

class Demo implements X, Y

{

 public void myMethod()

 {

 System.out.println(" Multiple inheritance example using interfaces");

 }

}

As you can see that the class implemented two interfaces. A class can implement any number of interfaces. In this case there is no ambiguity even though both the interfaces are having same method. Why? Because methods in an interface are always [abstract](http://beginnersbook.com/2013/05/java-abstract-class-method/) by default, which doesn’t let them to give their implementation (or method definition ) in interface itself.

