**Inheritance in java** is a mechanism in which a child class acquires all the properties and behaviors of a parent class. The aim of inheritance is to provide the reusability of code.

Child class can access parent class’s methods and variables.

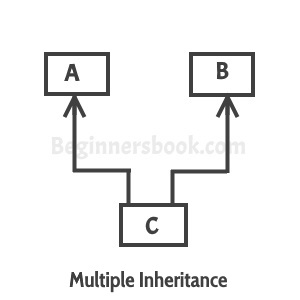
We achieve inheritance using extends keyword.

**VVVIMP** :

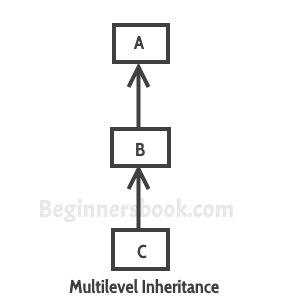
1. Parent class reference can't access child class members and methods **[**this is not applicable in OVERRIDING**]**. And, for **OVERRIDING** : Jya class cha object tya class chi method call hote.
2. Parent class reference can hold parent object and child object. Child class can hold object of child class, but CAN NOT hold object of parent.
3. **Static** methods : child can access parent class’s static method by creating child’s object or directly using childClassName.methodName
4. **Private** methods : We cannot override private methods. We cannot access private method of parent using child class object. We can overload private methods in Java as a normal method but, you can access these from the same class.
5. **Final :** We can overload a final method, and its possible in JAVA.But we cannot override final methods.
6. **Constructor** : In case of overriding, if parent and child class both possess constructor implementation, and we execute ‘parent newobj = **new** Chilld();’ this statement, then here 1st PARENT constructor and then CHILD constructor will be executed. Both will be executed not one.
7. **Constructor Overriding and Inheritance** : Java class **Does not allow** constructor Overriding and constructor inheritance. But, constructor overloading is allowed.

**multiple inheritance :**

When one class extends more than one classes then this is called **multiple inheritance**. For example: Class C extends class A and B then this [type of inheritance](https://beginnersbook.com/2013/05/java-inheritance-types/) is known as multiple inheritance. Java **doesn’t** allow multiple inheritance.



**multilevel inheritance :**

When a class extends a class, which extends another class then this is called **multilevel inheritance**. For example class C extends class B and class B extends class A then this [type of inheritance](https://beginnersbook.com/2013/05/java-inheritance-types/) is known as multilevel inheritance.  
Lets see this in a diagram:  


Can we overload a static method ? >> YES, we can. Static method overriding is nothing but method hiding.

**Exception handling and it’s impact on Inheritance :**

 **If the superclass method does not declare an exception**

* If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

 **If the superclass method declares an exception**

* If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

# **What is method hiding in Java and how to use it?**

When super class and sub class contains same method including parameters and if they are static.

The method in the super class will be hidden by the one that is in the sub class. This mechanism is known as method hiding. in method hiding, it calls method based on the reference not based on the object

We can not override the static methods in a derived class because static methods are linked with the class, not with the object. It means when we call a static method then JVM does not pass this reference to it as it does for all non-static methods. Therefore run-time binding cannot take place for static methods.

## Example

class Demo{

public static void demoMethod() {

System.out.println("method of super class");

}

}

public class Sample extends Demo {

public static void demoMethod() {

System.out.println("method of sub class");

}

public static void main(String args[] ) {

Sample.demoMethod();

}

}

## Output

method of sub class

**We can prevent method overriding in Java in 3 ways**

* By making method final in the base class
* By making a method static in the base class
* By making a method private in the base class

1. Using FINAL : Final method can not be overridden.
2. class Base {
3. public void show() {
4. System.out.println("Base class show() method");
5. }
6. public final void test() {
7. System.out.println("Base class test() method");
8. }
9. }
10. class Derived extends Base {
11. public void show() {
12. System.out.println("Derived class show() method");
13. }
14. // can not override test() method because its final in Base class
15. /\*
16. \*  public void test() { System.out.println("Derived class test() method"); }
17. \*/
18. }
19. public class Test {
20. public static void main(String[] args) {
21. Base ref = new Derived();
22. // Calling the final method test()
23. ref.test();
24. // Calling the overridden method show()
25. ref.show();
26. }
27. }

## Output

Base class test() method

Derived class show() method

2. Using Static : We cant override static methods.

class Base {

   public void show() {

      System.out.println("Base class show() method");

   }

   public static void test() {

      System.out.println("Base class test() method");

   }

}

class Derived extends Base {

   public void show() {

      System.out.println("Derived class show() method");

   }

      // This is not an overridden method, this will be considered as new method in Derived class

   public static void test() {

      System.out.println("Derived class test() method");

   }

}

public class Test {

   public static void main(String[] args) {

      Base ref = new Derived();

      // It will call the Base class's test() because it had static binding

      ref.test();

      // Calling the overridden method show()

      ref.show();

   }

}

## Output

Base class test() method

Derived class show() method

3. Using Private : Private methods of the base class are not visible in a derived class, hence they cannot be overridden.

class Base {

   public void show() {

      System.out.println("Base class show() method");

   }

   private void test() {

      System.out.println("Base class test() method");

   }

}

class Derived extends Base {

   public void show() {

      System.out.println("Derived class show() method");

   }

   // This is not an overridden method, this will be considered as other method.

   public void test() {

      System.out.println("Derived class test() method");

   }

}

public class Test {

   public static void main(String[] args) {

      Base ref = new Derived();

   // Cannot call the private method test(), this line will give compile time error

   // ref.test();

   // Calling the overridden method show()

      ref.show();

   }}

## Output

Derived class show() method