**Can a private method in super class be overriden in the sub-class?**

[September 13, 2013 9:01 pm](http://www.crazyforcode.com/private-method-super-class-overriden-sub-class/) | [Leave a Comment](http://www.crazyforcode.com/private-method-super-class-overriden-sub-class/#comments) | [crazyadmin](http://www.crazyforcode.com/author/dev1/)

No, a private method cannot be overridden since it is not visible from any other class. If we do this than its a new method for your subclass that has no relation to the superclass method. One way to look at it is to ask yourself whether it would be legal to write super.func() in the Derived class. Ofcourse, it will give compile time error.

|  |
| --- |
| class Base { |
| private void func(){ | |

|  |  |
| --- | --- |
| System.out.println("In base func method"); | |
| }; |

|  |
| --- |
| public void func2() { |
| System.out.println("func2"); | |

|  |  |
| --- | --- |
| func(); | |
| } |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| class Derived extends Base { |
| public void func(){   //  Is this an overriding method? | |

|  |  |
| --- | --- |
| System.out.println("In Derived Class func method"); | |
| } |

|  |  |
| --- | --- |
| } | |
|  |

|  |
| --- |
| class InheritDemo { |
| public static void main(String [] args) { | |

|  |  |
| --- | --- |
| Derived D = new Derived(); | |
| D.func2(); |

|  |  |
| --- | --- |
| } | |
| } |

It will print:

func2  
In base func method

When you change func() in Base to public, then it will be an override, and the output will change to:

func2  
In Derived Class func method

public class Sum {

class Base {

private void func(){

System.out.println("In base func method");

};

public void func2() {

System.out.println("func2");

func();

}

}

class Derived extends Base {

private void func(){ // Is this an overriding method?

System.out.println("In Derived Class func method");

}

}

// Driver code

public static void main(String args[])

{

Sum s = new Sum();

Derived D = s.new Derived();

D.func2();

}

}

func2

In base func method

**This is also prove that private method does not take part in overriding method().**

**Resolving of Non static method?**

**static varible, static method, non static varible are resolve basic of reference type. non static mehtod are resolve on the bASIC OF OBJECT TYPE.**

  class A

{

void m1() //non static method resolve on the bascis of obj type.

{

System.out.println("Inside A's m1 method");

}

}

class B extends A

{

// overriding m1()

void m1()

{

System.out.println("Inside B's m1 method");

}

}

// Driver class

class Dispatch

{

public static void main(String args[])

{

// object of type A

A a = new A();

// object of type B

B b = new B();

// obtain a reference of type A

**A ref;**

// ref refers to an A object

ref = a;

// calling A's version of m1()

ref.m1(); m1 is NSM invoked based on object type.

// now ref refers to a B object

ref = b;

// calling B's version of m1()

ref.m1(); m1 is NSM invoked based on object type.

}

}

Inside A's m1 method

Inside B's m1 method

class A

{

static void m1() resolve on the basic of reference type.

{

System.out.println("Inside A's m1 method");

}

}

class B extends A

{

// overriding m1()

static void m1()

{

System.out.println("Inside B's m1 method");

}

}

Inside A's m1 method

Inside A's m1 method

// now ref refers to a B object

B rr; rr = b;

// calling B's version of m1()

rr.m1();

Inside A's m1 method

Inside B's m1 method

### **Q) Can we declare a constructor final?**

No, because constructor is never inherited.

When you set a method as final it means: "I don't want any class override it." But according to the Java Language Specification:

[JLS 8.8](https://docs.oracle.com/javase/specs/jls/se11/html/jls-8.html#jls-8.8) - "Constructor declarations are not members. They are never inherited and therefore are not subject to hiding or overriding."

When you set a method as abstract it means: "This method doesn't have a body and it should be implemented in a child class." But the constructor is called implicitly when the new keyword is used so it can't lack a body.

When you set a method as static it means: "This method belongs to the class, not a particular object." But the constructor is implicitly called to initialize an object, so there is no purpose in having a static constructor.

**Can we overload static methods?**  
The answer is ‘Yes’. We can have two ore more static methods with same name, but differences in input parameters. For example, consider the following Java program.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public static void foo(int a) {          System.out.println("Test.foo(int) called ");      }      public static void main(String args[])      {          Test.foo();          Test.foo(10);      }  } |

Test.foo() called

Test.foo(int) called

**Can we overload methods that differ only by static keyword?**  
We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is same). See following Java program for example. This behaviour is same in C++

|  |
| --- |
| // filename Test.java  public class Test {      public static void foo() {          System.out.println("Test.foo() called ");      }      public void foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }      public static void main(String args[]) {          Test.foo();      }  } |

Output: Compiler Error, cannot redefine foo()

**Can we Override static methods in java?**  
We can declare static methods with same signature in subclass, but it is not considered overriding as there won’t be any run-time polymorphism. Hence the answer is ‘No’.  
If a derived class defines a static method with same signature as a static method in base class, the method in the derived class hides the method in the base class.

**Method hiding**

|  |
| --- |
| /\* Java program to show that if static method is redefined by     a derived class, then it is not overriding. \*/    // Superclass  class Base {        // Static method in base class which will be hidden in subclass      public static void display() {          System.out.println("Static or class method from Base");      }         // Non-static method which will be overridden in derived class       public void print()  {           System.out.println("Non-static or Instance method from Base");      }  }    // Subclass  class Derived extends Base {        // This method hides display() in Base      public static void display() {           System.out.println("Static or class method from Derived");      }        // This method overrides print() in Base      public void print() {           System.out.println("Non-static or Instance method from Derived");     }  }    // Driver class  public class Test {      public static void main(String args[ ])  {         Base obj1 = new Derived();           // calls Base's display() becoz resolving on the base of reference type.         obj1.display();           // Here overriding works and Derive's print() is called         obj1.print(); non static method resolved on the base of object type.      }  } |

Output:

Static or class method from Base

Non-static or Instance method from Derived

## if we do overrind on the base of different in arguments then we get compile time error.

class Derived extends Base {

// This method hides display() in Base

public static void display(int i) {

System.out.println("Static or class method from Derived");

}

// This method overrides print() in Base

public void print() {

System.out.println("Non-static or Instance method from Derived");

} }

prog.java:41: error: method display in class Base cannot be applied to given types;

obj1.display(1);

^

required: no arguments

found: int

reason: actual and formal argument lists differ in length

1 error

# **Why doesn’t the Java language support multiple inheritance?**

[August 28, 2013 10:55 pm](http://www.crazyforcode.com/java-language-support-multiple-inheritance/) | [1 Comment](http://www.crazyforcode.com/java-language-support-multiple-inheritance/#comments) | [crazyadmin](http://www.crazyforcode.com/author/dev1/)

There are 2 reasons mentioned that will give you a idea why we don’t have multiple inheritance in java.

1st is ambiguity around Diamond problem, consider a class A has foo() method and then B and C derived from A and has there own foo() implementation and now class D derive from B and C using multiple inheritance and if we refer just foo() compiler will not be able to decide which foo() it should invoke. This is also called Diamond problem because structure on this inheritance scenario is similar to 4 edge diamond, see below

          A foo()  
        /      \  
      /          \  
   B foo()     C foo()  
      \          /  
        \     /  
         D foo()

Even if we remove the top head of diamond class A and allow multiple inheritances we will see this problem of ambiguity.

2nd reason: Multiple inheritances does complicate the design and creates problem during casting, constructor chaining etc and given that there are not many scenario on which you need multiple inheritance its wise decision to omit it for the sake of simplicity. Also java avoids this ambiguity by supporting single inheritance with interfaces. Since interface only have method declaration and doesn’t provide any implementation there will only be just one implementation of specific method hence there would not be any ambiguity.

## **What modifiers are allowed for methods in an Interface?**

Only public and abstract modifiers are allowed for methods in interfaces.

**Note**:

* You can’t mark an interface as final.
* Interface variables must be static.
* An Interface cannot extend anything but another interfaces.
* Methods in an interface are implicitly public.

**We can write abstract keyword on the top of interface, but complier will delete it.**

**We can extend keyword on the top of interface, but not with classes only with interfaces.**

**We can’t make variables private and protected.**

**We can’t write concrete methods.**

**By default, methods of interface are public abstract so using of private, protected, static, final are illegal.**

**We can’t declare constructor in interface.**

**We can’t create object for interface.**

**We can create reference for interface.**

**We can’t declare static block and non-static block inside in interface.**