\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html

Difference between Method Overloading and Overriding in Java?

**Method Overloading vs Method Overriding**

1. overriding – связывание метода происходит при Runtime с помощью dynamic binding. Методы private, static and final НЕ могут быть overridded. Имя и сигнатура метода должна остаться такой же. Overriding делается в разных классах.

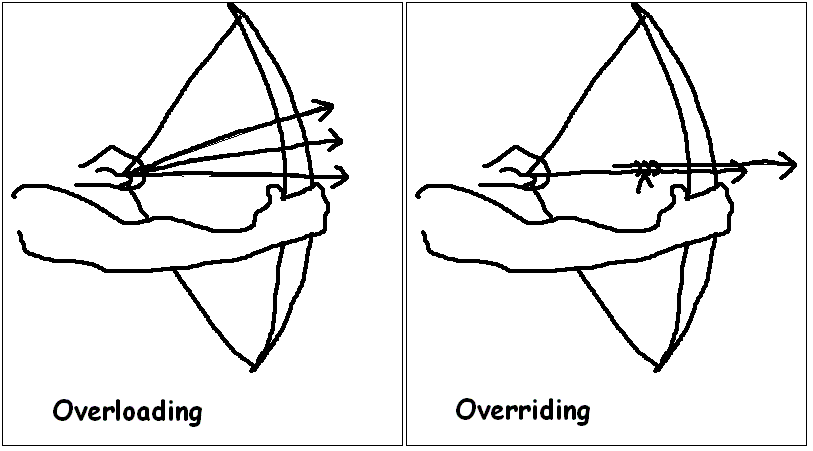
Нельзя оверридить public static final, а только точно такой же метод в классе наследнике.

Добавлено в Java 5: Covariant Method Overriding in Java – можно изменить возвражаемый тип у оверридед метода на более узкий, чем в родителе. Т.е. с Object на String например, в методе clone.

1. Overloading (перегрузка метода) – такие методы связываются (resolved) на этапе компиляции с помощью static binding. Сигнатура метода должна измениться, быть другой чем у исходного метода.

Т.е. надо изменить или кол-во входных параметров, или их тип, или их порядок.

Можно оверлоадить public static final методы и конструкторы.

Though name of method remains same in case of both method overloading and overriding, main difference comes form the fact that method overloading is resolved during compile time, while method overriding is resolved at runtime. Also rules of overriding or overloading a method is different in Java. For example, private, static and final method cannot be overriding in Java but you can still overload them. For overriding both name and signature of method must remain same, but in for overloading method signature must be different. Last but not the least difference between them is that call to overloaded methods are resolved using static binding while call to overridden method is resolved using dynamic binding in Java.  By the way, Method overloading and method overriding in Java is two important concept in Java which allows Java programmer to declare method with same name but different behavior. Method overloading and method overriding is based on Polymorphism in Java. In case of method overloading, method with same name co-exists in same class but they must have different method signature, while in case of method overriding, method with same name is declared in derived class or sub class.Method overloading is resolved using [static binding in Java](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time while method overriding is resolved using dynamic binding in Java at runtime. In short When you overload a method in Java its [method signature](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) got changed while in case of overriding method signature remains same but a method can only be overridden in sub class. Since Java supports Polymorphism and resolve object at run-time it is capable to call overridden method in Java. By the way difference between method overloading and overriding is also one of the popular [Java design question](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html) and appear in almost all levels of Java interviews.

## What is method overloading and overriding in Java?

In this Java tutorial we will see how Java allows you to create two methods of same name by using method overloading and method overriding. We will also touch base on how methods are bonded or called by Compiler and [Java Virtual Machine](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html) and finally we will answer of popular interview questions difference between method overloading and method overriding in Java. This article is in my series of Java article which discusses about Interview e.g. Difference between Synchronized Collection and Concurrent Collection or How to Stop Thread in Java. Please let me know if you have some other interview questions and you are looking answer or reason for that and here in Javarevisited we will try to find and discuss those interview questions.

### How to Overload a Method in Java

If you have two methods with same name in one Java class with different method signature than its called overloaded method in Java. Generally overloaded method in Java has different set of arguments to perform something based on different number of input. You can also [overload constructor in Java](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html), which we will see in following example of method overloading in Java. Binding of overloading method occurs during compile time and overloaded calls resolved using static binding. To overload a Java method just changes its signature. Just remember in order to change signature you either need to change number of argument, type of argument or order of argument in Java if they are of different types. Since return type is not part of method signature simply changing return type will result in duplicate method and you will get compile time error in Java. In our example of Loan and PersonalLoan class, createLoan method is overloaded. Since you have two crateLoan() method with one takes one argument lender while other take two argument both lender and interestRate. Remember you can overload [static method in Java](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), you can also overload private and [final method in Java](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) but you can not override them.

### How to Override a Method in Java

In order to override a Java method, you need to create a child class which extends parent. Overridden method in Java also shares same name as original method in Java but can only be overridden in sub class. Original method has to be defined inside [interface](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) or base class, which can be [abstract](http://javarevisited.blogspot.sg/2010/10/abstraction-in-java.html) as well. When you override a method in Java its signature remains exactly same including return type. JVM resolves correct overridden method based upon object at run-time by using dynamic binding in Java. For example in our case when we call personalLoan.toString() method even though personalLoan object is of type Loan actual method called would be from PersonalLoan class because object referenced by personalLoan variable is of type PersonalLoan(). This is very useful technique to modify behavior of a function in Java based on different implementation. equals(), hashcode() and compareTo() methods are classic example of overridden methods in Java.  
  
Another important point is that you can not override static method in Java because they are associated with Class rather than object and resolved and bonded during compile time and that’s the reason you cannot override main method in Java. Similar to static, private and final methods are also not overridden in Java. By the way, as part of overriding best practice, always [use @Override annotation](http://javarevisited.blogspot.co.uk/2012/11/why-use-override-annotation-in-java.html), while overriding method from an abstract class or interface.

### Rules of Method Overriding in Java

Following are rules of method overriding in java which must be followed while overriding any method. As stated earlier private, static and final method can not be overridden in Java.

1. Method signature must be same including return type, number of method parameters, type of parameters and order of parameters
2. Overriding method can not throw higher Exception than original or overridden method. means if original method throws IOException than overriding method can not throw super class of IOException e.g. Exception but it can throw any sub class of IOException or simply does not throw any Exception. This rule only applies to checked Exception in Java, overridden method is free to throw any [unchecked Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html).
3. Overriding method can not reduce accessibility of overridden method , means if original or overridden method is public than overriding method can not make it protected.

## Difference between Method Overloading vs Overriding in Java

Overloading vs Overriding in Java is one of the [popular java interview questions](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) at many companies and asked at different levels of programmers. Here are some important difference between overloading and overriding in Java. Though It's more important is to understand how to use both overloading and overriding, these difference are good from interview perspective and gives some basic idea as well:  
  
1) First and most important difference between method overloading and overriding is that, In case of method overloading in Java, signature of method changes while in case of method overriding it remain same.  
  
2) Second major difference between method overloading vs overriding in Java is that You can overload method in one class but overriding can only be done on subclass.  
  
3) You can not override static, final and private method in Java but you can overload static, final or private method in Java.  
  
4) Overloaded method in Java is bonded by static binding and overridden methods are subject to dynamic binding.  
  
5) Private and final method can also be not overridden in Java.  
  
By the way, you might have heard about "a picture is worth more than thousand words" and this is made true by following image. By looking at the pic you can clearly understand difference between method overloading and overriding in Java.

### Handling Exception while overloading and overriding method in Java

While overriding a method it can only throw [checked exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) declared by by overridden method or any subclass of it, means if overridden method throws IOExcpetion than overriding method can throw sub classes of IOExcpetion e.g. FileNotFoundException but not wider exception e.g. Exception or Throwable. This restriction is only for checked Exception for RuntimeException you can throw any RuntimeException. Overloaded method in Java doesn't have such restriction and you are free to modify throws clause as per your need.

## Method Overloading and Overriding Example in Java

Here is an example of both method overloading and method overriding in Java. In order to explain the concept we have create two classes Loan and PersonalLoan. createLoan() method is overloaded as it has different version with different signature, while toString() method which is original declared in Object class is overridden in both Loan and PersonalLoan class.

**public** **class** **OverloadingOverridingTest** {

**public** **static** **void** **main**(String[] args) {

// Example of method overloading in Java

Loan cheapLoan = Loan.createLoan("HSBC");

Loan veryCheapLoan = Loan.createLoan("Citibank", **8.5**);

// Example of method overriding in Java

Loan personalLoan = **new** PersonalLoan();

personalLoan.toString();

}

}

**public** **class** **Loan** {

**private** **double** interestRate;

**private** String customer;

**private** String lender;

**public** **static** Loan **createLoan**(String lender) {

Loan loan = **new** Loan();

loan.lender = lender;

**return** loan;

}

**public** **static** Loan **createLoan**(String lender, **double** interestRate) {

Loan loan = **new** Loan();

loan.lender = lender;

loan.interestRate = interestRate;

**return** loan;

}

**@Override**

**public** String **toString**() {

**return** "This is Loan by Citibank";

}

}

**public** **class** **PersonalLoan** **extends** Loan {

**@Override**

**public** String **toString**() {

**return** "This is Personal Loan by Citibank";

}

}

### Things to Remember

1) In case of method overloading method signature gets changed while in case of overriding signature remains same.  
  
2) Return type is not part of method signature in Java.  
  
3) Overloaded method can be subject to compile time binding but overridden method can only be bind at run-time.  
  
4) Both overloaded and overridden method has same name in Java.  
  
5) Static method can not be overridden in Java.  
  
6) Since private method is also not visible outside of class, it can not be overridden and method binding happens during compile time.  
  
7) From Java 5 onwards you can use annotation in Java to declare overridden method just like we did with @override. @override annotation allows compiler, IDE like [NetBeans](http://javarevisited.blogspot.com/2011/08/how-to-view-javadoc-in-netbeans-ide-70.html) and [Eclipse](http://javarevisited.blogspot.com/2010/10/eclipse-tutorial-most-useful-eclipse.html) to cross verify or check if this method is really overrides super class method or not.

### Covariant Method Overriding in Java

One of my reader Rajeev makes an interesting comment about one change related to return type of overriding method from Java 5 onwards, which enable to use subtype of return type of overridden method. This is really useful, when original method returns a general type like java.lang.Object. If you are [overriding clone() method in Java](http://javarevisited.blogspot.sg/2013/09/how-clone-method-works-in-java.html) then you can use this feature to return actual type, instead of returning java.lang.Object and can save caller from type-casting cloned object. Here is the actual comment from Rajeev:  
  
  
Hi Javin,I visit your blog regularly and I found that you missed *covariant return* which is added in Java 5 in the case of method overriding. When a subclass wants to change the method implementation of an inherited method (an override), the subclass must define a method that matches the inherited version exactly. Or, as of Java 5, you're allowed to change the return type in the overriding method as long as the new return type is a subtype of the declared return type of the overridden (super class) method. Let's look at a covariant return in action:

**class** **Alpha** {

Alpha **doStuff**(**char** c) {

**return** **new** **Alpha**();

}

}

**class** **Beta** **extends** Alpha {

Beta **doStuff**(**char** c) { // legal override in Java 1.5

**return** **new** **Beta**();

}

}

You can see that Beta class which is overriding doStuff() method from Alpha class is returning Beta type and not Alpha type. This will remove type casting on client side. See [here](http://javarevisited.blogspot.sg/2014/03/covariant-method-overriding-of-java-5.html) to learn more about covariant method overriding in Java.  
  
  
AS I said one of the good example of this is overriding clone method and using return type as Actual type instead of java.lang.Object, which is suggested by Joshua Bloch in Effective Java as well. This in in-fact one of the Java best practices while implementing clone method in Java. By the way don't forget to follow these [Java overloading best practices](http://javarevisited.blogspot.com.au/2013/01/java-best-practices-method-overloading-constructor.html), when doing it in your project.

----------------------

to add on what you have described, overloading is done in two steps, first JVM finds all the methods which are applicable for a given type of argument and than in second step it pick the most specific method. for exmaple if you have two method one which takes Object clas as parameter and other takes String class as parameter like below  
  
public void show(Object ob){}  
public void show(String str){}  
  
and you call show(null) than since both show() method which are overloaded are applicable but Java will pick the most specific one which is the String version because String is more specific than Object.

--------------------------

Overriding method cannot assign weaker access privileges:   
protected in parent  
must be kept  
public/protected in subclass

------------------------------

<http://javarevisited.blogspot.sg/2014/03/covariant-method-overriding-of-java-5.html>

Covariant Method Overriding of Java 5 - Coding Best Practices

Sometime knowledge of a specific Java feature can improve code quality, *Covariant method overriding* is one of such feature. Covariant method overriding was introduced in Java 5, but it seems it lost between other more powerful features of that release. Surprisingly not many Java programmer knows about Covariant overriding, including myself, until I read, one of the best Java book on Collection framework, Java Generics and Collection. **Covariant method overriding** helps to remove [type casting](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html) on client side, by allowing you to return subtype of actually return type of overridden method. Covariant overriding can be really useful, while overriding methods which returns object e.g. clone() method. Since clone() return object every client needs to cast on to appropriate subclass, not any more. By using Java 5 covariant overriding, we can directly return subtype instead of object, as we will seen in examples of this article. This feature is not a star feature like Generics or Enum, but it's definitely something worth knowing, given overriding methods are integral part of Java programming. I have discussed a bit about this feature earlier in [difference between overriding and overloading](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html) article, and here I will show couple of more examples to justify it's usage.  
  
Covariant Method Overriding Example

As I said, one of the best example of this is overriding clone() method, which is declared in java.lang.Object class and has a return type of Object. If you have used java.util.Date in your project and has called it's clone method to make a copy, you might know that you need to cast it back to Date as shown below :

Date d = **new** Date();

Date clone = (Date) d.clone();

wouldn't it be great if [clone() method](http://javarevisited.blogspot.sg/2013/09/how-clone-method-works-in-java.html) can return Date instead of Object? Well it can from Java 5 onwards, but unfortunately code in java.util.Date is not yet updated to take advantage of this, at least till Java 6. Now let's create our own class and override clone() method to demonstrate use of covariant method overriding in Java.

**import** **java.util.Date**;

/\*\*

\* Java program to demonstrate how to use covariant method overriding to avoid

\* casting at client side.

\* @author http://javarevisited.blogspot.com

\*/

**public** **class** **CovariantMethodOverridingTest** {

**public** **static** **void** **main**(String args[]) {

//without covariant overriding, cast at client side needed

Date d = **new** Date();

Date clone = (Date) d.clone(); //casting required

//with covariant method overriding, no client side cast

Duck duck = **new** Duck(**0xFFFFFF**);

Duck copy = duck.clone(); //no casting

}

}

**class** **Duck** **implements** Cloneable{

**private** **int** color = **0xFFFFFF**;

**public** **Duck**(**int** color){

**this**.color = color;

}

**public** **int** **getColor**(){

**return** color;

}

@Override

**protected** Duck **clone**() {

Duck clone = **null**;

**try**{

clone = (Duck) **super**.clone();

}**catch**(CloneNotSupportedException ex){

**throw** **new** **RuntimeException**(ex);

}

**return** clone;

}

}

You can see in our test program, we don't need to type cast object returned from clone() method to Duck, because instead of java.lang.Object, its returning java.util.Date. This is also obvious from looking at overriding clone() method in Duck class, you can see its return type is Duck, and not Object. In fact, if you have implemented your own [Comparator](http://javarevisited.blogspot.sg/2014/01/java-comparator-example-for-custom.html) or [Comparable](http://javarevisited.blogspot.sg/2014/02/java-comparable-example-for-natural-order-sorting.html) in Java, you can also see that their compare() and compareTo() directly return the type specified in Type parameter instead of object. Though this looks a very small and subtle change, it's impact is huge. I am sure they will implement clone() method of java.util.Date sooner or later to take advantage of this feature, unless they decide to deprecate it in favour of new Date and Time API coming in Java 8.  
  
  
That's all on **covariant method overriding of Java 5** guys. Always take advantage of this feature, while overriding method in Java 5, 6 ,7 and very soon on Java 8. You can return any subtype, based upon your need. All other [rules of method overriding](http://java67.blogspot.sg/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html) remains same. I know Java is vast, and there are many more features like this, which can be really handy on day to day job but many times remains hidden from large number of Java developers. I also suggest reading Java Generics and Collection, a must read book for any senior Java developer. I always remain in search for hidden gems of Java programming language, API or third-party library which can make your task easy, can improve code quality and add value by any means. If you come across any such feature, which you like to share with us, go ahead and comment about it.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://javarevisited.blogspot.com.au/2013/01/java-best-practices-method-overloading-constructor.html>

Method and Constructor Overloading Best Practices in Java

You need to be careful while overloading a method in Java, especially after introduction of autoboxing in Java 5. Poorly overloaded method not only adds confusion among developers who use that but also they are error prone and leaves your program on compiler's mercy to select proper method. One of the best example of poorly overloaded method is remove() method from ArrayList. There are two versions of remove, first one which takes an Object as argument i.e. remove(Object element) and second one, which takes an index as argument i.e. remove(int index). It worked fine until Java 1.4 where there is clearly a distinction between primitive types and objects type but in Java 1.5, where you can pass an int primitive to a method which accepts an Integer object, creates some nasty problem. Now suppose you have an ArrayList of Integer with values 1, 2 and 3, and you call remove(1) then which method will be called? JVM can interpret 1 as index also or 1 as Integer object also.  
  
  
It's best to avoid issues related to method overloading by following some Java best practices. For those who doesn’t know what is method overloading in Java? method overloading means declaring more than one method with same name but different [method signatures](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). This is generally done to create methods which does same thing but with different types. For example, one of the most popular example of method overloading is System.out.println() method, which is overloaded to accept different types of parameters like String, double, int etc, see this Java tutorial on [method overloading](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) and [static vs dynamic binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) for more details. By the way all of these Java best practices which are explained in context of method overloading are equally applicable to constructor overloading in Java, because in terms of overloading method and constructors are almost same.

## Java Best Practices - Method Overloading

[Java best practices for method and constructor overloading in Java](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Here are some of the common things which you can remember while overloading method or constructor in Java. These Java best practices are completely based upon experience and you may have some more to add on this list. let’s see my list of Java best practices while overloading method in Java.

**1) Don't overload method which accept same number of parameter with similar types**

Two overloaded method which accept same number of argument with similar types i.e. which follow same [type hierarchy](http://javarevisited.blogspot.sg/2012/12/what-is-type-casting-in-java-class-interface-example.html) is most common mistake while overloading method in Java. For example, find out which version of overloaded method will be invoked in following scenario :

**import** java.util.ArrayList;  
**import** java.util.LinkedList;  
**import** java.util.List;  
  
/\*\*  
 \* Java program to demonstrate some best practice to following while overloading

\* method in Java.This Java program shows a case of confusing method overloading in Java

\*  
 \* @author Javin Paul  
 \*/  
**public** **class** OverloadingTest {  
    
    **public** **static** **void** main(**String** args[]){  
       **List** abc = **new** **ArrayList**();  
       **List** bcd = **new** **LinkedList**();  
        
       ConfusingOverloading co = **new** ConfusingOverloading();  
       co.hasDuplicates(abc); *//should call to ArryList overloaded method*  
       co.hasDuplicates(bcd); *//should call to LinkedList overloaded method*  
    }  
  
    
}  
  
**class** ConfusingOverloading{  
    
    **public** **boolean** hasDuplicates (**List** collection){  
        **System**.out.println("overloaded method with Type List ");  
        **return** **true**;  
    }  
    
    **public** **boolean** hasDuplicates (**ArrayList** collection){  
        **System**.out.println("overloaded method with Type ArrayList ");  
        **return** **true**;  
    }  
    
    
    **public** **boolean** hasDuplicates (**LinkedList** collection){  
        **System**.out.println("overloaded method with Type LinkedList ");  
        **return** **true**;  
    }  
    
}  
  
**Output**  
overloaded method with **Type** **List**  
overloaded method with **Type** **List**

To surprise of some programmers method with argument type List is called both the time, instead of expected method which takes ArrayList and LinkedList, because method overloading is resolved at compile time using static binding in Java. This is also one of the reason, why its important to clearly understand [difference between method overloading and overriding in Java](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Here expected case is result of mistaking overloading as overriding, which work on actual object and happens at runtime. To know more about static and dynamic binding in Java , you can also see my post difference between static and dynamic binding in Java.

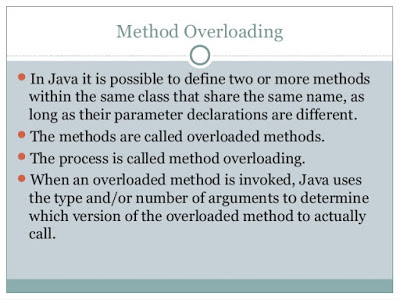
**2) Use radically different types while overloading method in Java**

It's completely legal and there is no ambiguity when two overloaded method accepts radically different types like String and Integer. Though both overloaded method will accept only one parameter, it’s still clear which method is called because both types are completely different to each other. Both programmer and compiler both know which method will be invoked for a particular call. One of the example of this kind of overloading is constructor of java.util.Scanner class which accepts File, InputStream or String as parameter, as shown below :

Scanner(File source)

Scanner(InputStream source)

Scanner(String source)

[](http://2.bp.blogspot.com/-oqQ1WUjQqb8/VXLspowzixI/AAAAAAAAC6Y/vJKmD0omMjA/s1600/method-overloading-21.jpg)

**3) Beware of Autoboxing while overloading method in Java**

Prior to introduction of [Autoboxing and unboxing in Java 5](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html), method which accept primitive type and object type were radically different and it’s clear which method will be invoked. Now with autoboxing it's really confusing. Clasical example of this kind overloading mistake is ArrayList’s remove() method, which is overloaded to accept index as well as Object. when you store Integer in ArrayList and call remove() method, It’s hard to find out which remove() method will be called, as shown in below example :

**List**<**Integer**> numbers = **new** **ArrayList**<**Integer**>();  
numbers.add(1);  
numbers.add(2);  
numbers.add(3);  
**System**.out.println("numbers: " + numbers);  
numbers.remove(1); *//should remove "1" as element or 2nd element from ArrayList*  
**System**.out.println("numbers: " + numbers);  
  
**Output:**  
numbers: [1, 2, 3]  
numbers: [1, 3]

Many Java programmer expect that Integer(1) object would be removed but since remove() is overloaded, compiler choose remove(int) over remove(Object). Rules of which overloaded method gets chosen in case of autoboxing is complex and hard to remember, so Its best to avoid two overloaded method where one accept Object and other accept primitive type. If by any chance you must have to do this then make sure both of them perform identical function.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2012/08/can-we-override-private-method-in-java.html>

Can we override private method in Java - Example program

No, We can not override private method in Java, just like [we can not override static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html). Like static methods, private method in Java is also bonded during compile time using static binding by Type information and doesn't depends on what kind of object a particular reference variable is holding. Since [method overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) works on [dynamic binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html), its not possible to override private method in Java. private methods are not even visible to Child class, they are only visible and accessible in the class on which they are declared. private keyword provides highest level of [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html). Though you can *hide private method* in Java by declaring another private method with same name and different method signature.

## Overriding private method in Java - Testing

[Can private method be overriden in Java ](http://1.bp.blogspot.com/-_GCqP1vu06Q/UBaTOZM869I/AAAAAAAAAaw/ykubu9U9kK4/s1600/java_logo_50_50.jpg)As per above paragraph we can not override private method in Java because its bonded during compile time using static binding. Now let's test this theory by an example Java program :

/\*\*  
 \*  
 \* Java program to demonstrate that **private method can not be overridden in Java**.  
 \* This Java programs calls both private and non private method with child class   
 \* object on constructor of parent class.  
 \* Only non private method of Child class invoked while private method of   
 \* Parent class is invoked, Which confirms that private method can not be overridden in Java  
 \*  and only be hidden if we declare same message in Child class.  
 \* @author  
 \*/  
**public** **class** PrivateMethodCanNotBeOverriden{  
    
    **public** **static** **void** main(**String** args[]) {  
        *//shows that private method can not be overridden in Java*   
        Parent parent = **new** Child();  
    }  
    
    
}  
  
**class** Parent{  
    
    **public** Parent(){  
        name();  
        normal();  
    }  
    
    **private** **void** name(){  
        **System**.out.printf("private method inside Parent class in Java %n");  
    }  
    
    **public** **void** normal(){  
        **System**.out.println("non private method from Parent class can be overridden");  
    }  
    
}  
  
**class** Child **extends** Parent{  
    
    */\*  
     \* Private method can not be overridden in Java, they can only be hidden  
     \*/*  
    **private** **void** name(){  
        **System**.out.printf("private method inside Child class in Java %n");  
    }  
    
    @**Override**  
    **public** **void** normal(){  
        **System**.out.println("non private overridden method from Child class ");  
    }  
    
}  
  
Output  
**private** method inside Parent **class** in Java  
non **private** overridden method from Child **class**

This example has two class Parent and Child each contains two method with same name and same signtuare, one of them is [private method](http://javarevisited.blogspot.sg/2012/03/private-in-java-why-should-you-always.html) and other is non private, public in this case. On constructor of Parent class we call both private and non private method and Output shows that overridding only applies in case of non private method. By the way calling overriden method from constructor is considering as bad practice and I have just shown here to demonstrate that we can not override private method in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html>

Can we override static method in Java - Method Hiding

**Can we override static method in Java**

No, you cannot override static method in Java because [method overriding](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) is based upon dynamic binding at runtime and static methods are bonded using [static binding](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time. Though you can declare a method with same name and method signature in sub class which does look like you can override static method in Java but in reality that is method hiding. Java won't resolve method call at runtime and depending upon type of Object which is used to call [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html), corresponding method will be called. It means if you use Parent class's type to call static method, original static will be called from patent class, on ther other hand if you use Child class's type to call static method, method from child class will be called. In short you can not override static method in Java. If you use Java IDE like [Eclipse](http://javarevisited.blogspot.sg/2011/02/how-to-setup-remote-debugging-in.html) or Netbeans, they will show warning that static method should be called using class name and not by using object becaues *static method can not be overridden in Java*.

## Overriding Static method in Java - Example

[Can we override static method in Java](http://1.bp.blogspot.com/-_GCqP1vu06Q/UBaTOZM869I/AAAAAAAAAaw/ykubu9U9kK4/s1600/java_logo_50_50.jpg)In last section we saw theory that we can not override static methods in Java, static method can only be hidden in sub class. Let's see an example to test that theory which says [you can not override static method in Java](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html)

/\*\*  
 \*  
 \* Java program which demonstrate that we **can not override static method in Java**.  
 \* Had Static method can be overridden, with Super class type and sub class object  
 \* static method from sub class would be called in our example, which is not the case.  
 \* @author  
 \*/  
**public** **class** CanWeOverrideStaticMethod {  
    
    **public** **static** **void** main(**String** args[]) {  
        
        Screen scrn = **new** ColorScreen();  
        
        *//if we can  override static , this should call method from Child class*  
        scrn.show(); *//IDE will show warning, static method should be called from classname*  
        
    }    
    
}  
  
**class** Screen{  
    
    */\*  
     \* public static method which can not be overridden in Java  
     \*/*  
    **public** **static** **void** show(){  
        **System**.out.printf("Static method from parent class");  
    }  
}  
  
**class** ColorScreen **extends** Screen{  
    */\*  
     \* static method of same name and method signature as existed in super  
     \* class, this is not method overriding instead this is called  
     \* method hiding in Java  
     \*/*  
    **public** **static** **void** show(){  
        **System**.err.println("Overridden static method in Child Class in Java");  
    }  
}  
  
**Output:**  
Static method from parent **class**

This output confirms that you can not override [static method in Java](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html) and static method are bonded based upon type information and not based upon Object. had Static mehtod be overridden, method from Child class or ColorScreen would have been called.

That's all on discussion Can we override static method in Java or not. We have confirmed that no, **we can not override static method**, we can only hide static method in Java. Creating static method with same name and mehtod signature is called Method hiding in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

<http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html>

What is method overloading in Java - Example Tutorial

**What is method overloading in Java**  
Method overloading in Java is a programming concept when programmer declare two methods of same name but with different method signature e.g. change in argument list or change in type of argument. method overloading is a powerful Java programming technique to declare method which does similar performance but with different kind of input. One of the most **popular example of method overloading** is System.out.println() method which is overloaded to accept all kinds of data types in Java. You have println() method which takes String, int, float,double or even char in output. All of those methods are collectively refereed as overloaded method in Java. [Difference between method overloading and overriding](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) is also a popular [Java interview question](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html). In next section we will some important points about method overloading in Java and then a simple example of how to overload a method in Java.  
  
  
  
**Properties of method overloading in Java**

[](http://1.bp.blogspot.com/-bPQUWs5WDLA/UCJQuUMwJ0I/AAAAAAAAAbU/F_4F6uILkmc/s1600/8.jpg)

1) *Overloaded methods* are bonded using [static binding in Java](http://javarevisited.blogspot.sg/2012/03/what-is-static-and-dynamic-binding-in.html). Which occurs during compile time i.e. when you compile Java program. During compilation process, compiler bind method call to actual method.  
  
2) Overloaded methods are fast because they are bonded during compile time and no check or binding is required during runtime.  
  
3) Most important rule of method overloading in Java is that two overloaded method must have different signature.Here is an example of *What does method signature means in Java*:  
  
1) Number of argument to a method is part of method signature.  
2) Type of argument to a method is also part of method signature  
3) Order of argument also forms part of method signature provided they are of different type.  
4) return type of method is not part of method signature in Java.  
  
Method Overloading Example in Java  
Here is a list of method and there corresponding overloaded method with reason that How they are overloaded :  
  
Original method :

 public void  show(String message){

      System.out.println(message);

}

Overloaded method : number of argument is different

 public void  show(String message, boolean show){

      System.out.println(message);

}

Overloaded method : type of argument is different

 public void  show(Integer message){

      System.out.println(message);

}  
Not a Overloaded method : only return type is different

 public boolean show(String message){

      System.out.println(message);

      return false;

}

In summary **method overloading means multiple method with same name** but with different signature. remember return type is not part of method signature. method overloading is also completely different to method overriding which is similar concept and we will see in next article. That's all on What is method overloading in Java, let me know if you have any question related to How to overload a method in Java.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*