**Overloading and Overriding Interview Questions**

Questions:

1. **What is method overloading?**
2. **Can we have two methods in a class with the same name?**
3. **Why is method overloading not possible by changing the return type in java?**
4. **What is method overriding?**
5. **What is the difference between method overriding and method overloading?**
6. **Can we overload main() method?**
7. **Can we override the overloaded method?**
8. **What are the differences between method Overloading and Overriding.**
9. **Can we override static methods of a class?**

## Properties of method overloading in Java 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 the compilation process, compiler bind method calls to the 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 methods must have a different signature.Here is an example of *What does method signature means in Java*: 1) A 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) The return type of method is not part of the 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 methods with the same name** but with a different signature. remember return type is not part of method signature. method overloading is also completely different to method overriding which is a 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.

**1) What is method overloading in Java?**  
If you have two method which does same thing its better they have same name, but two method cannot have same name until you overload them. So overloading is a process of declaring two methods with same name but different method signature e.g. System.out which is object of PrintStream class has several println() method to print different data types e.g. byte, short, int, char, float and double. All of them are called overloaded method. Overloaded method calls are resolved during compile time in Java and they must have different method signatures. See [here](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) to learn more about method overloading in Java.

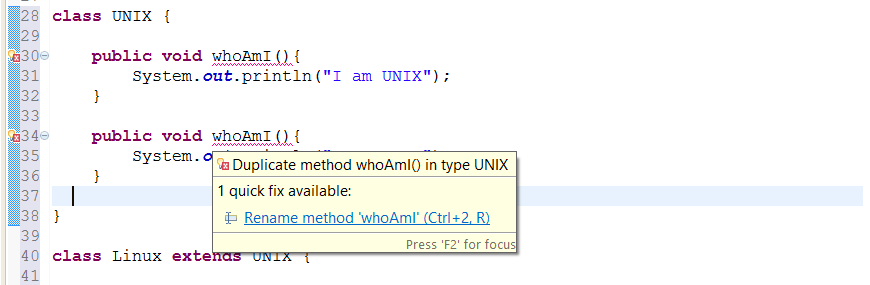
Best Example in java for Overloading;

### Commonly used methods of PrintStream class

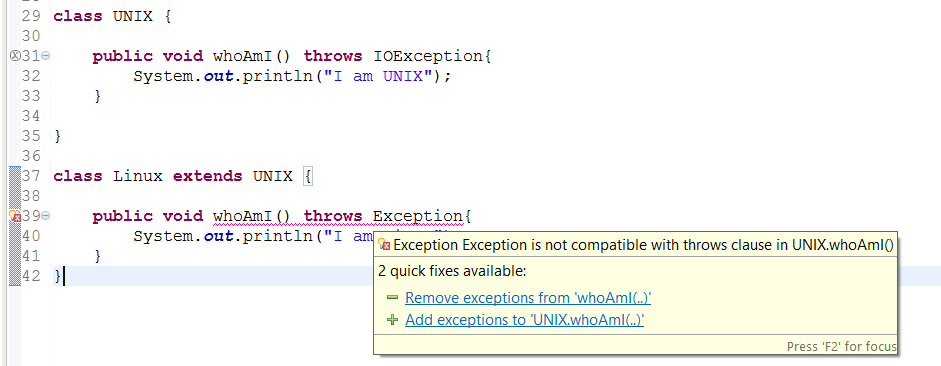
|  |
| --- |
| There are many methods in PrintStream class. Let's see commonly used methods of PrintStream class:   * **public void print(boolean b):** it prints the specified boolean value. * **public void print(char c):** it prints the specified char value. * **public void print(char[] c):** it prints the specified character array values. * **public void print(int i):** it prints the specified int value. * **public void print(long l):** it prints the specified long value. * **public void print(float f):** it prints the specified float value. * **public void print(double d):** it prints the specified double value. * **public void print(String s):** it prints the specified string value. * **public void print(Object obj):** it prints the specified object value. * **public void println(boolean b):** it prints the specified boolean value and terminates the line. * **public void println(char c):** it prints the specified char value and terminates the line. * **public void println(char[] c):** it prints the specified character array values and terminates the line. * **public void println(int i):** it prints the specified int value and terminates the line. * **public void println(long l):** it prints the specified long value and terminates the line. * **public void println(float f):** it prints the specified float value and terminates the line. * **public void println(double d):** it prints the specified double value and terminates the line. * **public void println(String s):** it prints the specified string value and terminates the line./li> * **public void println(Object obj):** it prints the specified object value and terminates the line. * **public void println():** it terminates the line only. * **public void printf(Object format, Object... args):** it writes the formatted string to the current stream. * **public void printf(Locale l, Object format, Object... args):** it writes the formatted string to the current stream. * **public void format(Object format, Object... args):** it writes the formatted string to the current stream using specified format. * **public void format(Locale l, Object format, Object... args):** it writes the formatted string to the current stream using specified format. |

## ======================================================================= 2) What is method overriding in Java? Method overriding is another way to define method with same name but different code but it must be in sub class. Overriding is based upon run-time Polymorphism as method calls are resolved at run-time depending upon actual object.  For example if a variable of type Parent holds an object of Child class then method invoked will be from child class and not parent class, provides its overridden. In order to override a method, you must follow rules of method overriding which means declaring method with same signature in sub class. See [here](http://java67.blogspot.sg/2012/08/what-is-method-overriding-in-java-example-tutorial.html) to learn more about method overriding in Java. Method Overriding Rules in Java

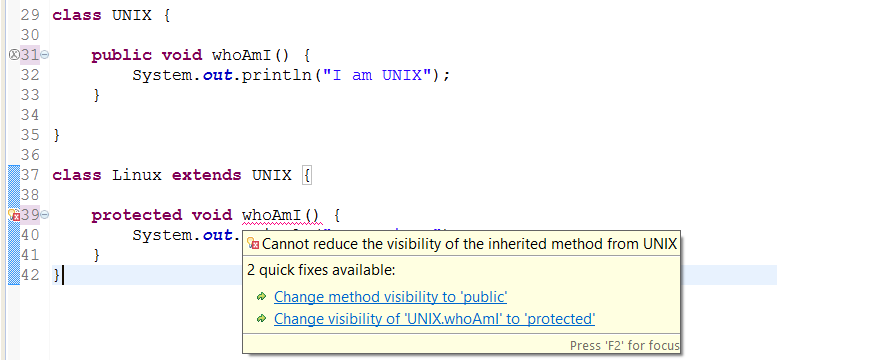
Overriding is completely different than overloading and so it's rules are also different. For terminology, original method is known as overridden method and new method is known as overriding method. Following rules must be followed to correctly override a method in Java :  
  
1) A method can only be overridden in sub class, not in same class. If you try to create two methods with same signature in one class compiler will complain about it saying *"duplicate method in type Class"*, as shown in following screenshot :

[](http://3.bp.blogspot.com/-t47anhssz58/VDahGgnVwnI/AAAAAAAACAg/5FMVdLJLmdA/s1600/Duplicate%2BMethods%2Bin%2BJava%2BClass.png)

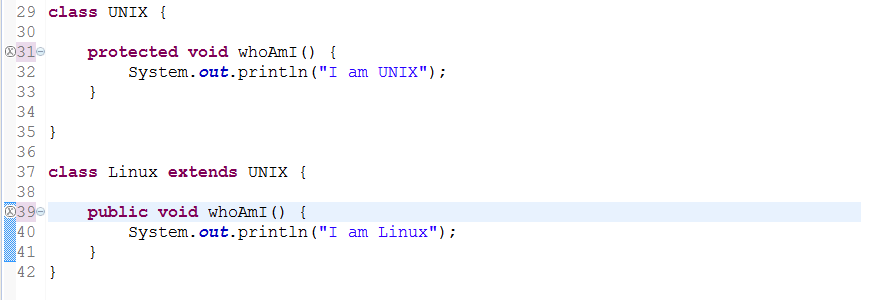
2) Overriding method cannot throw [checked Exception](http://javarevisited.blogspot.sg/2011/12/checked-vs-unchecked-exception-in-java.html) which is higher in hierarchy, than checked Exception thrown by overridden method. For example if overridden method throws IOException or [ClassNotfoundException](http://javarevisited.blogspot.sg/2011/08/classnotfoundexception-in-java-example.html), which are checked Exception, than overriding method can not throw java.lang.Exception because it comes higher in type hierarchy (it's super class of IOException and ClassNotFoundExcepiton). If you do so, compiler will catch you as seen in following image :

[](http://2.bp.blogspot.com/-T6aIcnQ2Azg/VDah4fe5XlI/AAAAAAAACAo/VGheJkOc0r4/s1600/Overriding%2BMethod%2Bcannot%2Bthrow%2Bhigher%2Bchecked%2BException%2Bin%2BJava.png)

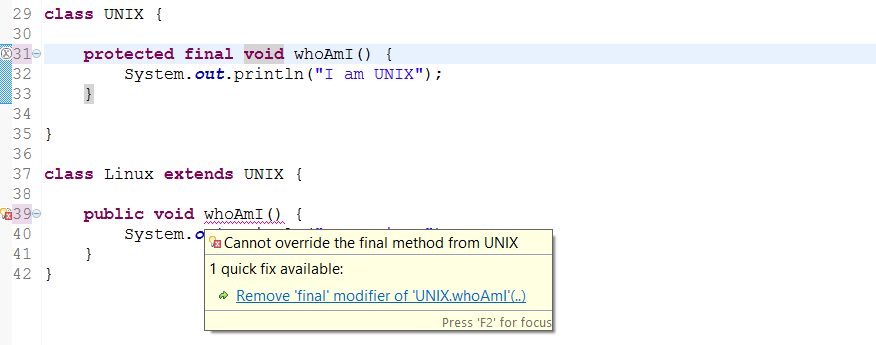
3) Overriding method can not reduce access of overridden method. It means if overridden method is defined as public than overriding method can not be protected or package private. Similarly if original method is protected then overriding method cannot be package-private. You can see what happens if you violate this rule in Java, as seen in this screenshot it will throw compile time error saying "You cannot reduce visibility of inherited method of a class".

[](http://2.bp.blogspot.com/-XFeTyJzJkDs/VDajQMf3U1I/AAAAAAAACA0/JvPw6S2ewGM/s1600/Cannot%2Breduce%2Bvisibility%2Bof%2Binherited%2Bmethod%2Bwhile%2Boverriding%2Bin%2BJava.png)

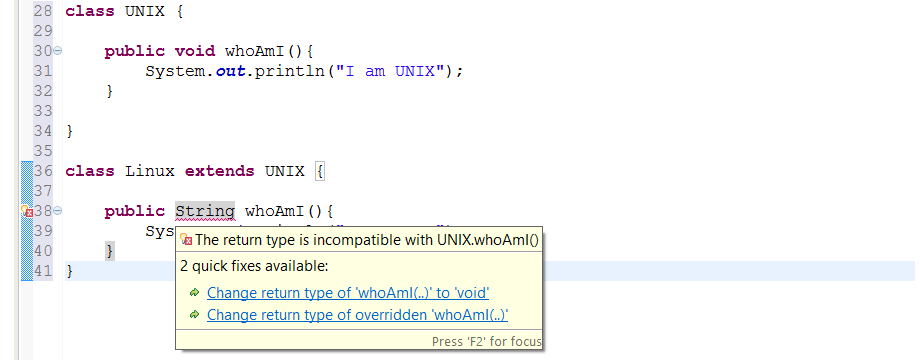
4) Overriding method can increase access of overridden method. This is opposite of earlier rule, according to this if overridden method is declared as protected than overriding method can be protected or public. Here is an example to see that it's allowed in Java :

[](http://3.bp.blogspot.com/-iBrh0VYzgC0/VDakAoMn-QI/AAAAAAAACA8/Yn-uSMFybLc/s1600/You%2Bcan%2Bincrease%2Bvisibility%2Bof%2Boverridden%2Bmethod%2Bin%2BJava.png)

5)[**private**](http://java67.blogspot.sg/2012/08/can-we-override-private-method-in-java.html)**,**[**static**](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html) and [**final method**](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) can not be overridden in Java. See other articles in this blog to learn why you cannot override private, static or final method in Java. By the way, you can hide private and static method but trying to override final method will result in compile time error "Cannot override the final method from a class" as shown in below screenshot :

[](http://1.bp.blogspot.com/-XgyBk3dQLhg/VDakx18bwiI/AAAAAAAACBE/RIw9w9P31hw/s1600/Cannot%2Boverride%2Bthe%2Bfinal%2Bmethod%2Bfrom%2Ba%2BClass%2Bin%2BJava.png)

6) Return type of overriding method must be same as overridden method. Trying to change return type of method in child class will throw compile time error "return type is incompatible with parent class method" as shown in following screenshot.

[](http://4.bp.blogspot.com/-lzbzFVMIxHk/VDafe9HmJQI/AAAAAAAACAU/81dKojfi_X0/s1600/Method%2BOverriding%2BCannot%2Bbe%2Bdone%2Bby%2Bjust%2Bchanging%2Breturn%2Btype.png)

Read more: <http://www.java67.com/2012/09/what-is-rules-of-overloading-and-overriding-in-java.html#ixzz4u24NZnEO>**Method Overriding Example in Java**

Now we know *what is method overriding in Java* and *rules of method overriding*, It's time to see an example of how to override method in Java. In this example we have used [Runnable interface](http://javarevisited.blogspot.sg/2011/02/how-to-implement-thread-in-java.html) which has an abstract run() method. We have two class Task and PeriodicTask which implements Runnable interface and override run method. For the purpose of demonstrating how method overriding works in Java we are calling run() method in same thread, which you should not, see [difference between run and start method](http://javarevisited.blogspot.sg/2012/03/difference-between-start-and-run-method.html) to know why. Because run() is overridden in two separate class, call to run() method will be resolved during runtime depending upon type of Object.

/\*\*  
 \*  
 \* Java program to demonstrate **how to override method in Java**.  
 \* Overridden method are resolved during runtime based upon type of object  
 \*  
 \* @author Javin  
 \*/  
**public** **class** CollectionTest {  
    
    **public** **static** **void** main(**String** args[]) {  
    
      **Runnable** task = **new** Task();  
      task.run(); *//call overridden method in Task*  
      
      task = **new** PeriodicTask();  
      task.run(); *//calls overridden method in PeriodicTas*  
  
    }  
    
    
}  
  
**class** Task **implements** **Runnable**{  
  
    @**Override**  
    **public** **void** run() {  
        **System**.out.println("Run method overridden in Task class");  
    }  
    
}  
  
**class** PeriodicTask **extends** Task{  
    
    @**Override**  
    **public** **void** run() {  
        **System**.err.println("overridden method run() in PeriodicTask class");  
    }  
}  
  
**Output:**  
Run method overridden in Task **class**  
overridden method run() in PeriodicTask **class**

That's all on **What is method overriding in Java**, *Rules of method overriding in Java* and an example of How to override method in Java. In summary remember to override all abstract method while extending form abstract class or implementing interface. Overridden method are also slower as compared to static and final methods because of dynamic binding but it provides you flexibility, many [popular Object oriented design principles](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) are based upon method overriding in Java.

========================

Can we override static method in Java - Method Hiding  
  
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 Objectwhich 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*.

**3) What is method hiding in Java?**  
static method cannot be overriding in Java because their method calls are resolved at compile time but it didn't prevent you from declaring method with same name in sub class. In this case we say that method in sub class has hided static method from parent class. If you have a case where variable of Parent class is pointing to object of Child class then also static method from Parent class is called because overloading is resolved at compile time. See [here](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html) to learn more about method hiding in Java.

Overriding Static method in Java - Example

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**