<http://javarevisited.blogspot.com.by/2011/11/static-keyword-method-variable-java.html>

What is Static Variable Class method and keyword in Java - Example Tutorial

What is Static in Java

Static in Java is an important keyword and used to create static method, static class and static variable in Java. Correct understanding of static keyword is required to understand and write sophisticated Java programs. Java 5 also introduced static imports along with [Autoboxing](http://javarevisited.blogspot.sg/2012/07/auto-boxing-and-unboxing-in-java-be.html), [Generics](http://javarevisited.blogspot.sg/2011/09/generics-java-example-tutorial.html), [Enum](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html) and [varargs method](http://javarevisited.blogspot.sg/2011/09/variable-argument-in-java5-varargs.html), which allows to import static members of one class or package into another using import keyword and then using them like they are member of that class. In this Java tutorial we will learn about What is is static in Java, What does it mean to be a static field, static class or method in Java and various points and issues involved around How to use static members in Java. This Java tutorial is also about how to use static keyword in Java and [where not to use static keyword](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html). Common rule is anything which you want to share between all object can be made static e.g. singleton instance of a  [Singleton Class in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html).

## What is static keyword in Java

static keyword is like any other keyword a simple keyword which can be applied to Java method , nested class or member variable inside a class. static variable in Java belong to whole Class than individual Object. Which means if Class A has a static int variable counter and A has two instance a1 and a2 both will have a static variable counter whose value would be always same except [race conditions](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html). Remember class is a blueprint while objects are real instances. So a static variable no matter whether its int, char or String will always hold same value for all instances of that class. In other words there is only one copy of static variable will be present in [Java Heap memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html), which can be accessed or altered by any object. When we *make a method static means that method belongs to class* and you can call it without creating any instance of that class. Mostly utility methods are declared as static method, so that program can call them directly by using class name and not to wait for object to be ready. One of the most popular example of static method in Java is main method and this is the reason [Why main is static in Java](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html)

### What is difference between static and non-static variable in Java

[What is static variable method and Class in Java - static keyword example](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Java member variable can be static or non-static. static variable belongs to [Java class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) while non-static variable belongs to object. static variable will keep same value for every object while value of non static variable varies from object to object. In other word one static variable is shared between all object in Java, which means in a multi-threading environment access to static variable must be [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) other wise you will get unexpected behavior. Its not suggest to use static variable in multi-threading and concurrent application because some time it create subtle bugs which is hard to find and debug. In short main difference between static and non static variable is that former belongs to class and later belongs to object.

## 10 points about static keyword in Java

In this section we will see some important properties of static variable, static method and static class in Java. We will also some [Java coding best practices](http://javarevisited.blogspot.sg/2012/08/top-10-jdbc-best-practices-for-java.html) related to static variables in Java.  
  
1) static keyword can be applied with variable, method or nested class. static keyword can not be applied on top level class. Making a [top level class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) static in Java will result in compile time error.

2) static variables are associated with class instead of object.

3) static variables in java keeps same value for every single object.

4) you can not use non-static variable inside a static method , it will result in compilation error as shown below. See [Why static variable can not be called from static method](http://javarevisited.blogspot.sg/2012/02/why-non-static-variable-cannot-be.html) for more details.

public class TradingSystem {

    String description = "electronic trading system";

    public static void main(String[] args) {

        description = "commodity trading system";

    }

}

Cannot make a static reference to the non-static field description

    at TradingSystem.main(TradingSystem.java:8)

5) Static variables are bonded using [static binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) at compile time so they are comparatively faster than there non-static counter part which were bonded during runtime.

6) Static fields are initialized at the time of [class loading in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), opposite to instance variable which is initialised when you create instance of a particular class.

7) Static keyword can also be used to create static block in Java which holds piece of code to executed [when class is loaded in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html). This is also known as static initialize block as shown in below example.

    static {

        String category = "electronic trading system";

        System.out.println("example of static block in java");

    }

Beware that if your static initialize block throws Exception than you may get [java.lang.NoClassDefFoundError](http://javarevisited.blogspot.sg/2011/06/noclassdeffounderror-exception-in.html) when you try to access the class which failed to load.

8) Static method can not be overridden in Java as they belong to class and not to object. so if you have same static  method in subclass and super class , method will be invoked based on declared type of object instead of runtime for example. [Can we override static method in Java](http://java67.blogspot.sg/2012/08/can-we-overload-static-method-in-java.html) is also a [popular Java question](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) asked in interviews.

public class TradingSystem {

    public static void main(String[] args) {

        TradingSystem system = new DirectMarketAccess();

        DirectMarketAccess dma = new DirectMarketAccess();

        // static method of Instrument class will be called,

        // even though object is of sub-class DirectMarketAccess

        system.printCategory();

        //static method of EquityInstrument class will be called  
        dma.printCategory();

    }

    public static void printCategory(){

        System.out.println("inside super class static method");

    }

}

class DirectMarketAccess extends TradingSystem{

    public static void printCategory(){

        System.out.println("inside sub class static method");

    }

}  
  
Output:  
inside super class static method  
inside sub class static method

This shows that static method can not be overridden in Java and concept of [method overloading](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) doesn't apply to static methods. Instead declaring same static method on Child class is known as **method hiding in Java**.

9. If you try to override a static method with a non-static method in sub class you will get compilation error.

10. Be careful while using static keyword in multi-threading or concurrent programming because most of the issue arise of concurrently modifying a static variable by different threads resulting in working with stale or incorrect value if not properly synchronized. most common issue is [race condition](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) which occurs due to poor synchronization or no synchronization of static variable.

**Best practices - static variable and static method in Java**

Here are some of the best practices you can follow while using static variable and method in Java.

1. Consider making a static variable final in Java to make it constant and avoid changing it from anywhere in the code. Also remember that if  you change value of static final variable in Java like in [enum String pattern](http://javarevisited.blogspot.sg/2011/12/convert-enum-string-java-example.html), you need to recompile all classes which use those variable, because static final variables are cached on client side.

2) Do not use static and non static synchronized method to protect a shared resource because both method locked on different object, which means they can be executed concurrently. See my post [Java Mistake 2 - Mixing non static and static method in Java](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html) for more details.

### 

### What is nested static class in Java

Nested static class in Java is a static member of any top level class. Though you can make any class static in Java, but you can only make nested classes i.e. class inside another class as static, you can not make any [top level class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) static. Those classes are called nested static classes. Since to create instance of any nested class you require instance of outer class but that is not required in case of static nested class in Java. You can have an instance of nested static class without any instance of outer class. Here is an example of *static nested class in Java*

public class StaticClass{  
  
 public static void main(String args[]){

StaticClass.NestedStaticClass ns = new StaticClass.NestedStaticClass();

System.out.println(ns.getDescription());

}

static class NestedStaticClass{

public String NestedStaticDescription =" Example of Nested Static Class in Java";

public String getDescription(){

return NestedStaticDescription;

}

}  
}   
  
Output:  
Example of Nested Static Class in Java

### When to use nested static class in Java

Normally we make a class static in Java when we want a single resource to be shared between all instances and normally we do this for utility classes which are required by all components and which itself doesn't have any state. Sometime interviewer ask  [when to use Singleton vs Static Class in Java](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) for those purpose,answer is that if its completely stateless and it work on provided data then you can go for static class otherwise [Singleton pattern](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) is a better choice.

### When to make a method static in Java

We can make a method static in Java in following scenario:

1) Method doesn't depends on object's state, in other words doesn't depend on any member variable and everything they need is passes as parameter to them.

2) Method belongs to class naturally can be made static in Java.

3) Utility methods are good candidate of making static in Java because then they can directly be accessed using class name without even creating any instance. Classic example is java.lang.Math

4) In various designs pattern which need a global access e.g. Singleton pattern, [Factory Pattern](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html).

### Disadvantage of static method in Java

There are certain disadvantages also if you make any method static in Java for example you can not override any static method in Java so it makes testing harder you can not replace that method with mock. Since static method maintains global state they can create subtle bug in concurrent environment which is hard to detect and fix.

### Example of static class and method in Java

Static method in Java is very popular to implement [Factory design pattern](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html). Since Generics also provides type inference during method invocation, use of static factory method to create object is popular Java idiom. JDK itself is a good example of  several static factory methods like String.valueOf().  Core Java library is also a great place to learn how to use static keyword in java with methods, variables and classes. Another popular example of static method is [main method in Java](http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html).

1. java.util.Collections has some static utility method which operates on provided collection.

2. java.lang.Math class has static method for maths operations.

3. BorderFactory has static method to control creation of object.

4. Singleton Classes like java.lang.Runtime.

Caution : Static methods should not manage or alter any state. and now a funny question what would happen if you execute following code

public class TradingSystem {

    private static String category = "electronic trading system";

    public static void main(String[] args) {

        TradingSystem system = null;

        System.out.println(system.category);

    }

will it throw [NullPointerException in Java](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html) or print "electronic trading system"

That's all on **What is static variable**, method and **nested static class in Java**. knowledge of static keyword in Java is must for any Java programmer and skill to find out when to use static variable or static method is an important skill. Incorrect and careless use of static variable and static method in Java will result in serious concurrency issues like [deadlock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html) and [race condition in Java](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html).

### Java Tutorial and fundamentals from Javarevisted

[How to Set Path for Java in Windows XP](http://javarevisited.blogspot.com/2011/10/how-to-set-path-for-java-unix-linux-and.html)  
[How to Set ClassPath for Java in Windows](http://javarevisited.blogspot.com/2011/01/how-classpath-work-in-java.html)  
[How to Convert String to Date in Java with Example](http://javarevisited.blogspot.com/2011/09/step-by-step-guide-to-convert-string-to.html)

[How to Split String in Java with Example](http://javarevisited.blogspot.com/2011/09/string-split-example-in-java-tutorial.html)  
[How to Convert String to Integer in Java](http://javarevisited.blogspot.com/2011/08/convert-string-to-integer-to-string.html)

[2 Solution of Java.lang.OutOfMemoryError](http://javarevisited.blogspot.com/2011/09/javalangoutofmemoryerror-permgen-space.html)

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

<http://javarevisited.blogspot.sg/2011/12/main-public-static-java-void-method-why.html>

Why main method is public static in Java

**Main method in Java** is the first programming method a Java programmer knows when he starts learning Java programming language.have you ever thought about **why main method in Java is public, static and void**, of-course Yes, since most of us first learn C and C++ than we move to Java in our programming path we familiar with main method but in Java main method is slightly different it doesn't return any value like in C it returns int, *main method is public static and void Why*? In this post we will try to find answer of these questions and have an idea of one of the most popular questions in Java why main method is declared Static.

### What is main method in Java?

Main method in Java is entry point for any core Java program. Remember we are not talking about Servlet, MIDlet or any other container managed Java program where life cycle methods are provided to control the execution. In core Java program, execution starts from main method when you type java main-class-name, JVM search for **public static void main(String args[])** method in that class and if it doesn't find that method it throws error **NoSuchMethodError:main** and terminates.

**Signature of main method in Java**

Main method has to strictly follow its syntax; other wise JVM will not be able to locate it and your program will not run. Here is the exact signature of main method

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

This signature is classic signature and there from start of Java but with introduction of  [variable argument or varargs in Java5](http://javarevisited.blogspot.com/2011/09/variable-argument-in-java5-varargs.html) you can also declare main method in Java using varargs syntax as shown in below example:

**public static void main(String... args)**

Remember varargs version of java main method will only work in Java 1.5 or later version. Apart from public, static and void there are certain keywords like final, synchronized and strictfp which are permitted in signature of java main method.

### Why main method is static in Java

[why main method is public static void in Java](http://javarevisited.blogspot.com/2011/10/override-hashcode-in-java-example.html)Now come to the main point *"Why main method is static in Java"*, there are quite a few reasons around but here are few reasons which make sense to me:

1. Since main method is static Java virtual Machine can call it without creating any instance of class which contains main method.

2. Since C and C++ also has similar main method which serves as entry point for program execution, following that convention will only help Java.

3. If main method were not declared static than JVM has to create instance of main Class and since constructor can be overloaded and can have arguments there would not be any certain and consistent way for **JVM to find main method in Java**.

4. Anything which is declared in [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html) comes under reference type and requires object to be created before using them but static method and static data are loaded into separate memory inside JVM called context which is created when a class is loaded. If main method is static than it will be loaded in JVM context and are available to execution.

**Why main mehtod is public in Java**

Java specifies several access modifiers e.g. private, protected and public. Any method or variable which is declared public in Java can be accessible from outside of that class. Since main method is public in

Java, JVM can easily access and execute it.

**Why main method is void in Java**

Since main method in Java is not supposed to return any value, its made void which simply means main is not returning anything.

**Summary:**

1. Main method must be declared **public, static and void in Java** otherwise JVM will not able to run Java program.

2. JVM throws **NoSuchMethodException:main** if it doesn't find main method of predefined signature in class which is provided to Java command. E.g. if you run java Helloworld than JVM will search for public static void main String args[]) method in HelloWorld.class file.

3. Main method is entry point for any Core Java program. Execution starts from main method.

4. Main method is run by a special thread called ["main" thread in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html). Your Java program will be running until your main thread is running or any non-daemon thread spawned from main method is running.

5. When you see **"Exception in Thread main”** e.g.

**Exception in Thread main: Java.lang.NullPointerException** it means Exception is thrown inside main thread.

6. You can declare main method using varargs syntax from Java 1.5 onwards e.g.

**public static void main(String... args)**

7. Apart from static, void and public you can use final, synchronized and strictfp modifier in signature of main method in Java.

8. Main method in Java can be overloaded like any other method in Java but JVM will only call main method with specified signature specified above.

9. You can use throws clause in signature of main method and can throw any checked or unchecked Exception.

10. [Static initializer block](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) is executed even before JVM calls main method. They are executed when a Class is loaded into Memory by JVM.

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

<http://javarevisited.blogspot.sg/2012/02/why-non-static-variable-cannot-be.html>

Why non-static variable cannot be referenced from a static context?

"**non-static variable cannot be referenced from a static context**" is biggest nemesis of some one who has just

started programming and that too in Java. Since [main method in java](http://javarevisited.blogspot.com/2011/12/main-public-static-java-void-method-why.html) is most popular method among all beginners and

they try to put program code there they face "*non-static variable cannot be referenced from a static context*" **compiler error** when they  try to access a non static member variable inside main in Java which is static. if you want to know

why main is declared static in Java see the link.

public class **StaticTest** {

    private int count=0;

    public static void main(String args[]) throws IOException {

        count++; //**compiler error: non-static variable count cannot be referenced from a static context**

    }

}

## Why non static variable can not be called from static method

[non-static variable cannot be referenced from a static context](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html)Now before finding answer of compiler error "non-static variable cannot be referenced from a static context", let's have a quick revision of static. [Static variable in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) belongs to Class and its **value remains same for all instance**. static variable initialized when class is loaded into [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) on the other hand instance variable has different value for each instances and they get created when instance of an object is created either by using new() operator or using reflection like Class.newInstance(). So if you try to access a non static variable without any instance compiler will complain because **those variables are not yet created** and they don't have any existence until an instance is created and they are associated with any instance. So in my opinion only reason which make sense to disallow [non static or instance variable](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) inside static context is non existence of instance.

In summary since code in static context can be run even without creating any instance of class, it does not make sense asking value for an specific instance which is not yet created.

## How to access non static variable inside static method or block

You can still access any non static variable inside any static method or block by creating an instance of [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html)

and using that instance to reference instance variable. This is the only legitimate way to access non static variable

on static context. here is a code **example of accessing non static variable inside static context**:

public class **StaticTest** {

    private int count=0;

    public static void main(String args[]) throws IOException {

        StaticTest test = new StaticTest(); *//accessing static variable by creating an instance of class*

        test.count++;

    }

}

So next time if you get compiler error “**non-static variable cannot be referenced from a static context”** access static member by creating an instance of Class. Let me know if you find any other reason on why non-static variable cannot be referenced from a static context.

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

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

Can we overload static method in Java Program – Example

**Overloading static method In Java**

Yes, we can overload static method in Java. In terms of [method overloading](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html) static method are just like normal methods and in order to overload [static method](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) you need to provide another static method with same name but different method signature. Static overloaded method are resolved using [Static Binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) during compile time. Overloading method in Java is completely different than overriding method and as discussed in our last article we [can not override static method in Java](http://java67.blogspot.sg/2012/08/can-we-override-static-method-in-java.html) but we can certainly overload static method in Java. Here is an example which confirms that *we can overload static method in Java*:

## Overloading Static method in Java - example

[Can we overload static method in Java with Example](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html)In this example we have a static method called greet(String name) which takes a [String](http://javarevisited.blogspot.sg/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) argument as name and print a default greeting message as "Hello John". Now to show that we can overload static method in Java I have provided another static method with same name but different [method signature](http://java67.blogspot.sg/2012/08/what-is-method-overloading-in-java-example.html) which not only takes name of person to greet but also greeting message e.g. Good Morning, Good Evening etc.

/\*\*   
 \* **Java program to show that we can overload static method in Java**.  
 \*/  
**public** **class** StaticOverloadingTest {  
    
    **public** **static** **void** main(**String** args[]) {  
        greet("John"); *//will call static method with one String argument*  
        greet("John", "Good Morning"); *//overloaded static method will be call*  
  
    }  
    
    */\*  
     \* static method which will be overloaded  
     \*/*  
    **public** **static** **void** greet(**String** name){  
        **System**.out.println("Hello " + name);  
    }  
    
    */\*  
     \* Another static method which overload above Hello method  
     \* This shows that we can overload static method in Java  
     \*/*  
    **public** **static** **void** greet(**String** name, **String** greeting){  
        **System**.out.println(greeting + " " + name);  
    }  
  
  
}  
**Output**  
Hello John  
Good Morning John

That's all on How can we overload static method in Java. In summary, Don't confuse between [method overloading and method overriding](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). In short, you can overload static method in Java but you can not override static method in Java.

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

<http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html>

Mixing static and non static synchronized method - Java mistake 2

**Using static and non static synchronized method** for protecting shared resource is another Java mistake we are going to discuss in this part of  our series “learning from mistakes in Java”. In last article we have seen [why double and float should not be used for monetary calculation](http://javarevisited.blogspot.com/2012/02/java-mistake-1-using-float-and-double.html) , In this tutorial we will find out why using static and non static synchronized method together for protecting same shared resource is not advisable.

I have seen some times Java  programmer mix [static synchronized method](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) and instance synchronized method to protect same shared resource. They either don't know or failed to realize that **static synchronized** and **non static synchronized** method **lock on two different object** which breaks purpose of synchronizing shared resource as two thread can concurrently execute these two method breaking mutual exclusive access, which can corrupt status of mutable object or even cause subtle [race condition in Java](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) or even more horrible [deadlock in java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html).

## Static and non static synchronized method Java

[static and non static synchronized method in java](http://javarevisited.blogspot.com/2011/09/invokeandwait-invokelater-swing-example.html)For those who are not familiar [static](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) synchronized method locked on class object e.g. for string class its String.class while instance synchronized method locks on current instance of Object denoted by “[this” keyword in Java](http://javarevisited.blogspot.com/2012/01/this-keyword-java-example-tutorial.html). Since both of these object are different they have different lock so while one thread is executing static synchronized method , other [thread in java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) doesn’t need to wait for that thread to return instead it will acquire separate lock denoted byte .class literal and enter into static synchronized method. This is even a [popular multi-threading interview questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) where interviewer asked on which lock a particular method gets locked, some time also appear in Java test papers.

Bottom line is that  *never mix static and non static synchronized method for protecting same resource*.

## Example of Mixing instance and static synchronized methods

Here is an **example of multithreading code which is using static and non static synchronized method** to protect same shared resource:

**public** **class** SynchornizationMistakes {  
    **private** static int [count](http://www.php.net/count) = 0;  
    
    *//locking on this object lock*  
    **public** synchronized int getCount(){  
        return [count](http://www.php.net/count);  
    }  
    
    *//locking on .class object lock*  
    **public** static synchronized void increment(){  
        [count](http://www.php.net/count)++;  
    }  
      
}

here shared count is not accessed in mutual exclusive fashion which may result in passing incorrect count to caller of getCount() while another thread is incrementing count using static increment() method.

That’s all on this part of learning from mistakes in Java. Now we know that **static and non static synchronized method are locked on different locks** and should not be used to protect same shared object.

Other **Java thread tutorials** you may like:

[When to use Thread or Runnable interface in Java?](http://javarevisited.blogspot.com/2012/01/difference-thread-vs-runnable-interface.html)

[What does Volatile keyword do in Java](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html)

[How to Stop Thread in Java](http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html)

[How to write thread safe code in Java](http://javarevisited.blogspot.com/2012/01/how-to-write-thread-safe-code-in-java.html)

[Why wait and notify method are called from synchronized method or block](http://javarevisited.blogspot.com/2011/05/wait-notify-and-notifyall-in-java.html)

[Why wait and notify are defined in Object class](http://javarevisited.blogspot.com/2012/02/why-wait-notify-and-notifyall-is.html)

[Difference between wait and sleep in Java](http://javarevisited.blogspot.com/2011/12/difference-between-wait-sleep-yield.html)

[How to solve Producer consumer problem in Java using BlockingQueue](http://javarevisited.blogspot.com/2012/02/producer-consumer-design-pattern-with.html)

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

<http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html>

10 Singleton Pattern Interview Questions in Java – Answered

Singleton design pattern is one of the most common patterns you will see in Java applications and it’s also used heavily in core Java libraries. Questions from Singleton pattern is very common in Java interviews and good knowledge of how to implement Singleton pattern certainly help.This is also one of my favorite [design pattern interview question](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html) and has lots of interesting follow-up to dig into details , this not only check the knowledge of design pattern but also check coding, multithreading aspect which is very important while working for a real life application. In this post have listed some of the most common question asked on Singleton pattern during a Java Interview. I have not provided the answers of these questions as they are easily available via google search but if you guys need I can try to modify this tutorial to include answers as well. As promised earlier and having received lot of request for providing answers of these question, I have decided to update this post along with answers. By the way if you are preparing for interview on Java technology than you can check my collection on [Java interview questions](http://javarevisited.blogspot.sg/2011/04/top-20-core-java-interview-questions.html) and [multi-threading interview questions](http://javarevisited.blogspot.sg/2011/07/java-multi-threading-interview.html). There are lot of resources in Javarevisited which can help you in your interview preparation. On the other hand if you are more interested on design pattern tutorials than you can check my post on [builder design pattern](http://javarevisited.blogspot.com/2012/06/builder-design-pattern-in-java-example.html) 

## 10 Interview question on Singleton Pattern in Java

Here is my collection of interview questions based upon Singleton design pattern. They are collected from various Java interviews and highlights key aspects of pattern and where it is broken, if you know how to create thread-safe singletons and different ways to implement this pattern, and pros and cons of each approach. Questions starts with :

### What is Singleton class? Have you used Singleton before?

Singleton is a class which has only one instance in whole application and provides a getInstance() method to access the singleton instance. There are many classes in JDK which is implemented using Singleton pattern like java.lang.Runtime which provides getRuntime() method to get access of it and used to get [free memory and total memory in Java](http://javarevisited.blogspot.sg/2012/01/find-max-free-total-memory-in-java.html).

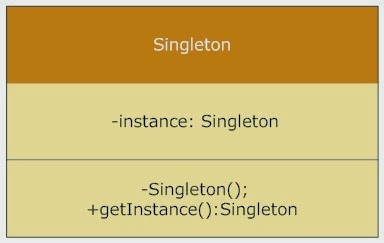
### Which classes are candidates of Singleton? Which kind of class do you make Singleton in Java?

Here they check whether candidate has enough experience on usage of singleton or not. Does he is familiar of advantage/disadvantage or alternatives available for singleton in Java or not.

Answer : Any class which you want to be available to whole application and whole only one instance is viable is candidate of becoming Singleton. One example of this is Runtime class , since on whole java application only one runtime environment can be possible making Runtime Singleton is right decision. Another example is a utility classes like Popup in GUI application, if you want to show popup with message you can have one PopUp class on whole GUI application and anytime just get its instance, and call show() with message.

### Can you write code for getInstance() method of a Singleton class in Java?

Most of the java programmer fail here if they have mugged up the singleton code because you can ask lots of follow-up question based upon the code they have written. I have seen many programmer write Singleton getInstance() method with double checked locking but they are not really familiar with the caveat associated with double checking of singleton prior to Java 5.  
  
Answer : Until asked don’t write code using double checked locking as it is more complex and chances of errors are more but if you have deep knowledge of double checked locking, [volatile variable](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html) and lazy loading than this is your chance to shine. I have shared code examples of writing singleton classes using enum, using static factory and with double checked locking in my recent post [Why Enum Singletons are better in Java](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html), please see there.

[](http://2.bp.blogspot.com/-vIw-2sGSrko/VJmIXk6xfnI/AAAAAAAACP4/clWZsrAY3Ro/s1600/Singleton+Design+Pattern+in+Java+2.jpg)

### Is it better to make whole getInstance() method synchronized or just critical section is enough? Which one you will prefer?

This is really nice question and I mostly asked to just quickly check whether candidate is aware of performance trade off of unnecessary locking or not. Since locking only make sense when we need to create instance and rest of the time its just read only access so locking of critical section is always better option. read more about synchronization on [How Synchronization works in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)

Answer : This is again related to double checked locking pattern, well synchronization is costly and when you apply this on whole method than call to getInstance() will be synchronized and contented. Since synchronization is only needed during initialization on singleton instance, to prevent creating another instance of Singleton, It’s better to only synchronize critical section and not whole method. Singleton pattern is also closely related to [factory design pattern](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) where getInstance() serves as static factory method.

### What is lazy and early loading of Singleton and how will you implement it?

This is another great Singleton interview question in terms of understanding of concept of loading and cost associated with class loading in Java. Many of which I have interviewed not really familiar with this but its good to know concept.  
  
Answer : As there are many ways to implement Singleton like using double checked locking or Singleton class with [static](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) [final](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) instance initialized during class loading. Former is called lazy loading because Singleton instance is created only when client calls getInstance() method while later is called early loading because Singleton instance is created when class is loaded into memory.

### Give me some examples of Singleton pattern from Java Development Kit?

This is open question to all, please share which classes are Singleton in JDK. Answer to this question is java.lang.Runtime

Answer : There are many classes in Java Development Kit which is written using singleton pattern, here are few of them:

1. Java.lang.Runtime with getRuntime() method
2. Java.awt.Toolkit with getDefaultToolkit()
3. Java.awt.Desktop with getDesktop()

### What is double checked locking in Singleton?

One of the most hyped question on Singleton pattern and really demands complete understanding to get it right because of Java Memory model caveat prior to Java 5. If a guy comes up with a solution of using [volatile keyword](http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html) with Singleton instance and explains it then it really shows it has in depth knowledge of Java memory model and he is constantly updating his Java knowledge.   
  
Answer : Double checked locking is a technique to prevent creating another instance of Singleton when call to getInstance() method is made in multi-threading environment. In Double checked locking pattern as shown in below example, singleton instance is checked two times before initialization. See [here](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html) to learn more about double-checked-locking in Java.

public static **Singleton** getInstance(){

**if**(**\_INSTANCE** == **null**){

synchronized(**Singleton**.class){

//double checked locking - because second check of Singleton instance with lock

**if**(**\_INSTANCE** == **null**){

**\_INSTANCE** **=** **new** **Singleton**();

}

}

}

**return** **\_INSTANCE**;

}

Double checked locking should only be used when you have requirement for lazy initialization otherwise [use Enum to implement singleton](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html) or simple static final variable.

### How do you prevent for creating another instance of Singleton using clone() method?

This type of questions generally comes some time by asking how to break singleton or when Singleton is not Singleton in Java.

Answer : Preferred way is not to implement Cloneable interface as why should one wants to create clone() of Singleton and if you do just throw Exception from clone() method as “Can not create clone of Singleton class”.

### How do you prevent for creating another instance of Singleton using reflection?

Open to all. In my opinion throwing exception from constructor is an option.   
Answer: This is similar to previous interview question. Since constructor of Singleton class is supposed to be private it prevents creating instance of Singleton from outside but [Reflection can access private fields and methods](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html), which opens a threat of another instance. This can be avoided by throwing Exception from constructor as “Singleton already initialized”

### How do you prevent for creating another instance of Singleton during serialization?

Another great question which requires knowledge of [Serialization in Java](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html) and how to use it for persisting Singleton classes. This is open to you all but in my opinion use of readResolve() method can sort this out for you.  
Answer: You can prevent this by using readResolve() method, since during serialization readObject() is used to create instance and it return new instance every time but by using readResolve you can replace it with original Singleton instance. I have shared code on how to do it in my post Enum as Singleton in Java. This is also one of the reason I have said that use Enum to create Singleton because serialization of enum is taken care by JVM and it provides guaranteed of that.

### When is Singleton not a Singleton in Java?

There is a very good article present in Sun's Java site which discusses various scenarios when a Singleton is not really remains Singleton and multiple instance of Singleton is possible. Here is the link of that article <http://java.sun.com/developer/technicalArticles/Programming/singletons/>  
  
  
Apart from these questions on Singleton pattern, some of my reader contribute few more questions, which I included here. Thank you guys for your contribution.

### Why you should avoid the singleton anti-pattern at all and replace it with DI?

Answer : Singleton Dependency Injection: every class that needs access to a singleton gets the object through its constructors or with a DI-container.

### Why Singleton is Anti pattern

With more and more classes calling getInstance() the code gets more and more tightly coupled, monolithic, not testable and hard to change and hard to reuse because of not configurable, hidden dependencies. Also, there would be no need for this clumsy double checked locking if you call getInstance less often (i.e. once).

### How many ways you can write Singleton Class in Java?

Answer : I know at least four ways to implement Singleton pattern in Java

1. Singleton by synchronizing getInstance() method
2. Singleton with public static final field initialized during class loading.
3. Singleton generated by static nested class, also referred as Singleton holder pattern.
4. From Java 5 on-wards using Enums

### How to write thread-safe Singleton in Java?

Answer : Thread safe Singleton usually refers to write [thread safe code](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) which creates one and only one instance of Singleton if called by multiple thread at same time. There are many ways to achieve this like by using double checked locking technique as shown above and by using [Enum](http://javarevisited.blogspot.in/2011/08/enum-in-java-example-tutorial.html) or Singleton initialized by class loader.

At last few more questions for your practice, contributed by Mansi, Thank you Mansi  
  
14) Singleton vs Static Class?  
15) When to choose Singleton over Static Class?  
16) Can you replace Singleton with Static Class in Java?  
17) Difference between Singleton and Static Class in java?  
18) Advantage of Singleton over Static Class?  
  
  
I have covered answers of couple of these questions in my post, [Singleton vs Static Class in Java - Pros and Cons](http://javarevisited.blogspot.com/2013/03/difference-between-singleton-pattern-vs-static-class-java.html).  If you like to read more Java interview questions you can have a look on some of my favorites below

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

<http://javarevisited.blogspot.com.by/2013/03/difference-between-singleton-pattern-vs-static-class-java.html>

Difference between Singleton Pattern vs Static Class in Java

Singleton pattern  vs Static Class (a class, having all static methods) is another interesting questions, which I missed while blogging about [Interview questions on Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html). Since both Singleton pattern and static class provides good accessibility, and they share some similarities e.g. both can be used without creating object and both provide only one instance, at very high level it looks that they both are intended for same task. Because of high level similarities, interviewer normally ask questions like, *Why you use Singleton instead of Static Methods,* or Can you replace Singleton with static class, and what are differences between [Singleton pattern](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html) and [static in Java](http://javarevisited.blogspot.sg/2012/03/mixing-static-and-non-static.html). In order to answer these question, it’s important to remember fundamental difference between Singleton pattern and static class, former gives you an [Object](http://javarevisited.blogspot.com/2012/12/what-is-object-in-java-or-oops-example.html), while later just provide static methods. Since an object is always much more capable than a method, it can guide you when to use Singleton pattern vs static methods.

In this Java article we will learn, where to use Singleton pattern in Java, and when static class is better alternative. By the way, JDK has examples of both singleton and static, and that too very intelligently e.g. java.lang.Math is a [final class](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) with full of [static methods](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html), on the other hand java.lang.Runtime is a Singleton class in Java. For those who are not familiar with Singleton design pattern or static class, static class is a [Java class](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html), which only contains static methods, good examples of static class is java.lang.Math,which contains lots of utility methods for various maths function e.g. sqrt(). While [Singleton classes](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) are those, which has only one instance during application life cycle like java.lang.Runtime.

## When to use Static Class in place of Singleton in Java

[When to choose Singleton pattern vs static in Java](http://3.bp.blogspot.com/-K6q0DQ1v-tw/TWu8owBtc2I/AAAAAAAAADA/oBoHDBiJ8ag/s1600/17.jpg)Indeed there are some situations, where static classes makes sense than Singleton. Prime example of this is java.lang.Math which is not Singleton, instead a class with all static methods. Here are few situation where I think using static class over Singleton pattern make sense:

1) If your Singleton is not maintaining any state, and just providing global access to methods, than consider using static class, as static methods are much faster than Singleton, because of [static binding](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) during compile time. But remember its not advised to maintain state inside static class, especially in concurrent environment, where it could lead subtle [race conditions](http://javarevisited.blogspot.com/2012/02/what-is-race-condition-in.html) when modified parallel by multiple threads without adequate synchronization.

You can also choose to use static method, if you need to combine bunch of utility method together. Anything else, which requires singles access to some resource, should use Singleton design pattern.

## Difference between Singleton vs Static in Java

This is answer of our second interview question about Singleton over static. As I said earlier, fundamental difference between them is, one represent object while other represent a method. Here are few more differences between static and singleton in Java.

1) Static class provides better performance than Singleton pattern, because static methods are bonded on compile time.

2) One more difference between Singleton and static is, ability to override. Since [static methods in Java cannot be overridden](http://java67.blogspot.com/2012/08/can-we-override-static-method-in-java.html), they leads to inflexibility. On the other hand, you can override methods defined in Singleton class by extending it.

3) Static classes are hard to mock and consequently hard to test than Singletons, which are pretty easy to mock and thus easy to test. It’s easier to write [JUnit test](http://javarevisited.blogspot.com/2013/03/how-to-write-unit-test-in-java-eclipse-netbeans-example-run.html) for Singleton than static classes, because you can pass mock object whenever Singleton is expected, e.g. into constructor or as method arguments.

4) If your requirements needs to maintain state than Singleton pattern is better choice than static class, because

maintaining state in later case is nightmare and leads to subtle bugs.

5) Singleton classes can be [lazy loaded](http://javarevisited.blogspot.sg/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) if its an heavy object, but static class doesn't have such advantages and always eagerly loaded.

6) Many [Dependency Injection framework](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html) manages Singleton quite well e.g. Spring, which makes using them very easy.

These are some differences between static class and singleton pattern, this will help to decide between two, which situation arises. In next section we will when to choose Singleton pattern over static class in Java.

## Advantage of Singleton Pattern over Static Class in Java

Main advantage of Singleton over static is that former is more object oriented than later. With Singleton, you can use [Inheritance](http://javarevisited.blogspot.com/2012/10/what-is-inheritance-in-java-and-oops-programming.html) and [Polymorphism](http://javarevisited.blogspot.com.au/2011/08/what-is-polymorphism-in-java-example.html) to extend a base class, implement an interface and capable of providing different implementations. If we talk about java.lang.Runtime, which is a Singleton in Java, call to getRuntime() method return different implementations based on different JVM, but guarantees only one instance per JVM, had java.lang.Runtime an static class, it’s not possible to return different implementation for different JVM.

That’s all on difference between Singleton and static class in Java. When you need a class with full OO capability , chose Singleton, while if you just need to store bunch of static methods together, than use static class.

Other **Java Design Pattern Tutorials** from Javarevisited Blog

[When to use Builder design pattern in Java](http://javarevisited.blogspot.com/2012/06/builder-design-pattern-in-java-example.html)

[A Real life example of Observer Pattern in Java](http://javarevisited.blogspot.in/2011/12/observer-design-pattern-java-example.html)

[How to use Decorator pattern in Java](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html)

[Difference between Factory and Abstract Factory pattern in Java](http://javarevisited.blogspot.sg/2013/01/difference-between-factory-and-abstract-factory-design-pattern-java.html)

[10 SOLID and Object Oriented design principles Java Programmer should know](http://javarevisited.blogspot.de/2012/03/10-object-oriented-design-principles.html)

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

<http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html>

What is final in Java? Final variable , Method and Class Example

**Final in java** is very important keyword and can be applied to class, method, and variables in Java. In this java final tutorial we will see **what is final keyword in Java**, *what does it mean by making final variable*, final method and final class in java and *what are primary benefits of using final keywords in Java* and finally some examples of final in Java. Final is often used along with [static keyword in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) to make static final constant and you will see how final in Java can increase performance of Java application.

## Example of Final variable, Final method and Class in Java

### What is final keyword in Java?

[Final keyword, final variable, final method and class in java example](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html)**Final is a keyword** or reserved word in java and can be applied to member variables, methods, class and local variables in Java. Once you make a reference final you are not allowed to change that reference and compiler will verify this and raise **compilation error** if you try to re-initialized **final variables in java**.

**What is final variable in Java?**

Any variable either member variable or local variable (declared inside method or block) modified by final keyword is called final variable. Final variables are often declare with static keyword in java and treated as constant. Here is an example of final variable in Java

public static final String LOAN = "loan";

LOAN = new String("loan") **//invalid compilation error**

Final variables are by default read-only.

**What is final method in Java**

Final keyword in java can also be applied to methods. A java method with final keyword is called final method and it can not be overridden in sub-class. You should make a method final in java if you think it’s complete and its behavior should remain constant in sub-classes. Final methods are faster than non-final methods because they are not required to be resolved during run-time and they are bonded on compile time. Here is an *example of final method in Java***:**

class **PersonalLoan**{

 public final String getName(){

     return "personal loan";

 }

}

class **CheapPersonalLoan** extends **PersonalLoan**{

    @Override

    public final String getName(){

        return "cheap personal loan"; **//compilation error: overridden method is final**

    }

}

### What is final Class in Java

Java class with final modifier is called final [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html). Final class is complete in nature and can not be sub-classed or inherited. Several classes in Java are final e.g. String, Integer and other wrapper classes. Here is an *example of final class in java*

final class **PersonalLoan**{

}

class **CheapPersonalLoan** extends **PersonalLoan**{  //compilation error: cannot inherit from final class

}

### Benefits of final keyword in Java

Here are few benefits or advantage of using final keyword in Java:

1. Final keyword improves performance. Not just JVM can cache **final variable** but also application can cache frequently use final variables.

2. Final variables are safe to share in [multi-threading](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) environment without additional synchronization overhead.

3. **Final keyword** allows [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) to optimized method, variable or class.

### Final and Immutable Class in Java

Final keyword helps to write immutable class. Immutable classes are the one which can not be modified once it gets created and String is primary example of immutable and final class which I have discussed in detail on [Why String is final or immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html). Immutable classes offer several benefits one of them is that they are effectively read-only and can be safely shared in between multiple threads without any synchronization overhead. You can not make a class immutable without making it final and hence final keyword is required to make a class immutable in java.

### Example of Final in Java

Java has several system classes in JDK which are final, some example of final classes are String, Integer, Double and other wrapper classes. You can also use final keyword to make your code better whenever it required. See relevant section of **java final tutorial** for *example of final variable***,** *final method* and *final class in Java*.

### Important points on final in Java

1. **Final keyword** can be applied to member variable, local variable, method or [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html).

2. **Final member variable** must be initialized at the time of declaration or inside constructor, failure to do so will result in compilation error.

3. You can not reassign value to *final variable in Java*.

4. **Local final variable** must be initializing during declaration.

5. Only final variable is accessible inside anonymous class in Java.

6. **Final method** can not be [overridden in Java](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html).

7. **Final class** can not be inheritable in Java.

8. **Final** is different than **finally** keyword which is used on [Exception handling in Java](http://javarevisited.blogspot.com/2011/12/checked-vs-unchecked-exception-in-java.html).

9. Final should not be confused with finalize() method which is declared in object class and called before an object is garbage collected by JVM.

10. All variable declared inside java interface are implicitly final.

11. **Final and abstract** are two opposite keyword and a final class can not be [abstract in java](http://javarevisited.blogspot.com/2010/10/abstraction-in-java.html).

12. Final methods are bonded during compile time also called static binding.

13. *Final variables* which is not initialized during declaration are called blank final variable and must be initialized on all constructor either explicitly or by calling this(). Failure to do so compiler will complain as "*final variable (name) might not be initialized*".

14. Making a class, method or variable final in Java helps to improve performance because JVM gets an opportunity to make assumption and optimization.

15. As per Java code convention **final variables are treated as constant** and written in all Caps e.g.

private final int COUNT=10;

16. Making a collection reference variable final means only reference can not be changed but you can add, remove or change object inside collection. For example:

private final List Loans = new ArrayList();

list.add(“home loan”);  **//valid**

list.add("personal loan"); **//valid**

loans = new Vector();  **//not valid**

That’s all on **final in Java**. We have seen what final variable, final method is and final class in Java and what does those mean. In Summary whenever possible start using final in java it would result in better and faster code.

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

<http://java67.blogspot.com.by/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.

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