<http://javarevisited.blogspot.com.by/2011/04/top-10-java-serialization-interview.html>

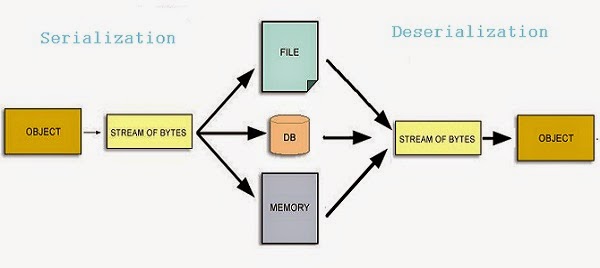
Top 10 Java Serialization Interview Questions and Answers

**What is Serialization in Java**

Java Serialization is one of important concept but it’s been rarely used as persistence solution and developer mostly overlooked Java serialization API. As per my experience Java Serialization is quite an important topic in any core Java interview, In almost all the interview I have faced there is one or two Java serialization questions and I have seen interview where after few question on serialization candidate start feeling uncomfortable because of lack of experience in this area. They don’t know How to serialize object in Java or they are not familiar with any Java Serialization example to explain, forget about questions like [Difference between transient and volatile variable](http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html) or [Difference between Externalizable and Serializable in Java](http://javarevisited.blogspot.sg/2012/01/serializable-externalizable-in-java.html). In this article we will question from both beginner and advanced level, which can be equally beneficial to freshers, new comers and senior Java developers with some years of Java development experience.

## 10 Interview questions on Serialization in Java

Most commercial project uses either database or memory mapped file or simply flat file for there persistence requirement and only few of them rely on serialization process in Java. Anyway this post is not a Java serialization tutorial or how to serialize object in java but about interview questions around serialization mechanism and Serialization API, Which is worth to have a look before going for any Java or [J2EE interview](http://javarevisited.blogspot.sg/2011/09/spring-interview-questions-answers-j2ee.html) and surprising yourself with some unknown contents. for those who are not familiar about java Serialization "Java serialization is the process which is used to serialize object in java by storing object’s state into a file with extension .ser and recreating object's state from that file, this reverse process is called deserialization.



The Java Serialization API provides a standard mechanism for developers to handle object serialization using Serializable and Externalizable interface. By the way this article is in continuation of my previous article [Top 20 design pattern interview questions](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html),  and [10 Interview questions on Singleton Pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) So here we go.

### What is Serialization in Java

Object Serialization in Java is a process used to convert Object into a binary format which can be persisted into disk or sent over network to any other running [Java virtual machine](http://javarevisited.blogspot.sg/2011/11/hotspot-jvm-options-java-examples.html); the reverse process of creating object from binary stream is called deserialization in Java. Java provides Serialization API for serializing and deserializing object which includes java.io.Serializable, java.io.Externalizable, ObjectInputStream and ObjectOutputStream etc. Java programmers are free to use default Serialization mechanism which Java uses based upon structure of class but they are also free to use there own custom binary format, which is often advised as Serialization best practice, Because serialized binary format becomes part of Class's exported API and it can potentially break [Encapsulation in Java](http://javarevisited.blogspot.sg/2012/03/what-is-encapsulation-in-java-and-oops.html) provided by private and [package-private fields](http://javarevisited.blogspot.sg/2012/05/how-to-access-private-field-and-method.html). This pretty much answer the question What is Serialization in Java.

### How to make a Java class Serializable?

Making a class Serializable in Java is very easy, Your Java class just needs to implements java.io.Serializable interface and JVM will take care of serializing object in default format. Decision to making a [Class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) Serializable should be taken concisely because though near term cost of making a Class Serializable is low, long term cost is substantial and it can potentially limit your ability to further modify and change its implementation because like any public API, serialized form of an object becomes part of public API and when you change structure of your class by implementing addition interface, adding or removing any field can potentially break default serialization, this can be minimized by using a custom binary format but still requires lot of effort to ensure backward compatibility. One example of How Serialization can put constraints on your ability to change class is SerialVersionUID. If you don't explicitly declare SerialVersionUID then JVM generates its based upon structure of class which depends upon interfaces a class implements and several other factors which is subject to change. Suppose you implement another interface than [JVM](http://javarevisited.blogspot.sg/2011/12/jre-jvm-jdk-jit-in-java-programming.html) will generate a different SerialVersionUID for new version of class files and when you try to load old object object serialized by old version of your program you will get InvalidClassException.

### Question 1) What is the difference between Serializable and Externalizable interface in Java?

This is most frequently asked question in Java serialization interview. Here is my version Externalizable provides us writeExternal() and readExternal() method which gives us flexibility to control java serialization mechanism instead of relying on Java's default serialization. Correct implementation of Externalizable interface can [improve performance of application](http://javarevisited.blogspot.sg/2012/01/improve-performance-java-database.html) drastically.

### Question 2) How many methods Serializable has? If no method then what is the purpose of Serializable interface?

Serializable interface exists in java.io package and forms core of java serialization mechanism. It doesn't have any method and also called [Marker Interface in Java](http://javarevisited.blogspot.sg/2012/01/what-is-marker-interfaces-in-java-and.html). When your class implements java.io.Serializable interface it becomes Serializable in Java and gives compiler an indication that use Java Serialization mechanism to serialize this object.

### Question 3) What is serialVersionUID? What would happen if you don't define this?

One of my favorite question interview question on Java serialization. SerialVersionUID is an ID which is stamped on object when it get serialized usually hashcode of object, you can use tool serialver to see serialVersionUID of a serialized object . SerialVersionUID is used for version control of object. you can specify serialVersionUID in your [class file](http://javarevisited.blogspot.sg/2012/05/10-points-about-class-file-in-java.html) also. Consequence of not specifying serialVersionUID is that when you add or modify any field in class then already serialized class will not be able to recover because serialVersionUID generated for new class and for old serialized object will be different. Java serialization process relies on correct serialVersionUID for recovering state of serialized object and throws java.io.InvalidClassException in case of serialVersionUID mismatch, to learn more about serialversionuid see this [article](http://javarevisited.blogspot.sg/2014/05/why-use-serialversionuid-inside-serializable-class-in-java.html).

### Question 4) While serializing you want some of the members not to serialize? How do you achieve it?

Another frequently asked Serialization interview question. This is sometime also asked as what is the use of [transient variable](http://javarevisited.blogspot.sg/2011/09/transient-keyword-variable-in-java.html), does transient and [static variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) gets serialized or not etc. so if you don't want any field to be part of object's state then declare it either static or transient based on your need and it will not be included during Java serialization process.

### Question 5) What will happen if one of the members in the class doesn't implement Serializable interface?

One of the easy question about Serialization process in Java. If you try to serialize an object of a class which implements Serializable, but the object includes a reference to an non- Serializable class then a ‘NotSerializableException’ will be thrown at runtime and this is why I always put a *SerializableAlert* (comment section in my code) , one of the [code comment best practices](http://javarevisited.blogspot.sg/2011/08/code-comments-java-best-practices.html), to instruct developer to remember this fact while adding a new field in a Serializable class.

### Question 6) If a class is Serializable but its super class in not, what will be the state of the instance variables inherited from super class after deserialization?

Java serialization process only continues in object hierarchy till the class is Serializable i.e. implements Serializable [interface in Java](http://javarevisited.blogspot.sg/2012/04/10-points-on-interface-in-java-with.html) and values of the instance variables inherited from super class will be initialized by calling constructor of Non-Serializable Super class during deserialization process. Once the [constructor chaining](http://javarevisited.blogspot.sg/2012/01/what-is-constructor-overloading-in-java.html) will started it wouldn't be possible to stop that , hence even if classes higher in hierarchy implements Serializable interface , there constructor will be executed. As you see from the statement this Serialization interview question looks very tricky and tough but if you are familiar with key concepts its not that difficult.

### Question 7) Can you Customize Serialization process or can you override default Serialization process in Java?

The answer is yes you can. We all know that for serializing an object ObjectOutputStream.writeObject (saveThisobject) is invoked and for reading object ObjectInputStream.readObject() is invoked but there is one more thing which Java Virtual Machine provides you is to define these two method in your class. If you define these two methods in your class then JVM will invoke these two methods instead of applying default serialization mechanism. You can customize behavior of object serialization and deserialization here by doing any kind of pre or post processing task. Important point to note is making these methods private to avoid being inherited, [overridden or overloaded](http://javarevisited.blogspot.sg/2011/12/method-overloading-vs-method-overriding.html). Since only Java Virtual Machine can call private method integrity of your class will remain and Java Serialization will work as normal. In my opinion this is one of the best question one can ask in any Java Serialization interview, a good follow-up question is why should you provide custom serialized form for your object?

### Question 8) Suppose super class of a new class implement Serializable interface, how can you avoid new class to being serialized?

One of the tricky interview question in Serialization in Java. If Super Class of a Class already implements Serializable interface in Java then its already Serializable in Java, since you can not unimplemented an interface its not really possible to make it Non Serializable class but yes there is a way to avoid serialization of new class. To avoid Java serialization you need to implement writeObject() and readObject() method in your Class and need to throw NotSerializableException from those method. This is another benefit of customizing java serialization process as described in above [Serialization interview question](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) and normally it asked as follow-up question as interview progresses.

### Question 9) Which methods are used during Serialization and DeSerialization process in Java?

This is very common interview question in Serialization basically interviewer is trying to know; Whether you are familiar with usage of readObject(), writeObject(), readExternal() and writeExternal() or not. Java Serialization is done by java.io.ObjectOutputStream class. That class is a filter stream which is wrapped around a lower-level byte stream to handle the serialization mechanism. To store any object via serialization mechanism we call ObjectOutputStream.writeObject(saveThisobject) and to deserialize that object we call ObjectInputStream.readObject() method. Call to writeObject() method trigger serialization process in java. one important thing to note about readObject() method is that it is used to read bytes from the persistence and to create object from those bytes and its return an [Object](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html) which needs to be type cast to correct type.

### Question 10) Suppose you have a class which you serialized it and stored in persistence and later modified that class to add a new field. What will happen if you deserialize the object already serialized?

It depends on whether class has its own serialVersionUID or not. As we know from above question that if we don't provide serialVersionUID in our code java compiler will generate it and normally it’s [equal to hashCode of object](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html). by adding any new field there is chance that new serialVersionUID generated for that class version is not the same of already serialized object and in this case Java Serialization API will [throw](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) java.io.InvalidClassException and this is the reason its recommended to have your own serialVersionUID in code and make sure to keep it same always for a single class.

### 11) What are the compatible changes and incompatible changes in Java Serialization Mechanism?

The real challenge lies with change in class structure by adding any field, method or removing any field or method is that with already serialized object. As per Java Serialization specification adding any field or method comes under compatible change and changing class hierarchy or UN-implementing Serializable interfaces some under non compatible changes. For complete list of compatible and non compatible changes I would advise reading Java serialization specification.

### 12) Can we transfer a Serialized object vie network?

Yes you can transfer a Serialized object via network because Java serialized object remains in form of bytes which can be transmitter via network. You can also store serialized object in Disk or database as Blob.

### 13) Which kind of variables is not serialized during Java Serialization?

This question asked sometime differently but the purpose is same whether Java developer knows specifics about [static and transient variable](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) or not. Since static variables belong to the class and not to an object they are not the part of the state of object so they are not saved during Java Serialization process. As Java Serialization only persist state of object and not object itself. Transient variables are also not included in java serialization process and are not the part of the object’s serialized state. After this question sometime interviewer ask a follow-up if you don't store values of these variables then what would be value of these variable once you deserialize and recreate those object? This is for you guys to think about .

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

<http://javarevisited.blogspot.com.by/2012/01/serializable-externalizable-in-java.html>

Difference between Serializable and Externalizable in Java Serialization

**Difference between serializable and externalizable** is [popular java interview question](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html) which we have touched on my earlier post on Serialization: [Top 10 java serialization interview questions](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html). knowing *differences between externalizable and serializable* is not just important from interview point of view but also getting control of serialization process and optimizing performance of serialization. both **serializable** and **extenalizable** used to serialize or persist java objects but the way they do is little different. In case of Serializable Java Virtual machine has full control for serializing object while in case of Externalizable, application gets control for persisting objects. writeExternal() and readExternal() method provides complete **control on format and content of Serialization process** to application which can be leverage to increase performance and speed of serialization process.

## Serialization and Externalization in Java

### Serializable vs Externalization in Java

[difference between Java Serialization vs Externalization in Java](http://javarevisited.blogspot.com/2012/01/tomcat-javalangoutofmemoryerror-permgen.html)here are some more differences between Serializable and Externalizable interface in Java:

1. In case of Serializable, **default serialization process** is used. while in case of Externalizable custom Serialization process is used which is implemented by application.

2. JVM gives call back to readExternel() and writeExternal() of java.io.Externalizalbe interface for restoring and writing objects into persistence.

3. **Externalizable** interface provides complete control of serialization process to application.

4. readExternal() and writeExternal() supersede any specific implementation of writeObject and readObject methods.

Though Externalizable provides complete control, it also presents challenges to serialize super type state and take care of default values in case of [transient variable](http://javarevisited.blogspot.com/2011/09/transient-keyword-variable-in-java.html) and [static variables in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html). If used correctly **Externalizable** interface can improve performance of serialization process.

That’s all on **Difference between Externalizable and Serializable interface in Java**. This is always asked when Java interview take turn towards Serialization after [Multi-Threading questions](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html) and [Collections Interview questions](http://javarevisited.blogspot.com/2011/11/collection-interview-questions-answers.html). Only problem with Serialization is that not many programmer use it and that’s why it look little difficult otherwise once you familiar with Serialization process and rules of Serialization, interview questions can be more easily handled.

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

<http://javarevisited.blogspot.com.by/2011/09/transient-keyword-variable-in-java.html>

How to work with transient variable in java (an example tutorial)

Transient keyword in java is comparatively less common than any other keyword like [volatile](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html).since transient is less common it becomes even more important to understand correct usage of it. In One word **transient keyword** is used in serialization process to prevent any variable from being serialized, so if you have any field which is not making sense to serialize, you can simply declare that as transient and it won't be serialized.In this article we will revise some basics like

What is transient variable in java, why do we need transient variable and most importantly where should we use transient variable or **which fields need to be declared as transient** with example.

### What is transient variable in Java?

[transient variable, transient keyword java](http://javarevisited.blogspot.com/2011/07/java-multi-threading-interview.html)In simple sentence any variable which is modified with transient keyword becomes *transient variable* in java.But before understanding about transient variable let recap something about [serialization in java](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html). Serialization is a process by which object's state is saved by JVM and during deserializaiton it’s recovered by JVM. During Serialization all property of object gets saved except static and transient. So if we would like to exclude any property of an object from being serialized we mark it transient and then JVM doesn't serialize it. While marking any property transient its worth noting to provide it a default value during deserialization otherwise deserialized object is not properly usable.

### Why do we need transient variable in java?

Transient keyword provides you some control over serialization process and gives you flexibility to exclude some of object properties from serialization process. Some time it does make sense not to serialize certain attributes of an object, we will see which variables should not be serialized and should be made transient in next section.

### Which variable you should mark transient?

This is a good question; since we know the purpose of *transient keyword* or having transient variable its make sense to think about which variable should be marked as transient. My rule is that any variable whose value can be calculated from other variables doesn't require to be saved. For example if you have a field called "interest" whose value can be derived from other fields e.g. principle, rate, time etc then there is no need to serialize it.

Another example is of word count, if you are saving article then no need to save word count, because it can be created when article gets deserialized. Another good example of transient keyword is "Logger" since most of the time you have logger instance for [logging in Java](http://javarevisited.blogspot.com/2011/05/top-10-tips-on-logging-in-java.html) but you certainly don't want it to serialize correct?

### Example of transient variable in java

To understand the concept of transient variables let see a live example in java.

public class **Stock** {

    private **transient** Logger logger = Logger.getLogger(Stock.class); **//will not serialized**

    private String symbol; //will be serialized

    private BigInteger price; //serialized

    private long quantity; //serialized

}

### Important points about transient keyword in java

Here are few important points about transient variables in java which I found, let me know if you have some more which I missed out here and I will include here.

1) Transient keyword can only be applied to fields or member variable. Applying it to method or local variable is compilation error.

2) Another important point is that you can declare an variable static and transient at same time and java compiler doesn't complain but doing that doesn't make any sense because transient is to instruct "do not save this field" and static variables are not saved anyway during serialization.

3) In similar way you can apply transient and final keyword together to a variable compiler will not complain but you will face another problem of reinitializing a final variable during deserialization.

4) Transient variable in java is not persisted or saved when an object gets serialized.

That's all from me on transient keyword, let me know how do you use it and if you know any pecularity about transient keyword or something which we need to be aware while using it and missed out here. You can also refer’s Sun Glossary for [meaning of different keywords in Java](http://java.sun.com/docs/glossary.html).

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

<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 tim**e** 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).

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