<http://java67.blogspot.sg/2012/08/what-is-singleton-pattern-in-java.html>

What is Singleton pattern in Java – Example

**ingleton Pattern in Java**  
Singleton pattern in Java is used to create a Singleton class which is accessed by single instance though out Java program life cycle.Singleton is one of the old but very useful design pattern and used in several core Java libraries including Java Development Kit (JDK); java.lang.Runtime is an example of [Singleton pattern in Java](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html). This class represent single instance of Java environment running as JVM and provides several utility method to query important details about runtime like number of available processor, [Java heap memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) etc. Singletons are also very popular in Java interview and one of the frequently asked question. This article contains great collection of [interview question on Singleton pattern](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) in Java and can be useful to enhance your knowledge on Singleton pattern.

### How to write Singleton class in Java

[What is Singleton pattern in Java Example](http://java67.blogspot.sg/2012/07/java-program-to-find-armstrong-numbers.html)Singleton pattern is often implemented either using [lazy loading](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) or by using [double-checked locking](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html), which was not safe and broker in Java 1.4, In short before Java 5 writing Singleton was very difficult and one of the tricky interview question asked on Java interviews, but with introduction of Java memory model in Java 5 and change in volatile variable in Java its possible to write thread-safe singleton using double checked locking. Java 5 also introduced Enum in Java, which is the best choice to implement [thread-safe Singleton in Java](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html). You can now implement Singleton pattern in Java by using just four lines of code as shown below :

public enum Java5Singleton{

 SINGLETON;

}  
  
  
**What is benefit of using Enum to implement Singleton pattern?**  
There are number of advantage to use Enum to implement Singleton pattern in Java :  
  
1) Singleton instance is thread-safe because JVM guarantees that Enum instances are created in thread-safe manner.  
2) JVM also guarantee to maintains Singleton status when Singleton class implements Serializable which is still possible without Enum by using readResolve() method but tedious and complicated.  
  
  
In Summary Singleton is very useful design pattern and Java programmer must be familiar with What is Singleton in Java, How to create Singleton classes, issues faced to maintain Singleton status etc. By using Enum as Singleton we get some benefits in terms of these issues.

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

<http://javarevisited.blogspot.com/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?

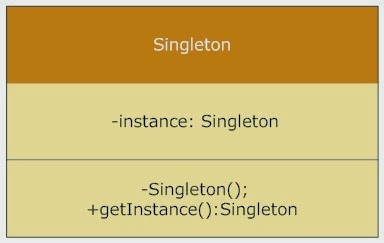
Singleton is a class which has only one instance in whole application and provides a getInstance() method to access the singleton instance. There are many classes in JDK which is implemented using Singleton pattern like java.lang.Runtime which provides getRuntime() method to get access of it and used 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?

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

### 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/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

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.gr/2012/07/why-enum-singleton-are-better-in-java.html>

<http://javarevisited.blogspot.sg/2012/07/why-enum-singleton-are-better-in-java.html>

Why Enum Singleton are better in Java

**Enum Singletons** are new way to implement Singleton pattern in Java by using Enum with just one instance. Though Singleton pattern in Java exists from long time Enum Singletons are relatively new concept and in practice from Java 5 onwards after introduction of Enum as keyword and feature. This article is somewhat related to my earlier post on Singleton, [10 interview questions on Singleton pattern in Java](http://javarevisited.blogspot.sg/2011/03/10-interview-questions-on-singleton.html) where we have discussed common questions asked on interviews about Singleton pattern and [10 Java enum examples](http://javarevisited.blogspot.in/2011/08/enum-in-java-example-tutorial.html), where we have seen how versatile enum can be. This post is about **why should we use Enum as Singleton in Java**, What benefit it offers compared to conventional singleton methods etc.

## Java Enum and Singleton Pattern

Following are some reasons which make sense to me for using Enum to implement Singleton pattern in Java. By the way If you like articles on design pattern than you can also check my post on [Builder design pattern](http://javarevisited.blogspot.com/2012/06/builder-design-pattern-in-java-example.html) and [Decorator design pattern](http://javarevisited.blogspot.sg/2011/11/decorator-design-pattern-java-example.html) .

**1) Enum Singletons are easy to write**

This is by far biggest advantage, if you have been writing Singletons prior ot Java 5 than you know that even with double checked locking you can have more than one instances. though that issue is fixed with Java memory model improvement and gurantee provided by volatile variables from Java 5 onwards but it still tricky to write for many beginners. compared to double checked locking with synchronization Enum singletons are cake walk. If you don't believe than just compare below code for conventional singleton with double checked locking and Enum Singletons:

**Singleton using Enum in Java**

This is the way we generally declare Enum Singleton , it may contain instace variable and instance method but for sake of simplicity I haven’t used any, just beware that if you are using any instance method than you need to ensure thread-safety of that method if at all it affect the state of object. By default creation of Enum instance is thread safe but any other method on Enum is programmers responsibility.

***/\*\****

***\* Singleton pattern example using Java Enumj***

***\*/***

**public** **enum** EasySingleton{

    INSTANCE;

}

You can acess it by EasySingleton.INSTANCE, much easier than calling getInstance() method on Singleton.

**Singleton example with double checked locking**

Below code is an example of double checked locking in Singleton pattern, here getInstance() method checks two times to see whether INSTANCE is null or not and that’s why it’s called double checked locking pattern, remember that double checked locking is broker before Java 5 but with the guranteed of [volatile variable in Java 5](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html) memory model, it should work perfectly.

***/\*\****

***\* Singleton pattern example with Double checked Locking***

***\*/***

**public** **class** DoubleCheckedLockingSingleton{

**private** **volatile** DoubleCheckedLockingSingleton INSTANCE;

**private** DoubleCheckedLockingSingleton(){}

**public** DoubleCheckedLockingSingleton getInstance(){

**if**(INSTANCE == **null**){

**synchronized**(DoubleCheckedLockingSingleton.**class**){

*//double checking Singleton instance*

**if**(INSTANCE == **null**){

                    INSTANCE = **new** DoubleCheckedLockingSingleton();

                }

            }

         }

**return** INSTANCE;

     }

}

You can call DoubleCheckedLockingSingleton.getInstance() to get access of this Singleton class.

Now Just look at amount of code needed to create a **lazy loaded thread-safe Singleton**. With Enum Singleton pattern you can have that in one line because creation of Enum instance is [thread-safe](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) and guranteed by JVM.

People may argue that there are better way to write Singleton instead of Double checked locking approach but every approach has there own advantages and disadvantages like I mostly prefer static field Singleton intialized during classloading as shwon in below example, but keep in mind that is not a **lazy loaded Singleton**:

**Singleton pattern with static factory method**

This is one of my favorite method to impelemnt Singleton pattern in Java, Since Singleton instance is [static](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) and [final variable](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html) it initialized when class is first loaded into memeory so creation of instance is inherently thread-safe.

***/\*\****

***\* Singleton pattern example with static factory method***

***\*/***

**public** **class** Singleton{

*//initailzed during class loading*

**private** **static** **final** Singleton INSTANCE = **new** Singleton();

*//to prevent creating another instance of Singleton*

**private** Singleton(){}

**public** **static** Singleton getSingleton(){

**return** INSTANCE;

    }

}

You can call Singleton.getSingleton() to get access of this class.

**2) Enum Singletons handled Serialization by themselves**

Another problem with conventional Singletons are that once you implement [serializable interface](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html) they are no longer remain Singleton because readObject() method always return a new instance just like constructor in Java. you can avoid that by using readResolve() method and discarding newly created instance by replacing with Singeton as shwon in below example :

*//readResolve to prevent another instance of Singleton*

**private** Object readResolve(){

**return** INSTANCE;

    }

This can become even more complex if your Singleton Class maintain state, as you need to make them [transient](http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html), but witn **Enum Singleton**, Serialization is guarnateed by JVM.

**3) Creation of Enum instance is thread-safe**

As stated in point 1 since creatino of Enum instance is thread-safe by default you don't need to worry about double checked locking.

In summary, given the **Serialzation and thraead-safety guaranteed** and with couple of line of code enum Singleton pattern is best way to create Singleton in Java 5 world. you can still use other popular methods if you feel so but I still have to find a *convincing reason not to use Enum as Singleto*n, let me know if you got any.

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

<http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html>

Double Checked Locking on Singleton Class in Java

Singleton class is quite common among Java developers, but it poses many challenges to junior developers. One of the key challenge they face is how to keep Singleton class as Singleton? i.e. how to prevent multiple instances of a Singleton due to whatever reasons. *Double checked locking of Singleton* is a way to ensure only one instance of Singleton class is created through application life cycle. As name suggests, in double checked locking, code checks for an existing instance of [Singleton class](http://javarevisited.blogspot.sg/2013/03/difference-between-singleton-pattern-vs-static-class-java.html) twice with and without locking to double ensure that no more than one instance of singleton gets created. By the way, it was broken before Java fixed its memory models issues in JDK 1.5. In this article, we will see *how to write code for double checked locking of Singleton in Java*, why double checked locking was broken before Java 5 and How that was fixed. By the way this is also important from interview point of view, I have heard it’s been asked to code double checked locking of Singleton by hand on companies in both financial and service sector, and believe me it’s tricky, until you have clear understanding of what you are doing. You can also see my full list of [Singleton design pattern questions](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) to prepare well.

### Why you need Double checked Locking of Singleton Class?

One of the common scenario, where a Singleton class breaks its contracts is multi-threading. If you ask a beginner to write code for [Singleton design pattern](http://java67.blogspot.sg/2012/08/what-is-singleton-pattern-in-java.html), there is good chance that he will come up with something like below :

*private* *static* *Singleton* \_instance;

*public* *static* *Singleton* getInstance() {

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

\_instance = new *Singleton*();

}

return \_instance;

}

and when you point out that this code will create multiple instances of Singleton class if called by more than one thread parallel, he would probably make this whole getInstance() method [synchronized](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html), as shown in our 2nd code example getInstanceTS() method. Though it’s a thread-safe and solves issue of multiple instance, it's not very efficient. You need to bear cost of synchronization all the time you call this method, while synchronization is only needed on first class, when Singleton instance is created. This will bring us to **double checked locking pattern**, where only critical section of code is locked. Programmer call it double checked locking because there are two checks for \_instance == null, one without locking and other with locking (inside synchronized) block. Here is how double checked locking looks like in Java :

*public* *static* *Singleton* getInstanceDC() {

if (\_instance == **null**) { *// Single Checked*

*synchronized* (*Singleton*.class) {

if (\_instance == **null**) { *// Double checked*

\_instance = new *Singleton*();

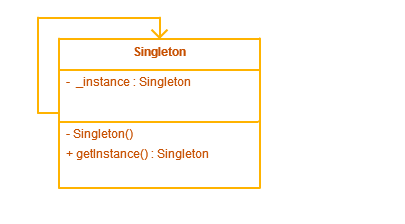
}

}

}

return \_instance;

}



On surface this method looks perfect, as you only need to pay price for synchronized block one time, but it still broken, until you make \_instance variable [volatile](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html). Without volatile modifier it's possible for another thread in Java to see half initialized state of \_instance variable, but with volatile variable guaranteeing happens-before relationship, all the write will happen on volatile \_instance before any read of \_instance variable. This was not the case prior to Java 5, and that's why double checked locking was broken before. Now, with *happens-before guarantee*, you can safely assume that this will work. By the way this is not the best way to create thread-safe Singleton, you can [use Enum as Singleton](http://javarevisited.blogspot.com/2012/07/why-enum-singleton-are-better-in-java.html), which provides inbuilt thread-safety during instance creation. Another way is to use static holder pattern.

/\*

\* A journey to write double checked locking of Singleton class in Java.

\*/

class Singleton {

private volatile static Singleton \_instance;

private Singleton() {

// preventing Singleton object instantiation from outside

}

/\*

\* 1st version: creates multiple instance if two thread access

\* this method simultaneously

\*/

public static Singleton getInstance() {

if (\_instance == null) {

\_instance = new Singleton();

}

return \_instance;

}

/\*

\* 2nd version : this definitely thread-safe and only

\* creates one instance of Singleton on concurrent environment

\* but unnecessarily expensive due to cost of synchronization

\* at every call.

\*/

public static synchronized Singleton getInstanceTS() {

if (\_instance == null) {

\_instance = new Singleton();

}

return \_instance;

}

/\*

\* 3rd version : An implementation of double checked locking of Singleton.

\* Intention is to minimize cost of synchronization and improve performance,

\* by only locking critical section of code, the code which creates

instance of Singleton class.

\* By the way this is still broken, if we don't make \_instance volatile,

as another thread can

\* see a half initialized instance of Singleton.

\*/

public static Singleton getInstanceDC() {

if (\_instance == null) {

synchronized (Singleton.class) {

if (\_instance == null) {

\_instance = new Singleton();

}

}

}

return \_instance;

}

}

That's all about *double checked locking of Singleton class in Java*. This is one of the controversial way to create [thread-safe Singleton in Java](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html), with simpler alternatives available in terms of using Enum as Singleton class. I don't suggest you to implement your Singleton like that as there are many better way to implement Singleton pattern in Java. Though, this question has historical significance and also teaches how concurrency can introduce subtle bugs. As I said before, this is very important from interview point of view. Practice writing double checked locking of Singleton class by hand before going for any Java interview. This will develop your insight on coding mistakes made by Java programmers. On related note, In modern day of Test driven development, Singleton is regarded as anti pattern because of difficulty it present to mock its behaviour, so if you are TDD practitioner better avoid using Singleton pattern.

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

<http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html>

How to create thread safe Singleton in Java - Java Singleton Example

Thread safe Singleton means a Singleton class which returns exactly same instance even if exposed to multiple threads. Singleton in Java has been a classical design pattern like [Factory method pattern](http://javarevisited.blogspot.sg/2011/12/factory-design-pattern-java-example.html) or [Decorator design pattern](http://javarevisited.blogspot.sg/2011/11/decorator-design-pattern-java-example.html) and has been used a lot even inside JDK like java.lang.Runtime is an example of Singleton class. Singleton pattern ensures that exactly one instance of class will remain in Java program at any time. In our last post [10 Interview questions on Singleton in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) we have discussed many different questions asked on Singleton pattern, One of them was writing Thread safe singleton in Java. Prior to Java 5 [double checked locking](http://javarevisited.blogspot.sg/2012/07/why-enum-singleton-are-better-in-java.html) mechanism is used to create *thread-safe singleton* in Java which breaks if one Thread doesn't see instance created by other thread at same time and eventually you will end up with more than one instance of Singleton class. From Java 5 onwards [volatile variable](http://javarevisited.blogspot.sg/2011/06/volatile-keyword-java-example-tutorial.html) guarantee can be used to write thread safe singleton by using double checked locking pattern. I personally don't prefer that way as there are many other simpler alternatives to write thread-safe singleton is available available like using [static field](http://javarevisited.blogspot.sg/2011/11/static-keyword-method-variable-java.html) to initialize Singleton instance or using [Enum as Singleton in Java](http://javarevisited.blogspot.sg/2012/07/why-enum-singleton-are-better-in-java.html). Let’s see example of both ways to create thread-safe Singleton in Java.

## Java Singleton Example – Thread safe Singleton in Java using Enum

This is one of the example of Enum which I missed while writing [10 Examples of Enum in Java](http://javarevisited.blogspot.sg/2011/08/enum-in-java-example-tutorial.html). Using Enum to create Singleton is by far most simple and effective way to create thread-safe Singleton in Java, as thread-safety guarantee is provided by Java programming language itself. You don't need to bother about thread-safety issue. Since Enum instances are by default [final in Java](http://javarevisited.blogspot.sg/2011/12/final-variable-method-class-java.html), it also provides safety against multiple instance due to [serialization](http://javarevisited.blogspot.sg/2011/04/top-10-java-serialization-interview.html). One point worth remembering is that, when we talk about thread-safe Singleton, we are talking about thread-safety during instance creation of Singleton class and not when we call any method of Singleton class. If your Singleton class maintain any state and contains method to modify that state, you need to write code to avoid and [thread-safety](http://javarevisited.blogspot.sg/2012/01/how-to-write-thread-safe-code-in-java.html) and [synchronization issues](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html). Any way here is code *example of creating thread safe Singleton in Java using Enum*.

**public** **enum** Singleton{  
    INSTANCE;  
    
    **public** **void** show(){  
        **System**.out.println("Singleton using Enum in Java");  
    }  
}

//You can access this Singleton as Singleton.INSTANCE and call any method like below

Singleton.INSTANCE.show();

If this suits your need than this is the most easy way of writing thread-safe Singleton in Java. Using Enum as Singleton also provide couple of more benefits which you can find out on [Why Enum Singleton are better in Java](http://javarevisited.blogspot.sg/2012/07/why-enum-singleton-are-better-in-java.html).

## Java Singleton Example - Thread Safe Singleton using Static field Initialization

You can also create thread safe Singleton in Java by creating Singleton instance during [class loading](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html). static fields are initialized during class loading and [Classloader](http://javarevisited.blogspot.com.au/2012/12/how-classloader-works-in-java.html) will guarantee that instance will not be visible until its fully created. Here is example of creating thread safe singleton in Java using static factory method. Only disadvantage of this implementing Singleton patter using static field is that this is *not a lazy initialization* and Singleton is initialized even before any clients call there getInstance() method.

**public** **class** Singleton{  
    **private** **static** **final** Singleton INSTANCE = **new** Singleton();  
    
    **private** Singleton(){ }  
  
    **public** **static** Singleton getInstance(){  
        **return** INSTANCE;  
    }  
    **public** **void** show(){  
        **System**.out.println("Singleon using static initialization in Java");  
    }  
}

//Here is how to access this Singleton class

Singleton.getInstance().show();

here we are not creating Singleton instance inside getInstance() method instead it will be created by ClassLoader. Also [private constructor](http://javarevisited.blogspot.sg/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) makes impossible to create another instance , except one case. You can still access private constructor by reflection and calling setAccessible(true). By the way You can still prevent creating another instance of Singleton by this way by [throwing Exception](http://javarevisited.blogspot.sg/2012/02/difference-between-throw-and-throws-in.html) from constructor.

That's all on **how to create thread safe Singleton in Java**. Both the approach are safe with thread-safety issue but my personal favorite is using Enum because of its simplicity, prevention of multiple instance against Serialization attack and concise code.

Other **Java design pattern tutorials** from Javarevisited Blog

[What is Observer pattern in Java with real world Example](http://javarevisited.blogspot.sg/2011/12/observer-design-pattern-java-example.html)

[20 design pattern and software design interview question](http://javarevisited.blogspot.sg/2012/06/20-design-pattern-and-software-design.html)

[10 OOPS and SOLID design principles for Java programmer](http://javarevisited.blogspot.de/2012/03/10-object-oriented-design-principles.html)

[What is Builder design pattern in Java with example](http://javarevisited.blogspot.ro/2012/06/builder-design-pattern-in-java-example.html)

[Why wait and notify are declared in object class in Java](http://javarevisited.blogspot.sg/2012/02/why-wait-notify-and-notifyall-is.html)

Posted by Javin Paul at [7:18 PM](http://javarevisited.blogspot.com.by/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) [http://img1.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=8712770457197348465&postID=5059866504982958729)

[Email This](https://www.blogger.com/share-post.g?blogID=8712770457197348465&postID=5059866504982958729&target=email) [BlogThis!](https://www.blogger.com/share-post.g?blogID=8712770457197348465&postID=5059866504982958729&target=blog) [Share to Twitter](https://www.blogger.com/share-post.g?blogID=8712770457197348465&postID=5059866504982958729&target=twitter) [Share to Facebook](https://www.blogger.com/share-post.g?blogID=8712770457197348465&postID=5059866504982958729&target=facebook)

Labels: [core java](http://javarevisited.blogspot.com.by/search/label/core%20java?max-results=3) , [core java interview question](http://javarevisited.blogspot.com.by/search/label/core%20java%20interview%20question?max-results=3) , [design patterns](http://javarevisited.blogspot.com.by/search/label/design%20patterns?max-results=3)

Location: [United States](https://maps.google.com/maps?q=United+States@39.095962936305476,-103.0078125&z=10)

#### 10 comments :

Anonymous said...

public class Singleton also needs to be declared final, to avoid being subclassed (and, for example, instantiate a new Singleton object via reflection)

[December 28, 2012 at 5:15 AM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1356700530538#c3683543220460449383)

[Javin @Linkded List vs ArrayList](http://javarevisited.blogspot.de/2012/02/difference-between-linkedlist-vs.html) said...

@Anonymous, does it really necessary to make Singleton final, doesn't making constructor private will prevent it from being extended? Regarding Reflection, I think only way to prevent creating instance is throwing Exception from constructor itself.

[December 28, 2012 at 7:30 PM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1356751838048#c7685934695028302834)

[Dmitry Semenov](https://www.blogger.com/profile/05688156677604765521) said...

How about static inner class with singleton instance?

[January 2, 2013 at 1:08 AM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1357117735174#c9210703634303608309)

[Alexander](https://www.blogger.com/profile/12389587104046970446) said...

Hi, and Happy New Year  
  
Showed singleton is only one of more thread-safe singleton realizations.  
  
Each of which have own pros and cons.  
  
Actually this have one problem only if you have...  
Imagin you have huge entrprise platfrom that have several class loaders. Each class loader work simultaneously. In this case, class loaders can create several singleton instances from only one class.

[January 2, 2013 at 2:19 AM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1357121940750#c741133347791830995)

Anonymous said...

Writing a thread safe singleton for multithreaded environment is not a joke. Anything with threading requires serious knowledge of Java multithreading concept. I like your blog for spreading words and experience on Java Singleton multithreading issues.

[February 5, 2013 at 10:10 PM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1360131015440#c612053200561608855)

[SARAL SAXENA](https://www.blogger.com/profile/01084233786047386880) said...

If you guys are looking for thread safety and the combination of lazy initialization..then follow Initialization-on-demand holder idiom..  
  
public class Singleton {  
private Singleton() {  
}  
  
private static class LazyHolder {  
private static final Singleton INSTANCE = new Singleton();  
}  
  
public static Singleton getInstance() {  
return LazyHolder.INSTANCE;  
}  
}   
  
The implementation relies on the well-specified initialization phase of execution within the Java Virtual Machine (JVM); see section 12.4 of Java Language Specification (JLS) for details.  
When the class Something is loaded by the JVM, the class goes through initialization. Since the class does not have any static variables to initialize, the initialization completes trivially. The static class definition LazyHolder within it is not initialized until the JVM determines that LazyHolder must be executed. The static class LazyHolder is only executed when the static method getInstance is invoked on the class Something, and the first time this happens the JVM will load and initialize the LazyHolder class. The initialization of the LazyHolder class results in static variable INSTANCE being initialized by executing the (private) constructor for the outer class Something. Since the class initialization phase is guaranteed by the JLS to be serial, i.e., non-concurrent, no further synchronization is required in the static getInstance method during loading and initialization. And since the initialization phase writes the static variable INSTANCE in a serial operation, all subsequent concurrent invocations of the getInstance will return the same correctly initialized INSTANCE without incurring any additional synchronization overhead.

[April 12, 2013 at 7:59 PM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1365821953743#c87501025907064023)

Anonymous said...

Why Singleton is not thread-safe? Sorry but I didn't get the problem with Singleton and multiple threads. Can you please explain me in Simple words.

[October 1, 2014 at 4:01 AM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1412161280793#c3381606004425828494)

[Bimales](https://www.blogger.com/profile/04307734505669461465) said...

as per given example... corrected class should be like below   
  
package org.tutorial.designpattern.singleton;  
  
public final class SingletonClass {  
  
private static SingletonClass singletonClass = new SingletonClass();  
  
private SingletonClass(){}  
  
public static SingletonClass getInstance(){  
return singletonClass;  
}  
public int showHashCode(){  
return singletonClass.hashCode();  
}  
}

[December 14, 2014 at 11:55 AM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1418586926049#c8315222937590172129)

[VARDHAN](https://www.blogger.com/profile/03238131467745570521) said...

For the following code  
public enum SingletonConnection  
{  
INSTANCE;  
  
private Connection conn;  
private SingletonConnection ()  
{  
//... some code heavy opetation  
//. some code heavy operation  
// come Code heavy operation  
//conn = ConstructConnection(); heavy operation  
}  
  
public Connection getConnection()  
{  
return connection;  
}  
}  
  
  
Basically I want to have heavy Enum Constant  
  
Now under the following scenario how jvm guarantees that no two connection objects are initialized;  
If two threads access SingletonConnection.INSTANCE, where we get a case where two threads executing constructor code at same point of time and will two Connection Objects are initialize?. If so how to avoid.  
  
Basically I want to have heavy Enum Constant which is a heavy operation. How can I can I acieve

[August 12, 2015 at 8:54 PM](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html?showComment=1439438078926#c8034609384360061749)

[Javin Paul](https://www.blogger.com/profile/15028902221295732276) said...

Enum constants are created by JVM and this is a single threaded operation guaranteed by JVM, so you don't need to worry about it. That's why Enum Singletons are better than normal one because you don't have to do anything special to guarantee that thread-safe creation of Singleton.

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

<http://javarevisited.blogspot.com.by/2011/06/volatile-keyword-java-example-tutorial.html>

How Volatile in Java works? Example of volatile keyword in Java

**How to use Volatile keyword in Java**  
What is volatile variable in Java and when to use Volatile variable in Java is famous multi-threading interview question in Java interviews. Though many programmer knows what is a volatile variable but they fail on second part i.e. where to use volatile variable in Java as its not common to have clear understanding and hands-on on volatile in Java. In this tutorial we will address this gap by providing simple example of volatile variable in Java and discussing some when to use Volatile variable in Java. Any way Volatile keyword in Java is used as an indicator to Java compiler and Thread that do not cache value of this variable and always read it from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html). So if you want to share any variable in which read and write operation is atomic by implementation e.g. read and write in int or boolean variable you can declare them as volatile variable. From Java 5 along with major changes like Autoboxing, Enum, Generics and Variable arguments , Java introduces some change in Java Memory Model (JMM), Which guarantees visibility of changes made by one thread to another also as "happens-before" which solves the problem of memory writes that happen in one thread can "leak through" and be seen by another thread. Java volatile keyword cannot be used with method or class and it can only be used with variable. Java volatile keyword also guarantees visibility and ordering , after Java 5 write to any volatile variable happens before any read into volatile variable. By the way use of volatile keyword also prevents compiler or JVM from reordering of code or moving away them from [synchronization barrier](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html).

## Volatile variable Example in Java

To Understand example of volatile keyword in java let’s go back to [Singleton pattern in Java](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html) and see [double checked locking in Singleton](http://javarevisited.blogspot.sg/2014/05/double-checked-locking-on-singleton-in-java.html) with Volatile and without volatile keyword in java.

/\*\*

\* Java program to demonstrate where to use Volatile keyword in Java.

\* In this example Singleton Instance is declared as volatile variable to ensure

\* every thread see updated value for \_instance.

\*

\* @author Javin Paul

\*/

**public** **class** **Singleton**{

**private** **static** **volatile** Singleton \_instance; //volatile variable

**public** **static** Singleton **getInstance**(){

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

**synchronized**(Singleton.class){

**if**(\_instance == **null**)

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

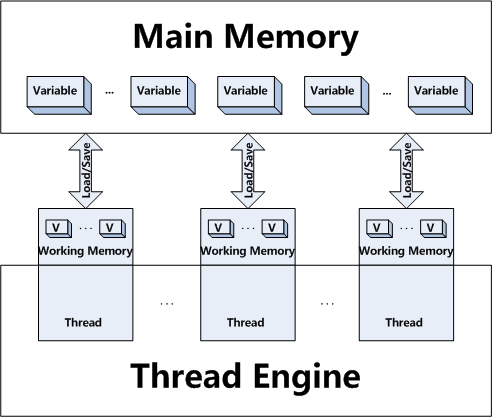
}

}

**return** \_instance;

}

If you look at the code carefully you will be able to figure out:  
1) We are only creating instance one time  
2) We are creating instance lazily at the time of first request comes.  
  
  
If we do not make \_instance variable volatile then Thread which is creating instance of Singleton is not able to communicate other thread, that instance has been created until it comes out of the Singleton block, so if Thread A is creating Singleton instance and just after creation lost the CPU, all other thread will not be able to see value of \_instance as not null and they will believe its still [null](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html).



Why? because reader threads are not doing any locking and until writer thread comes out of synchronized block, memory will not be synchronized and value of \_instance will not be updated in main memory. With Volatile keyword in Java this is handled by Java himself and such updates will be visible by all reader threads.  
  
  
So in Summary apart from [synchronized keyword in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), volatile keyword is also used to communicate content of memory between threads.

### Let’s see another example of volatile keyword in Java

most of the time while writing game we use a variable bExit to check whether user has pressed exit button or not, value of this variable is updated in [event thread](http://javarevisited.blogspot.sg/2011/09/invokeandwait-invokelater-swing-example.html) and checked in game thread , So if we don't use volatile keyword with this variable , Game Thread might miss update from event handler thread if its not synchronized in java already. volatile keyword in java guarantees that value of volatile variable will always be read from main memory and "*happens-before"* relationship in Java Memory model will ensure that content of memory will be communicated to different threads.

**private** **boolean** bExit**;**

**while(!**bExit**)** **{**

checkUserPosition**();**

updateUserPosition**();**

**}**

In this code example One Thread (Game Thread) can cache the value of "bExit" instead of getting it from [main memory](http://javarevisited.blogspot.sg/2011/05/java-heap-space-memory-size-jvm.html) every time and if in between any other thread (Event handler Thread) changes the value; it would not be visible to this thread. Making boolean variable "bExit" as volatile in java ensures this will not happen.

## When to use Volatile variable in Java

One of the most important thing in learning of volatile keyword is understanding when to use volatile variable in Java. Many [programmer](http://javarevisited.blogspot.sg/2011/06/top-programming-interview-questions.html) knows what is volatile variable and How does it work but they never really used volatile for any practical purpose. Here are couple of example to demonstrate when to use Volatile keyword in Java:  
  
  
1) You can use Volatile variable if you want to read and write long and [double](http://javarevisited.blogspot.sg/2011/10/convert-double-to-string-example.html) variable atomically. long and double both are [64 bit](http://javarevisited.blogspot.sg/2012/01/find-jvm-is-32-or-64-bit-java-program.html) data type and by default writing of long and double is not atomic and platform dependence. Many platform perform write in long and double variable 2 step, writing 32 bit in each step, due to this its possible for a Thread to see 32 bit from two different write. You can avoid this issue by making long and double variable volatile in Java.  
  
  
2) Volatile variable can be used as an alternative way of achieving [synchronization in Java](http://javarevisited.blogspot.sg/2011/04/synchronization-in-java-synchronized.html) in some cases, like Visibility. with volatile variable its guaranteed that all reader thread will see updated value of volatile variable once write operation completed, without volatile keyword different reader thread may see different values.  
  
  
3) volatile variable can be used to inform compiler that a particular field is subject to be accessed by multiple threads, which will prevent compiler from doing any reordering or any kind of optimization which is not desirable in multi-threaded environment. Without volatile variable compiler can re-order code, free to cache value of volatile variable instead of always reading from main memory. like following example without volatile variable may result in [infinite loop](http://javarevisited.blogspot.sg/2011/12/how-to-traverse-or-loop-hashmap-in-java.html)

**private** **boolean** isActive **=** thread**;**

**public** **void** printMessage**(){**

**while(**isActive**){**

System**.**out**.**println**(**"Thread is Active"**);**

**}**

**}**

without *volatile modifier* its not guaranteed that one [Thread](http://javarevisited.blogspot.sg/2012/01/difference-thread-vs-runnable-interface.html) see the updated value of isActive from other thread. compiler is also free to cache value of isActive instead of reading it from main memory in every iteration. By making isActive a volatile variable you avoid these issue.  
  
  
  
4) Another place where volatile variable can be used is to fixing double checked locking in Singleton pattern. As we discussed in [Why should you use Enum as Singleton](http://javarevisited.blogspot.gr/2012/07/why-enum-singleton-are-better-in-java.html) that double checked locking was broken in Java 1.4 environment.

### Important points on Volatile keyword in Java

1. volatile keyword in Java is only application to variable and using volatile keyword with class and method is illegal.  
  
  
2. volatile keyword in Java guarantees that value of volatile variable will always be read from main memory and not from Thread's local cache.  
  
  
3. In Java reads and writes are [atomic](http://javarevisited.blogspot.sg/2012/02/what-is-race-condition-in.html) for all variables declared using Java volatile keyword (including long and double variables).  
  
  
4. Using Volatile keyword in Java on variables reduces the risk of memory consistency errors, because any write to a volatile variable in Java establishes a happens-before relationship with subsequent reads of that same variable.  
  
  
5. From Java 5 changes to a volatile variable are always visible to other threads. What’s more it also means that when a thread reads a volatile variable in Java, it sees not just the latest change to the volatile variable but also the side effects of the code that led up the change.  
  
  
6. Reads and writes are atomic for reference variables are for most primitive variables (all types except long and double) even without use of volatile keyword in Java.  
  
  
7. An access to a volatile variable in Java never has chance to block, since we are only doing a simple read or write, so unlike a synchronized block we will never hold on to any lock or wait for any [lock](http://javarevisited.blogspot.sg/2010/10/what-is-deadlock-in-java-how-to-fix-it.html).  
  
  
8. Java volatile variable that is an object reference may be null.  
  
  
9. Java volatile keyword doesn't means atomic, its common misconception that after declaring volatile ++ will be atomic, to make the operation atomic you still need to ensure exclusive access using synchronized method or block in Java.  
  
  
10. If a variable is not shared between [multiple threads](http://javarevisited.blogspot.sg/2013/02/how-to-join-multiple-threads-in-java-example-tutorial.html) no need to use volatile keyword with that variable.

## Difference between synchronized and volatile keyword in Java

What is difference between volatile and synchronized is another popular core Java question asked in multi-threading and concurrency interviews. Remember volatile is not a replacement of synchronized keyword but can be used as an alternative in certain cases. Here are few differences between volatile and synchronized keyword in Java.  
  
  
1. Volatile keyword in Java is a field modifier, while synchronized modifies code blocks and methods.  
  
  
2. Synchronized obtains and releases lock on monitor’s Java volatile keyword doesn't require that.  
  
  
3. Threads in Java can be blocked for waiting any monitor in case of synchronized, that is not the case with volatile keyword in Java.  
  
  
4. Synchronized method affects performance more than volatile keyword in Java.  
  
  
5. Since volatile keyword in Java only synchronizes the value of one variable between Thread memory and "main" memory while synchronized synchronizes the value of all variable between thread memory and "main" memory and locks and releases a monitor to boot. Due to this reason synchronized keyword in Java is likely to have more overhead than volatile.  
  
  
6. You can not synchronize on null object but your volatile variable in java could be null.  
  
  
7. From Java 5 Writing into a volatile field has the same memory effect as a monitor release, and reading from a volatile field has the same memory effect as a monitor acquire   
  
  
In Summary volatile keyword in Java is not a replacement of synchronized block or method but in some situation is very handy and can save performance overhead which comes with [use of synchronization in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html)  
  
  
if you like to know more about volatile I would also suggest to go thorough FAQ on Java Memory Model here which explains happens-before operations quite well.

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

<http://javarevisited.blogspot.sg/2012/03/difference-between-transient-and.html>

Difference between transient and volatile keyword in Java

Surprisingly "**Difference between transient and volatile keyword in Java**" has asked many times on [various java interview](http://javarevisited.blogspot.com/2011/04/top-20-core-java-interview-questions.html). volatile and transient are two completely different keywords from different areas of Java programming language. transient keyword is used during [serialization of Java object](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html) while volatile is related to visibility of variables modified by multiple thread during concurrent programming. Only similarity between volatile and transient is that they are less used or uncommon keywords and not as popular as public, [static](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) or [final](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html). Anyway its good to know what transient keyword do in Java or how to use volatile keyword in Java. In this article we will couple of points between volatile and transient which can be used to answer this interview question.

[Difference between transient and volatile keyword in Java](http://2.bp.blogspot.com/-wrzDeQGAe1I/TWu8pLuLr4I/AAAAAAAAADE/V017G-6Q61w/s1600/java_logo_50_50.jpg)This article is in continuation of earlier interview question on serialization like [difference between Serializable and Externaliable](http://javarevisited.blogspot.com/2012/01/serializable-externalizable-in-java.html)  and  [Top 10 Java serialization interview question](http://javarevisited.blogspot.com/2011/04/top-10-java-serialization-interview.html). If you haven’t read them already you may find them useful and interesting.

## Difference between volatile and transient keyword in Java

1) transient keyword is used along with [instance variables](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) to exclude them from serialization process. if a field  is transient its value will not be persisted. see my post [what is transient keyword in java](http://javarevisited.blogspot.com/2011/09/transient-keyword-variable-in-java.html) for more details. On the other hand volatile keyword can also be used in variables to indicate compiler and JVM that always read its value from main memory and follow happens-before relationship on visibility of volatile variable among multiple thread. see my post how and [when to use volatile keyword in Java](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html) for more details.

2) transient keyword can not be used along with static keyword but volatile can be used along with static.

3) transient variables are initialized with default value during de-serialization and there assignment or restoration of value has to be handled by application code.

That’s all on **difference between transient and volatile keyword in java**. As I said this interview question doesn’t really test you and just try to find whether you are familiar with those less known keywords in java or not. Let us know if you come across any other difference between volatile and transient keyword in java.

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