**Core Java Famous Interview Questions:**

[**Why non-static variable cannot be referenced from a static context?**](http://javarevisited.blogspot.in/2012/02/why-non-static-variable-cannot-be.html)

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

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

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

why main is declared static in Java see the link.

public class **StaticTest** {

private int count=0;

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

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

}}

Why non static variable can not be called from static method

Now before finding answer of compiler error "non-static variable cannot be referenced from a static context", let's have a quick revision of static. [Static variable in Java](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) belongs to Class and its **value remains same for all instance**. static variable initialized when class is loaded into [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) on the other hand instance variable has different value for each instances and they get created when instance of an object is created either by using new() operator or using reflection like Class.newInstance().

So if you try to access a non static variable without any instance compiler will complain because **those variables are not yet created** and they don't have any existence until an instance is created and they are associated with any instance. So in my opinion only reason which make sense to disallow [non static or instance variable](http://javarevisited.blogspot.com/2012/02/difference-between-instance-class-and.html) inside static context is non existence of instance.In summary since code in static context can be run even without creating any instance of class, it does not make sense asking value for an specific instance which is not yet created.

How to access non static variable inside static method or block

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

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

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

public class **StaticTest** {

private int count=0;

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

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

test.count++;

}

}

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

[**How to avoid deadlock in Java Threads**](http://javarevisited.blogspot.in/2010/10/what-is-deadlock-in-java-how-to-fix-it.html)

Answer is simple, when two or more threads are waiting for each other to release lock and get stuck for infinite time, situation is called deadlock . It will only happen in case of multitasking.

**How do you detect deadlock in Java ?**

Though this could have many answers , my version is first I would look the code if I see nested synchronized block or calling one synchronized method from other or trying to get lock on different object then there is good chance of deadlock if developer is not very careful.  
  
Other way is to find it when you actually get locked while running the application , try to take thread dump , in Linux you can do this by command **"kill -3"** , this will print status of all the thread in application log file and you can see which thread is locked on which object.Other way is to use **jconsole**, it will show you exactly which threads are get locked and on which object.

**Write a Java program which will result in deadlock?**

Once you answer this , they may ask you to **write code which will result in deadlock ?**  
here is one of my version  
  
/\*\*

\* Java program to create a deadlock by imposing circular wait.

\*

\* @author WINDOWS 8

\*

\*/

public class DeadLockDemo {

/\*

\* This method request two locks, first String and then Integer

\*/

public void method1() {

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

}

}

}

/\*

\* This method also requests same two lock but in exactly

\* Opposite order i.e. first Integer and then String.

\* This creates potential deadlock, if one thread holds String lock

\* and other holds Integer lock and they wait for each other, forever.

\*/

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

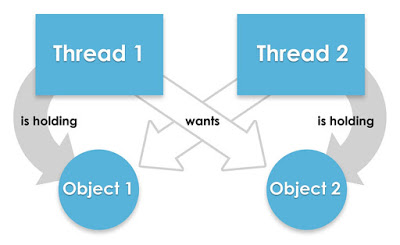
System.out.println("Aquired lock on String.class object");

}

}

}

}

If method1() and method2() both will be called by two or many threads , there is a good chance of deadlock because if thread 1 acquires lock on Sting object while executing method1() and thread 2 acquires lock on Integer object while executing method2() both will be waiting for each other to release lock on Integer and String to proceed further which will never happen.  
  
This diagram exactly demonstrate our program, where one thread holds lock on one object and waiting for other object lock which is held by other thread.  
  


How to avoid deadlock in Java?

Now interviewer comes to final part, one of the most important in my view; *How do you fix deadlock?* or **How to avoid deadlock in Java?**  
  
If you have looked above code carefully then you may have figured out that real reason for deadlock is not multiple threads but ***the way they are requesting lock*** , if you provide an ordered access then problem will be resolved , here is my fixed version, which avoids deadlock by avoiding circular wait with no preemption.  
public class DeadLockFixed {

/\*\*

\* Both method are now requesting lock in same order, first Integer and then String.

\* You could have also done reverse e.g. first String and then Integer,

\* both will solve the problem, as long as both method are requesting lock

\* in consistent order.

\*/

public void method1() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

public void method2() {

synchronized (Integer.class) {

System.out.println("Aquired lock on Integer.class object");

synchronized (String.class) {

System.out.println("Aquired lock on String.class object");

}

}

}

}  
  
Now there would not be any deadlock because both methods are accessing lock on

Integerand String class literal in same order. So, if thread A acquires lock on Integer object , thread B will not proceed until thread A releases Integer lock, same way thread A will not be blocked even if thread B holds String lock because now thread B will not expect thread A to release Integer lock to proceed further.

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.

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 which is 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).

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

thread-safety or **thread-safe code in Java** refers to code which can safely be used or shared in concurrent ormulti-threading environment and they will behave as expected. any code, class or object which can behave differently from its contract on concurrent environment is not thread-safe. thread-safety is one of the riskintroduced by using [threads in Java](http://javarevisited.blogspot.com/2011/02/how-to-implement-thread-in-java.html) and I have seen java programmers and developers struggling to *write* thread-safe code or just understanding *what is thread-safe* codeand what is not? This will not be very detailed article on thread-safety or low

level details of [synchronization in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) instead we will keep it simple and focus on oneexample of non thread-safe code and try to understand what is thread-safety and **how to make an** codethread-safe.

How to make Thread-Safe Code in Java

Example of Non Thread Safe Code in Java

Here is an example of **non thread-safe code**, look at the code and find out *why this code is not thread safe* ?

/\*

**\* Non Thread-Safe Class in Java**

\*/

public class **Counter** {

private int count;

/\*

\* This method is not thread-safe because ++ is not an atomic operation

\*/

public int getCount(){

return count++;

}

}

**Above example is not thread-safe** because ++ (increment operator) is not an **atomic operation** and can be broken down into read, update and write operation. if multiple thread call getCount() approximately same time each of these three operation may coincide or overlap with each other for example while thread 1 is updating value , thread 2 reads and still gets old value, which eventually let thread 2 override thread 1 increment and **one count is lost** because multiple thread called it concurrently.

How to make code Thread-Safe in Java

There are multiple ways to make this code thread safe in Java:

1) Use [synchronized keyword in Java](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html) and lock the getCount() method so that only one thread can execute it at a time which removes possibility of coinciding or interleaving.

2) use **Atomic Integer**, which makes this ++ operation atomic and since **atomic operations are thread-safe** and saves cost of external synchronization.

here is a thread-safe version of Counter class in Java:

/\*

\* **Thread-Safe Example in Java**

\*/

public class Counter {

private int count;

AtomicInteger atomicCount = new AtomicInteger( 0 );

/\*

\* **This method thread-safe now because of locking and synchornization**

\*/

public synchronized int getCount(){

return count++;

}

/\*

\* **This method is thread-safe because count is incremented atomically**

\*/

public int getCountAtomically(){

return atomicCount.incrementAndGet();

}

}

Important points about Thread-Safety in Java

Here is some points worth remembering to **write thread safe code in Java**, these knowledge also helps you to avoid some serious concurrency issues in Java like race condition or [deadlock in Java](http://javarevisited.blogspot.com/2010/10/what-is-deadlock-in-java-how-to-fix-it.html):

1) Immutable objects are by default thread-safe because there state can not be modified once created. Since String is immutable in Java, its inherently thread-safe.

2) Read only or [final variables in Java](http://javarevisited.blogspot.com/2011/12/final-variable-method-class-java.html) are also thread-safe in Java.

3) Locking is one way of achieving thread-safety in Java.

4) [Static variables](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) if not synchronized properly becomes major cause of thread-safety issues.

5) Example of thread-safe class in Java: Vector, Hashtable, ConcurrentHashMap, String etc.

6) Atomic operations in Java are thread-safe e.g. reading a 32 bit int from memory because its an atomic operation it can't interleave with other thread.

7) local variables are also thread-safe because each thread has there own copy and using local variables is good way to writing thread-safe code in Java.

8) In order to avoid thread-safety issue minimize sharing of objects between multiple thread.

9) [Volatile keyword in Java](http://javarevisited.blogspot.com/2011/06/volatile-keyword-java-example-tutorial.html) can also be used to instruct thread not to cache variables and read from main memory and can also instruct JVM not to reorder or optimize code from threading perspective.

That’s all on **how to write thread safe class or code in Java** and avoid serious concurrency issues in Java. To be frank thread-safety is a little tricky concept to grasp, you need to think concurrently in order to catch whether a code is thread-safe or not. Also [JVM](http://javarevisited.blogspot.com/2011/12/jre-jvm-jdk-jit-in-java-programming.html) plays a spoiler since it can **reorder code** for optimization, so the code which looks sequential and runs fine in development environment not guaranteed to run similarly in production environment because JVM may ergonomically adjust itself as server JVM and perform more optimization and reorder which cause **thread-safety issues**.

**20) Difference between PATH and Classpath in Java?**  
Answer : PATH is a environment variable in Java which is used to help Java program to compile and run.To set the PATH variable we have to include JDK\_HOME/bin directory in PATH environment variable and also we cannot override this variable. On the other hand, [ClassPath variable](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html) is used by class loader to locate and load compiled Java codes stored in .class file. We can set classpath we need to include all those directory where we have put either our .class file or JAR file which is required by your Java application,also we can override this environment variable.

10) Difference between Serializable and Externalizable in Java?  
Answer : If I say this is one of the most frequently asked Java question on both face-to-face and telephonic interview then it would be an exaggeration. Serialization is a default process of serializing or persisting any object's state in Java. It's triggered by implementing Serializable interface which is a marker interface (an interface without any method). While Externalizable is used to customize and control default serialization process which is implemented by application. Main difference between these two is that Externalizable interface provides complete control to the class implementing the interface whereas Serializable interface normally uses default implementation to handle the object serialization process.  
Externalizable interface has two method writeExternal(ObjectOutput) and readExternal(ObjectInput) method which are used to handle customized object serialize process and in terms of performance its good because everything is under control. to learn more about this classical question, see this [answer](http://java67.blogspot.sg/2012/10/difference-between-serializable-vs-externalizable-interface.html) as well.

How ClassLoader works in Java

**Delegation principles**

As discussed on [when a class is loaded and initialized in Java](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), a class is loaded in Java, when its needed. Suppose you have an application specific class called Abc.class, first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to Primordial or Bootstrap class loader. Primordial will look for that class in rt.jar and since that class is not there, request comes to Extension class loader which looks on

jre/lib/ext directory and tries to locate this class there, if class is found there than Extension class loader will load that class and Application class loader will never load that class but if its not loaded by extension class-loader than Application class loader loads it from [Classpath in Java](http://java67.blogspot.sg/2012/08/what-is-path-and-classpath-in-java-difference.html). Remember Classpath is used to load class files while [PATH](http://javarevisited.blogspot.ca/2011/10/how-to-set-path-for-java-unix-linux-and.html) is used to locate executable like javac or java command.V**isibility Principle**

According to visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. Which mean if class Abc is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either [java.lang.ClassNotFoundException](http://javarevisited.blogspot.ca/2011/08/classnotfoundexception-in-java-example.html).

**Uniqueness Principle**

According to this principle a class loaded by Parent should not be loaded by Child ClassLoader again. Though its completely possible to write class loader which violates Delegation and Uniqueness principles and loads class by itself, its not something which is beneficial. You should follow all class loader principle while writing your own ClassLoader.

How to load class explicitly in Java

Java provides API to explicitly load a class by Class.forName(classname) and Class.forName(classname, initialized, classloader), remember JDBC code which is used to load JDBC drives we have seen in [Java program to Connect Oracle database](http://javarevisited.blogspot.ca/2012/04/java-program-to-connect-oracle-database.html). As shown in above example you can pass name of ClassLoader which should be used to load that particular class along with binary name of class. Class is loaded by calling loadClass() method ofjava.lang.ClassLoader class which calls findClass() method to locate bytecodes for corresponding class. In this example Extension ClassLoader uses

java.net.URLClassLoader which search for class files and resources in [JAR](http://javarevisited.blogspot.ca/2012/10/5-ways-to-add-multiple-jar-to-classpath-java.html) and directories. any search path which is ended using "/" is considered directory. If findClass() does not found the class than it throws [java.lang.ClassNotFoundException](http://javarevisited.blogspot.de/2012/03/jdbc-javalangclassnotfoundexception.html) and if it finds it calls defineClass() to convert bytecodes into a .class instance which is returned to the caller.

**What will happen if we put a key object in a HashMap which is already there?**  
This tricky Java question is part of another frequently asked question, How HashMap works in Java. HashMap is also a popular topic to create confusing and tricky question in Java. Answer of this question is if you put the same key again then it will replace the old mapping because HashMap doesn't allow duplicate keys. The Same key will result in the same hashcode and will end up at the same position in the bucket.

Each bucket contains a linked list of Map.Entry object, which contains both Key and Value. Now Java will take the Key object from each entry and compare with this new key using equals() method, if that return true then value object in that entry will be replaced by new value. See [How HashMap works in Java](http://java67.blogspot.sg/2013/06/how-get-method-of-hashmap-or-hashtable-works-internally.html) for more tricky Java questions from HashMap.

**What will happen if we put a key object in a HASHSET which is already there?**

The [docs](http://docs.oracle.com/javase/6/docs/api/java/util/HashSet.html#add%28E%29) are pretty clear on this: HashSet.add *doesn't* replace:

Adds the specified element to this set if it is not already present. More formally, adds the specified element e to this set if this set contains no element e2 such that (e==null ? e2==null : e.equals(e2)). *If this set already contains the element, the call leaves the set unchanged and returns false.*

But [HashMap.put](http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html#put%28K,%20V%29) *will* replace:

If the map previously contained a mapping for the key, the old value is replaced.

The first thing you need to know is that HashSet acts like a Set, which means you add your object directly to the HashSet and it cannot contain duplicates. You just add your value directly in HashSet.

However, HashMap is a Map type. That means every time you add an entry, you add a key-value pair.

In HashMap you can have duplicate values, but not duplicate keys. In HashMap the new entry will replace the old one. The most recent entry will be in the HashMap.

**Understanding Link between HashMap and HashSet:**

Remember, HashMap can not have duplicate keys. Behind the scene HashSet uses a HashMap.

When you attempt to add any object into a HashSet, this entry is actually stored as a key in the HashMap - the same HashMap that is used behind the scene of HashSet. Since this underlying HashMap needs a key-value pair, a dummy value is generated for us.

Now when you try to insert another duplicate object into the same HashSet, it will again attempt to be insert it as a key in the HashMap lying underneath. However, HashMap does not support duplicates. Hence, HashSet will still result in having only one value of that type. As a side note, for every duplicate key, since the value generated for our entry in HashSet is some random/dummy value, the key is not replaced at all. it will be ignored as removing the key and adding back the same key (the dummy value is the same) would not make any sense at all.

**Summary:**

HashMap allows duplicate values, but not keys. HashSet cannot contains duplicates.

**To play with whether the addition of an object is successfully completed or not, you can check the boolean value returned when you call .add() and see if it returns true or false. If it returned true, it was inserted.**

How ConcurrentHashMap is implemented in Java

ConcurrentHashMap is introduced as an alternative of Hashtable and provided all functions supported by Hashtable with an additional feature called "concurrency level", which allows ConcurrentHashMap to partition Map. ConcurrentHashMap allows multiple readers to read concurrently without any [blocking](http://javarevisited.blogspot.com/2012/02/what-is-blocking-methods-in-java-and.html). This is achieved by partitioning Map into different parts based on concurrency level and locking only a portion of Map during updates. Default concurrency level is 16, and accordingly Map is divided into 16 part and each part is governed with a different lock. This means, 16 thread can operate on Map simultaneously until they are operating on different part of Map. This makes

ConcurrentHashMap high performance despite keeping thread-safety intact. Though, it comes with a caveat. Since update operations like put(), remove(), putAll() or clear() is not [synchronized](http://javarevisited.blogspot.com/2011/04/synchronization-in-java-synchronized.html), **concurrent retrieval may not reflect most recent change on Map**.

In case of putAll() or clear(), which operates on whole Map, concurrent read may reflect insertion and removal of only some entries. Another important point to remember is iteration over CHM, [Iterator](http://javarevisited.blogspot.com/2011/10/java-iterator-tutorial-example-list.html) returned by keySet of ConcurrentHashMap are weekly consistent and they only reflect state of ConcurrentHashMap and certain point and may not reflect any recent change. Iterator of ConcurrentHashMap's keySet area also [fail-safe](http://javarevisited.blogspot.in/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and doesn’t throwConcurrentModificationExceptoin..

Default concurrency level is 16 and can be changed, by providing a number which make sense and work for you while creating ConcurrentHashMap. Since concurrency level is used for internal sizing and indicate number of concurrent update without contention, so, if you just have few writers or thread to update Map keeping it low is much better. ConcurrentHashMap also uses ReentrantLock to internally lock its segments.

ConcurrentHashMap putifAbsent example in Java

ConcurrentHashMap examples are similar to [Hashtable examples](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html), we have seen earlier, but worth knowing is the use of putIfAbsent() method. Many times we need to insert entry into Map if it's not present already, and we wrote following kind of code:

synchronized(map){

if (map.get(key) == null){

return map.put(key, value);

} else{

return map.get(key);

}

}

Though this code will work fine in [HashMap and Hashtable](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html), This won't work in ConcurrentHashMap; because, during put operation whole map is not locked, and while one thread is putting value, other thread's get() call can still return null which result in one thread overriding value inserted by other thread. Ofcourse, you can wrap whole code in [synchronized block](http://java67.blogspot.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html) and make it [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html) but that will only make your code single threaded. ConcurrentHashMap provides putIfAbsent(key, value) which does same thing but atomically and thus eliminates above race condition.

When to use ConcurrentHashMap in Java

ConcurrentHashMap is best suited when you have multiple readers and few writers. If writers outnumber reader, or writer is equal to reader, than performance of ConcurrentHashMap effectively reduces to [synchronized map](http://javarevisited.blogspot.com/2011/04/difference-between-concurrenthashmap.html) or [Hashtable](http://javarevisited.blogspot.com/2012/01/java-hashtable-example-tutorial-code.html). Performance of CHM drops, because you got to lock all portion of Map, and effectively each reader will wait for another writer, operating on that portion of Map. ConcurrentHashMap is a good choice for caches, which can be initialized during application start up and later accessed my many request processing threads. As javadoc states, CHM is also

a [good replacement of Hashtable](http://javarevisited.blogspot.sg/2013/02/concurrent-collections-from-jdk-56-java-example-tutorial.html) and should be used whenever possible, keeping in mind, that CHM provides slightly weeker form of synchronization than Hashtable.

Summary

Now we know What is ConcurrentHashMap in Java and when to use ConcurrentHashMap, it’s time to know and revise some important points about CHM in Java.

1. ConcurrentHashMap allows concurrent read and thread-safe update operation.

2. During the update operation, ConcurrentHashMap only locks a portion of Map instead of whole Map.

3. The concurrent update is achieved by internally dividing Map into the small portion which is defined by concurrency level.

4. Choose concurrency level carefully as a significantly higher number can be a waste of time and space and the lower number may introduce thread contention in case writers over number concurrency level.

5. All operations of ConcurrentHashMap are [thread-safe](http://javarevisited.blogspot.com/2012/12/how-to-create-thread-safe-singleton-in-java-example.html).

6. Since ConcurrentHashMap implementation doesn't lock whole Map, there is chance of read overlapping with update operations like put() and remove(). In that case result returned by get() method will reflect most recently completed operation from there start.

7. Iterator returned by ConcurrentHashMap is weekly consistent, [fail-safe](http://javarevisited.blogspot.com/2012/02/fail-safe-vs-fail-fast-iterator-in-java.html) and never throw ConcurrentModificationException. In Java.

8. ConcurrentHashMap doesn't allow null as key or value.

9. You can use ConcurrentHashMap in place of [Hashtable](http://javarevisited.blogspot.com/2010/10/difference-between-hashmap-and.html) but with caution as CHM doesn't lock whole Map.

10. During putAll() and clear() operations, the concurrent read may only reflect insertion or deletion of some entries.

That’s all on **What is ConcurrentHashMap in Java** and when to use it. We have also seen little bit about internal working of ConcurrentHashMap and how it achieves it’s thread-safety and better performance over Hashtable and synchronized Map. Use ConcurrentHashMap in Java program, when there will be more reader than writers and it’s a good choice for creating cache in Java as well.  
  
Read more: <http://javarevisited.blogspot.com/2013/02/concurrenthashmap-in-java-example-tutorial-working.html#ixzz4tEtsWdIL>

**Difference between Wait and Sleep method in Java Thread**  
  
1) First and most important difference between Wait and sleep method is that wait method must be called from synchronized context i.e. from synchronized method or block in Java. If you call wait method without synchronization, it will throw IllegalMonitorStateException in Java. On the other hand there is no requirement of synchronization for calling sleep method , you can call it normally.  
  
3) Third and another significant difference between wait and sleep in Java is that, wait() method releases the lock of object on which it has called, it does release other locks if it holds any while sleep method of Thread class does not release any lock at all.  
Where to use wait and sleep method in Java

By reading properties and behavior of wait and sleep method it's clear that wait() method should be used in conjunction with notify() or notifyAll() method and intended for communication between two threads in Java while Thread.sleep() method is a utility method to introduce short pauses during program or thread execution. Given the requirement of synchronization for wait, it should not be used just to introduce pause or sleep in Java.

In summary **wait and sleep method are completely different to each other** and have different use cases. Use wait() and notify() method for inter thread communication while use sleep() method for introducing small pause during thread execution. Also remember, that wait() method will release the lock acquired when it entered into synchronized block or method, but sleep() method will keep the lock with itself. So if you design require releasing the lock during wait period then use wait() and notify method otherwise just use sleep().  
  
Read more: <http://www.java67.com/2012/08/what-are-difference-between-wait-and.html#ixzz4tEukcBgt>

HASH CODE AND EQUALS()

You must override hashCode() in every class that overrides equals(). Failure to do so will result in a violation of the general contract for Object.hashCode(), which will prevent your class from functioning properly in conjunction with all hash-based collections, including HashMap, HashSet, and Hashtable.

Let's try to understand it with an example of what would happen if we override equals() without overriding hashCode() and attempt to use a Map.

Say we have a class like this and that two objects of MyClass are equal if their importantFieldis equal (with hashCode() and equals() generated by eclipse)

public class MyClass {

private final String importantField;

private final String anotherField;

public MyClass(final String equalField, final String anotherField) {

this.importantField = equalField;

this.anotherField = anotherField;

}

public String getEqualField() {

return importantField;

}

public String getAnotherField() {

return anotherField;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result

+ ((importantField == null) ? 0

: importantField.hashCode());

return result;

}

@Override

public boolean equals(final Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

final MyClass other = (MyClass) obj;

if (importantField == null) {

if (other.importantField != null)

return false;

} else if (!importantField.equals(other.importantField))

return false;

return true;

}

}

**Override only** **equals**

If only equals is overriden, then when you call myMap.put(first,someValue) first will hash to some bucket and when you call myMap.put(second,someOtherValue) it will hash to some other bucket (as they have a different hashCode). So, although they are equal, as they don't hash to the same bucket, the map can't realize it and both of them stay in the map.

Although it is not necessary to override equals() if we override hashCode(), let's see what would happen in this particular case where we know that two objects of MyClass are equal if their importantField is equal but we do not override equals().

**Override only** **hashCode**

Imagine you have this

MyClass first = new MyClass("a","first");

MyClass second = new MyClass("a","second");

If you only override hashCode then when you call myMap.put(first,someValue) it takes first, calculates its hashCode and stores it in a given bucket. Then when you call myMap.put(second,someOtherValue) it should replace first with second as per the [Map Documentation](http://java.sun.com/j2se/1.4.2/docs/api/java/util/Map.html#put%28java.lang.Object,%20java.lang.Object%29) because they are equal (according to the business requirement).

But the problem is that equals was not redefined, so when the map hashes second and iterates through the bucket looking if there is an object k such that second.equals(k) is true it won't find any as second.equals(first) will be false.

hashCode() :

If you only override hash-code method nothing will happen. Because it always return new hashCode for each object as an Object class.

equals() :

If you only override equal method, a.equals(b) is true it means the hashCode of a and b must be same but not happen. Because you did not override hashCode method.

Note : hashCode() method of Object class always return new hashCode for each object.

## So when you need to use your object in the hashing based collection, must override both equals() and hashCode().