Contents

[Multithreading, Concurrency and Thread basics Questions 1](#_Toc81254235)

[https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#axzz752aoT4ZX 1](#_Toc81254236)

[1) Can we make array volatile in Java? 1](#_Toc81254237)

[2) Can volatile make a non-atomic operation to atomic? 1](#_Toc81254238)

[4) What guarantee volatile variable provides? 2](#_Toc81254239)

[5) Which one would be easy to write? synchronization code for 10 threads or 2 threads? 2](#_Toc81254240)

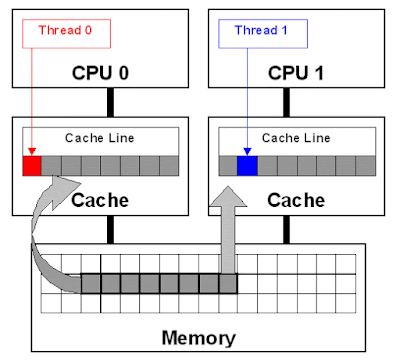
# Multithreading, Concurrency and Thread basics Questions

# https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#axzz752aoT4ZX

1. Can we make array volatile in Java?  
   This is one of the tricky Java multi-threading questions you will see in senior Java developer Interview. Yes, you can make an array volatile in Java but only the reference which is pointing to an array, not the whole array. What I mean, if one thread changes the reference variable to points to another array, that will provide a volatile guarantee, but if multiple threads are changing individual array elements they won't be having happens before guarantee provided by the volatile modifier.
2. Can volatile make a non-atomic operation to atomic?  
   This another good question I love to ask on volatile, mostly as a follow-up of the previous question. This question is also not easy to answer because volatile is not about atomicity, but there are cases where you can use a volatile variable to make the operation atomic.  
     
   One example I have seen is having a long field in your class. If you know that a long field is accessed by more than one thread e.g. a counter, a price field or anything, you better make it volatile. Why? because reading to a long variable is not atomic in Java and done in two steps, If one thread is writing or updating long value, it's possible for another thread to see half value (fist 32-bit). While reading/writing a volatile long or double (64 bit) is atomic.  
     
     
     
   3) What are practical uses of volatile modifier?  
   One of the practical use of the volatile variable is to make reading double and long atomic. Both double and long are 64-bit wide and they are read in two parts, first 32-bit first time and next 32-bit second time, which is non-atomic but volatile double and long read is atomic in Java. Another use of the volatile variable is to provide a memory barrier, just like it is used in Disrupter framework. Basically, Java Memory model inserts a write barrier after you write to a volatile variable and a read barrier before you read it. Which means, if you write to volatile field then it's guaranteed that any thread accessing that variable will see the value you wrote and anything you did before doing that right into the thread is guaranteed to have happened and any updated data values will also be visible to all threads, because the memory barrier flushed all other writes to the cache.

4) What guarantee volatile variable provides?([answer](http://java67.blogspot.sg/2012/08/what-is-volatile-variable-in-java-when.html))  
volatile variables provide the guarantee about ordering and visibility e.g. volatile assignment cannot be re-ordered with other statements but in the absence of any synchronization instruction compiler, JVM or JIT are free to reorder statements for better performance. volatile also provides the happens-before guarantee which ensures changes made in one thread is visible to others. In some cases volatile also provide atomicity e.g. reading 64-bit data types like long and double are not atomic but read of volatile double or long is atomic.  
  
Read more: <https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#ixzz752fz7M3r>

5) Which one would be easy to write? synchronization code for 10 threads or 2 threads?  
In terms of writing code, both will be of same complexity because synchronization code is independent of a number of threads. Choice of synchronization though depends upon a number of threads because the number of thread present more contention, so you go for advanced synchronization technique e.g. lock stripping, which requires more complex code and expertise.  
  
  
6) How do you call wait() method? using if block or loop? Why?([answer](http://javarevisited.blogspot.sg/2015/07/how-to-use-wait-notify-and-notifyall-in.html))  
wait() method should always be called in loop because it's possible that until thread gets CPU to start running again the condition might not hold, so it's always better to check condition in loop before proceeding. Here is the standard idiom of using wait and notify method in Java:

1. // The standard idiom for using the wait method
2. synchronized (obj) {
3. while (condition does not hold)
4. obj.wait(); // (Releases lock, and reacquires on wakeup)
5. ... // Perform action appropriate to condition
6. }
7. See [Effective Java Item 69](http://www.amazon.com/dp/0321356683/?tag=javamysqlanta-20) to learn more about why wait method should call in the loop.  
     
     
   7)  What is false sharing in the context of multi-threading?   
   false sharing is one of the well-known performance issues on multi-core systems, where each process has its local cache. false sharing occurs when threads on different processor modify variables that reside on same cache line as shown in the following image:
8. [](https://2.bp.blogspot.com/-Tze9foqpb74/VepwCzXHGCI/AAAAAAAADtM/i4KQDaefqk4/s1600/False%2BSharing%2Bin%2BMulti-threaded%2Bapplication.gif)
9. False sharing is very hard to detect because the thread may be accessing completely different global variables that happen to be relatively close together in memory. Like many concurrency issues, the primary way to avoid false sharing is careful code review and aligning your data structure with the size of a cache line.  
     
     
   8) What is busy spin? Why should you use it?  
   Busy spin is one of the technique to wait for events without releasing CPU. It's often done to avoid losing data in CPU cached which is lost if the thread is paused and resumed in some other core. So, if you are working on low latency system where your order processing thread currently doesn't have any order, instead of sleeping or calling wait(), you can just loop and then again check the queue for new messages. It's only beneficial if you need to wait for a very small amount of time e.g. in micro seconds or nano seconds. [LMAX Disrupter](http://lmax-exchange.github.io/disruptor/) framework, a high-performance inter-thread messaging library has a BusySpinWaitStrategy which is based on this concept and uses a busy spin loop for EventProcessors waiting on the barrier.  
     
     
   9) How do you take thread dump in Java?  
   You can take a thread dump of Java application in Linux by using **kill -3 PID**, where PID is the process id of Java process. In Windows, you can press **Ctrl + Break**. This will instruct JVM to print thread dump in standard out or err and it could go to console or log file depending upon your application configuration. If you have used Tomcat then when  
     
     
     
   10) is Swing thread-safe?([answer](http://javarevisited.blogspot.sg/2013/08/why-swing-is-not-thread-safe-in-java-Swingworker-Event-thread.html))  
   No, Swing is not thread-safe. You cannot update Swing components e.g. JTable, JList or JPanel from any thread, in fact, they must be updated from GUI or AWT thread. That's why swings provide invokeAndWait() and invokeLater() method to request GUI update from any other threads. This methods put update request in AWT threads queue and can wait till update or return immediately for an asynchronous update. You can also check the detailed answer to learn more.  
     
     
   11) What is a thread local variable in Java? ([answer](http://javarevisited.blogspot.sg/2012/05/how-to-use-threadlocal-in-java-benefits.html))  
   Thread-local variables are variables confined to a thread, its like thread's own copy which is not shared between multiple threads. Java provides a ThreadLocal class to support thread-local variables. It's one of the many ways to achieve thread-safety. Though be careful while using thread local variable in manged environment e.g. with web servers where worker thread out lives any application variable. Any thread local variable which is not removed once its work is done can potentially cause a memory leak in Java application.  
     
     
   12) Write wait-notify code for producer-consumer problem? ([answer](http://java67.blogspot.sg/2012/12/producer-consumer-problem-with-wait-and-notify-example.html))  
   Please see the answer for a code example. Just remember to call wait() and notify() method from synchronized block and test waiting for condition on the loop instead of if block.  
     
     
   13) Write code for thread-safe Singleton in Java? ([answer](http://javarevisited.blogspot.in/2012/12/how-to-create-thread-safe-singleton-in-java-example.html))  
   Please see the answer for a code example and step by step guide to creating thread-safe singleton class in Java. When we say thread-safe, which means Singleton should remain singleton even if initialization occurs in the case of multiple threads. Using Java enum as Singleton class is one of the easiest ways to create a thread-safe singleton in Java.  
     
     
   14) The difference between sleep and wait in Java?([answer](http://java67.blogspot.sg/2012/08/what-are-difference-between-wait-and.html))  
   Though both are used to pause currently running thread, sleep() is actually meant for short pause because it doesn't release lock, while wait() is meant for conditional wait and that's why it release lock which can then be acquired by another thread to change the condition on which it is waiting.  
     
     
   15) What is an immutable object? How do you create an Immutable object in Java? ([answer](http://javarevisited.blogspot.sg/2013/03/how-to-create-immutable-class-object-java-example-tutorial.html))  
   Immutable objects are those whose state cannot be changed once created. Any modification will result in a new object e.g. String, Integer, and other wrapper class. Please see the answer for step by step guide to creating Immutable class in Java.  
     
     
   16) Can we create an Immutable object, which contains a mutable object?  
   Yes, its possible to create an Immutable object which may contain a mutable object, you just need to be a little bit careful not to share the reference of the mutable component, instead, you should return a copy of it if you have to. Most common example is an Object which contain the reference of java.util.Date object.  
     
     
     
   Read more: <https://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#ixzz752gByT5G>

J2EE 1

Servlets, JDBC 1

Spring core 1

Spring MVC 1

Spring AOP 1

Spring Boot 1

Spring Data 1

Design Patterns 1

SOLID 1

REST 1

Jersey, Jackson, RestTemplate, HttpClient, RestAssured, Spring Rest 1

Hibernate 1

Angular 1

Build, configuration, pipeline 1

Hello