

[Courses](#)[Login](#)[Write an Article](#)

volatile keyword in Java

Using volatile is yet another way (like synchronized, atomic wrapper) of making class thread safe. Thread safe means that a method or class instance can be used by multiple threads at the same time without any problem.

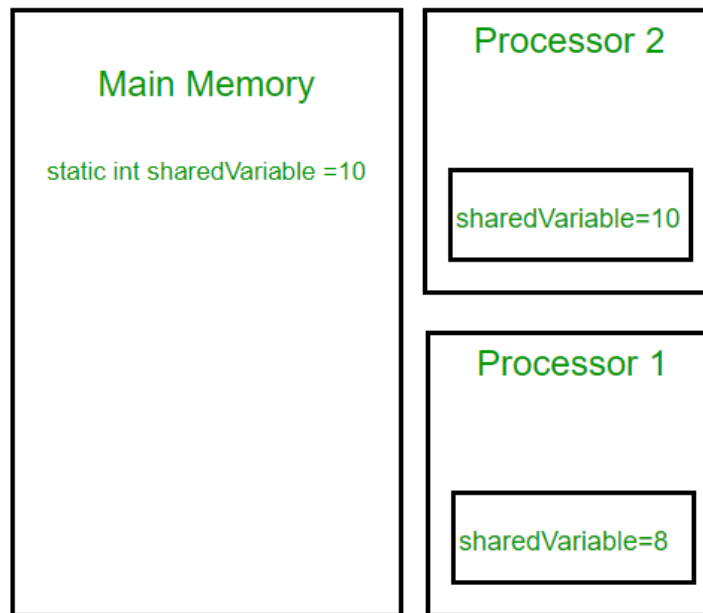
Consider below simple example.

```
class SharedObj
{
    // Changes made to sharedVar in one thread
    // may not immediately reflect in other thread
    static int sharedVar = 6;
}
```

Suppose that two threads are working on **SharedObj**. If two threads run on different processors each thread may have its own local copy of **sharedVariable**. If one thread modifies its value the change might not reflect in the original one in the main memory instantly. This depends on the **write policy** of cache. Now the other thread is not aware of the modified value which leads to data inconsistency.

Below diagram shows that if two threads are run on different processors, then value of **sharedVariable** may be different in different threads.





Note that write of normal variables without any synchronization actions, might not be visible to any reading thread (this behavior is called **sequential consistency**). Although most modern hardware provide good cache coherence therefore most probably the changes in one cache are reflected in other but it's not a good practice to rely on hardware for to 'fix' a faulty application.

1 **Top IT MNCs hiring freshers - CTC:25k-5Lac/Month Exp:3-15+**

2 **Apply For Your Dream Job** Search From Over 3 Lacs Jobs Across AI

```
class SharedObj
{
    // volatile keyword here makes sure that
    // the changes made in one thread are
    // immediately reflect in other thread
    static volatile int sharedVar = 6;
}
```

Note that volatile should not be confused with static modifier. static variables are class members that are shared among all objects. There is only one copy of them in main memory.

volatile vs synchronized:

Before we move on let's take a look at two important features of locks and synchronization.

1. **Mutual Exclusion:** It means that only one thread or process can execute a block of code (critical section) at a time.



2. **Visibility:** It means that changes made by one thread to shared data are visible to other threads.

Java's synchronized keyword guarantees both mutual exclusion and visibility. If we make the blocks of threads that modifies the value of shared variable synchronized only one thread can enter the block and changes made by it will be reflected in the main memory. All other thread trying to enter the block at the same time will be blocked and put to sleep.

In some cases we may only desire the visibility and not atomicity. Use of synchronized in such situation is an overkill and may cause scalability problems. Here volatile comes to the rescue. Volatile variables have the visibility features of synchronized but not the atomicity features. The values of volatile variable will never be cached and all writes and reads will be done to and from the main memory. However, use of volatile is limited to very restricted set of cases as most of the times atomicity is desired. For example a simple increment statement such as `x = x + 1;` or `x++` seems to be a single operation but is really a compound read-modify-write sequence of operations that must execute atomically.

volatile in Java vs C/C++:

Volatile in java is different from "volatile" qualifier in C/C++. For Java, "volatile" tells the compiler that the value of a variable must never be cached as its value may change outside of the scope of the program itself. In C/C++, "volatile" is needed when developing embedded systems or device drivers, where you need to read or write a memory-mapped hardware device. The contents of a particular device register could change at any time, so you need the "volatile" keyword to ensure that such accesses aren't optimized away by the compiler.

References:

<https://www.ibm.com/developerworks/java/library/j-jtp06197/>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/atomic.html>

<http://tutorials.jenkov.com/java-concurrency/volatile.html>

<https://pveentjer.wordpress.com/2008/05/17/jmm-thank-god-or-the-devil-for-strong-cache-coherence/>

This article is contributed by Sulabh Kumar. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



1 Top IT MNCs hiring freshers - CTC:25k-5Lac/Month Exp:3-15+

2 Apply For Your Dream Job Search From Over 3 Lacs Jobs Across AI

Recommended Posts:

Java.util.BitSet class methods in Java with Examples | Set 2

Shadowing of static functions in Java

How does default virtual behavior differ in C++ and Java ?

How are Java objects stored in memory?

How are parameters passed in Java?

Are static local variables allowed in Java?

final variables in Java

Default constructor in Java

Assigning values to static final variables in Java

Comparison of Exception Handling in C++ and Java



Does Java support goto?

Arrays in Java

Inheritance and constructors in Java

More restrictive access to a derived class method in Java

Comparison of static keyword in C++ and Java

 **Top IT MNCs hiring freshers - CTC:25k-5Lac/**
 Engineering, BCA, MCA, BSc Graduates & Diploma Holders c

Article Tags : Java

Practice Tags : Java



12

☐ To-do ☐ Done

3.1

Based on 63 vote(s)



Feedback/ Suggest Improvement

Add Notes

Improve Article