10 points about volatile modifier or field in Java

The `volatile` modifier has always been an interesting and tricky topic to many Java programmers. I still feel that it's one of the most underutilized modifiers in Java, which can do a lot of good if understood and applied correctly, after all, it provides a *lock-free* way to achieve synchronization in Java. If a field is shared between multiple threads and one of them change its value i.e. one thread reads from the field which is written by other threads, then, by using a volatile modifier, you can synchronize access to this field. The volatile modifier in Java provides **visibility** and **ordering** guarantee without any locking. You might know that compiler and JVM can re-order your code due to various reasons e.g. performance improvement which can be a problem in concurrent Java application.

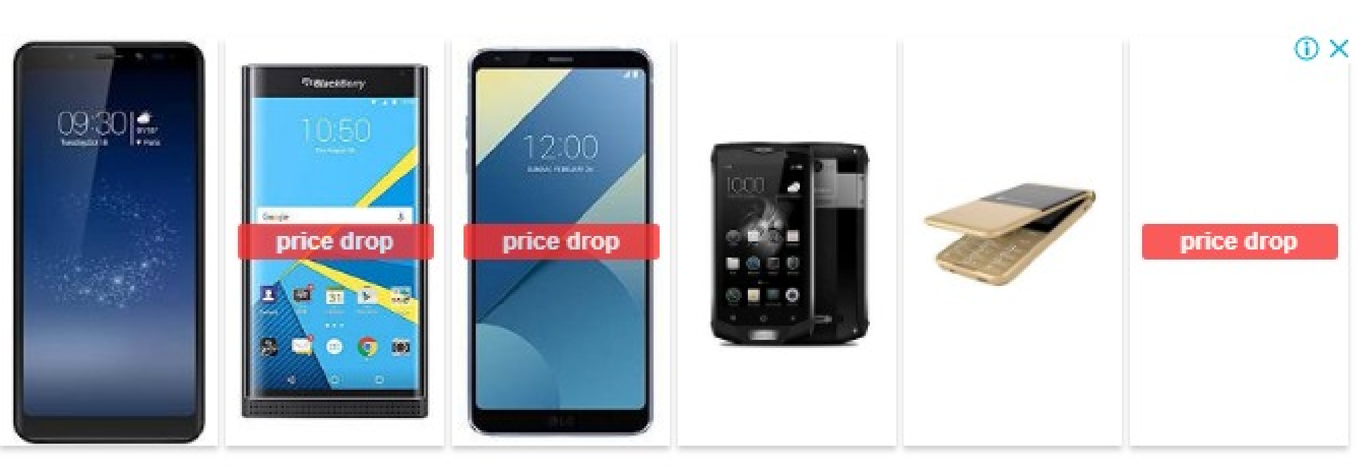
Custom SoCs Made Easy - Get Custom Chips for Dummies
Your guide to 80% BOM reduction, higher profit margins and quick time to market.
pages.arm.com

By making a filed volatile in Java you can instruct compiler, JVM, and JIT to doesn't reorder them preventing subtle and hard-to-find multi-threading bugs.

Similarly, the visibility guarantee ensures that memory barriers are refreshed when a volatile variable is read, hence all the changes made by one thread before writing into a volatile variable is visible to the other thread who read from a volatile variable, this is also a part of "happens-before" guarantee provided by volatile variables.

Though, you need to be a little bit careful because volatile doesn't provide **atomicity** and **mutual exclusive** access, which is one of the key difference between synchronized and volatile keyword in Java. Hence, the volatile variable should only be used if the assignment is the only operation performed on them.

In this article, I am going to 10 such important points about volatile modifier in a field which will help you to learn this useful concept better in Java multi-threading world. If you want to learn more, you can always read Java concurrency in Practice by Brian Goetz.



amazon.in

10 Points about volatile variable in Java

Here are a couple of useful points a Java developer should know about volatile modifier. Since many of you might not have the first-hand experience of using volatile variable in your application, this points will help you to understand the existing code written by some experienced programmers and which uses the volatile modifier for synchronization and inter-thread communication.

1) The `volatile` modifier provides **lock-free synchronization** of fields. Unlike `synchronized` keyword, when a thread read from the volatile variable or write into the volatile field, no lock is acquired or released.

2) The `volatile` modifier can only be applied to a field. You cannot apply volatile keyword with methods and local variables. Of course, you don't need any synchronization for local variables because they are not shared between multiple threads. Every thread has their own copy of local variables.

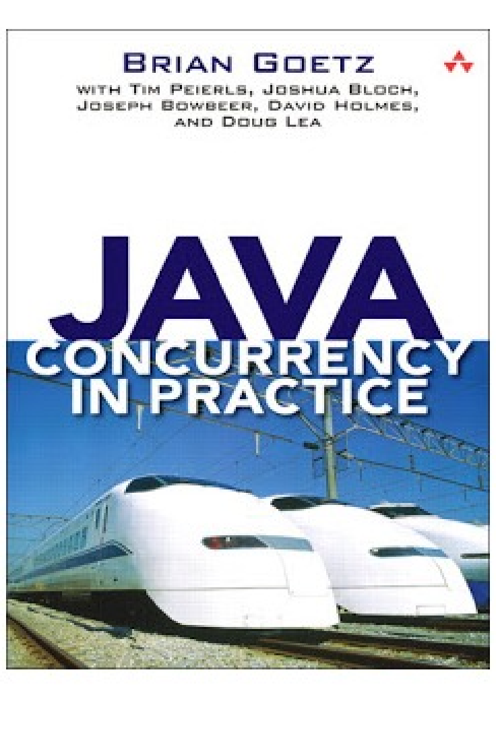
3) When you make a field volatile in Java, it signals compiler and Java virtual machine that this field may be concurrently updated by other threads. Due to this reason, compiler stops re-ordering instructions for maximum throughput involving the `volatile` field.

ASP.net Controls
Build intuitive, full-featured business applications for any web browser

4) Apart from ordering, `volatile` modifier also provides visibility guarantee. Any change made to the volatile variable is visible to all threads. The value of the volatile variable is not cached by threads, instead, they are alway read from the main memory.

5) The `volatile` modifier also provides the **happens-before** guarantee, A write to volatile variable happens before any subsequent read. It also causes memory barrier to be flushed, which means all changes made by thread A before writing into the volatile variable will be visible to thread B when it read the value of the volatile field. Several high-performance concurrency frameworks e.g. LMAX Disrupter utilizes this property of volatile variable to achieve lock-free synchronization.

If you want to know more about *"happens-before"* rules, please read Java Concurrency in Practice by Brian Goetz, it has a nice list of all the actions covered under this rule and how they work under Java Memory model.


BRIAN GOETZ
WITH TIM PEIERLS, JOSHUA BLOCH, JOSEPH BOWBEER, DAVID HOLMES, AND DOUG LEA
JAVA
CONCURRENCY
IN PRACTICE

6) The `volatile` modifier provides a low-cost synchronization alternative of synchronized keyword, albeit it doesn't replace synchronized block or synchronized method because it doesn't provide `atomicity` or `mutual exclusion`. The low cost is because it's lock free. Threads are not blocked for lock and they don't spend time on acquiring and releasing the lock.

7) When to use the volatile variable is the most common question from many Java developers, even experienced developers ask this question. The reason being is the lack of opportunity to write concurrent code which makes use of the volatile variable. Well, one of the most common uses of the volatile variable is shared boolean flag. It is also one of the most popular Java concurrency interview questions for senior developers

Suppose an object has a boolean flag, `bExit`, which is set by one thread and queries by another thread. In the classical game loop scenario, a user can press exit button to set the value of `bExit` to true to stop the game. Here the thread which will set the `bExit = true` will be event listener thread and the thread which will read this value will be your game thread.

In this case, it's reasonable to declare the `bExit` field as volatile as shown below

```
private volatile boolean bExit;

public boolean isExit() {
   return bExit;
}

public void setExit(boolean exit){
  this.bExit = exit;
}
```

Another common use of the volatile field is in double checked locking pattern for implementing thread-safe Singleton in Java. If you don't use volatile on a shared field that it's possible that game thread will never see the change made by event listener thread.

volatile modifier and double checked locking in Java

8) Always remember, volatile fields do not provide any atomicity guarantee. For example, you cannot make a counter volatile and assume that `i++` will be atomic. Similarly, flipping a volatile boolean variable is not atomic

```
public void flip() {
  exit = !exit ; // not atomic
};
```

If you need atomicity, you should use Atomic classes from `java.util.concurrent.atomic` package e.g. `AtomicInteger` can be used as concurrent shared counter or you can use the plain old synchronized keyword to make compound statement atomic. If you have trouble understanding concurrency fundamentals e.g. locking, synchronization, mutual exclusion, atomicity, I strongly suggest to first read an introductory book in threading e.g. Java Threads: Understanding and Mastering Concurrent Programming by Scott Oaks.

9) You should only make a variable volatile if you perform an **atomic operation** on them e.g. assignment. Assigning a value to a boolean or int variables are atomic but reading and writing `long` and `double` is **not atomic** unless they are *volatile*. So, one more use of volatile keyword is to make the `long` and `double` assignment atomic in Java.

10) The key difference between `volatile` and `synchronized` modifier is a locking, the synchronized keyword need a lock but volatile is lock free. Due to this reason, the cost of synchronization is also less in the case of the volatile variable.

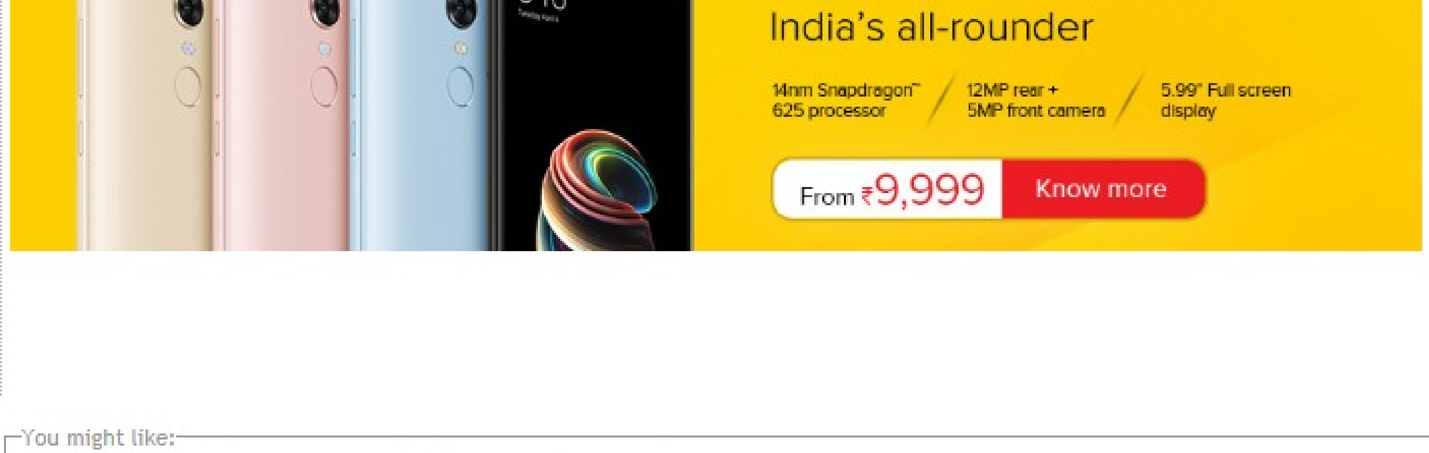
The second significant difference between synchronized and volatile modifier is *atomicity*, synchronized keyword provides the atomic guarantee and can make a block of code atomic but volatile variable doesn't provide such guarantee, except the case discussed in the last case of making long and double read atomic in Java.

That's all about **volatile modifier** in Java. There is a lot to learn as writing a concurrent application is not easy in Java but you can use these points to refresh your knowledge and understand the concept better. Use `volatile` modifier if you just need to synchronize access to shared variable whose value is set by one thread and queried by other. It provides a low-cost alternative to synchronized keyword or lock interface introduced in Java 5 without atomicity and mutual exclusion.

Further Learning
Multithreading and Parallel Computing in Java
Java Concurrency in Practice - The Book
Applying Concurrency and Multi-threading to Common Java Patterns
Java Concurrency in Practice Course by Heinz Kabutz

Other **10 points Java articles** you may like to explore:
10 points about Thread in Java (read)
10 points about Java Heap space or heap memory (read)
10 points about wait, notify, and notifyAll in Java (see)
10 points about static modifier in Java (see)
10 points about instanceof operator in Java (read)
10 points about Enum in Java (read)
10 points about finalize method in Java (tutorial)
10 points about java.lang.String class in Java (tutorial)

If one thing you can do this week to improve your understanding of multi-threading and concurrency in Java, then you should read Java Concurrency in Practice by Brian Goetz, one of the must-read book for any serious Java developer.


Redmi Note 5
India's all-rounder
14nm Snapdragon™ 625 processor
12MP rear + 5MP front camera
5.99" Full screen display
From ₹9,999
Know more

You might like:
- Top 20 String Algorithm Questions from Coding Interviews
- 3 Ways to Prevent Method Overriding in Java - Private, Static and Final
- How to Remove Duplicates from Array without using Java Collection API
- 3 ways to Find First Non Repeated Character in a String - Java Programming Problem

Recommended by

By Javin Paul
Labels: Java multithreading Tutorials

No comments :

Post a Comment

Newer Post | Home | Older Post

Subscribe to: Post Comments (Atom)

Interview Questions
- core java interview question (168)
- Coding Interview Question (66)
- data structure and algorithm (59)
- interview questions (48)
- SQL Interview Questions (30)
- design patterns (30)
- thread interview questions (30)
- object oriented programming (28)
- collections interview questions (25)
- database interview questions (17)
- spring interview questions (17)
- servlet interview questions (16)
- Programming interview question (7)
- hibernate interview questions (6)

Follow by Email
Email address... Submit

Best of Javarevisited
- 10 Tools Every Programmer Should Know
- 20 Libraries Java Programmers Should Know
- 10 Frameworks Java Developers Should Learn in 2018
- 20 Books Java Developers Should Learn in 2018
- 10 Things Java Developer Should Learn in 2018
- How Spring MVC works internally?
- How to design a vending machine in Java?
- 10 Tips to improve Programming Skill
- 10 OOP design principles programmer should know
- 10 tips to work fast in Linux
- 5 Books to improve Coding Skills

Followers

Categories
- SQL (50)
- linux (38)
- database (35)
- Eclipse (28)
- Java Certification OCPJP SCJP (23)
- general (23)
- JVM Internals (21)
- JQuery (17)
- courses (16)
- REST (14)
- Maven (11)
- Testing (11)

Subscribe to Download the E-book

Download The E-book | Building a REST API with Spring 4?
Email address... Submit

Pages
- Privacy Policy

Search This Blog
Search