**4. Difference between volatile, atomic, and synchronized in Java**

Most of the difference between these three synchronization construct comes from the fact mentioned above. Though all three offer the same memory semantics guaranteed by the [happens-before rule](https://javarevisited.blogspot.com/2020/01/what-is-happens-before-in-java-concurrency.html) and visibility guarantee, there is a significant difference between synchronized vs. atomic and volatile variables.  
  
1) The first and significant difference between synchronized keyword and volatile and atomic variables comes in terms of [**locking**](https://javarevisited.blogspot.com/2014/05/double-checked-locking-on-singleton-in-java.html). The synchronized keyword is used to implement a lock-based concurrent algorithm, and that's why to suffer from the limitation of locking. A volatile and atomic variable gives you the power to implement non-blocking algorithms, which is more scalable.  
  
2) On the difference between atomic and volatile variables, they provide the same **visibility guarantee**, but the atomic variable also provides the ability to make compound action, like the read-modify-write atomic. The volatile variable can not be used when one variable's value depends upon others or its own increment value.  
  
3) Atomic variables perform better because they use concurrency support provided by hardware for various atomic operations, like**compare-and-swap** or read-modify-write.  
  
4) Because of non-blocking nature, atomic variables are immune to concurrency hazards such as [deadlock](https://javarevisited.blogspot.com/2018/08/how-to-avoid-deadlock-in-java-threads.html) and livelock.  
  
5) Another significant difference between [synchronized](https://www.java67.com/2013/01/difference-between-synchronized-block-vs-method-java-example.html), [volatile](http://javarevisited.blogspot.sg/2017/02/10-points-about-volatile-modifier-and-field-in-java.html#axzz4wLVhZoF9), and [atomic variables](https://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html) comes in terms of performance. Atomic variables perform better than synchronized keyword because of no overhead of acquisition and release of the lock.  
  
This difference may not be visible in an uncontended environment, but when multiple threads try for the same lock and the same time, you can see performance degradation due to context switching, putting the thread on sleep, and then resuming, etc.  
  
Apart from the differences mentioned above, there is a couple of syntactic difference between synchronized, volatile and atomic variables, like **volatile keyword is only applicable to variables**, and s**ynchronized keyword can only be used with methods and blocks**.  
  
The key thing to learn is when to use volatile, synchronized, and atomic variables in Java, as incorrect understanding can cause serious bugs and performance impact. If you want to learn more or deep dive into Concurrency and Performance topic, I also recommend you check out [**Java Multithreading, Concurrency, and Performance Optimization**](https://click.linksynergy.com/deeplink?id=JVFxdTr9V80&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fjava-multithreading-concurrency-performance-optimization%2F) course on Udemy. It's one of the best course on the topic and highly recommend this course to every experienced Java programmers.

