**Thread Safety in Java**

Thread safety in java is the process to make our program safe to use in multithreaded environment, there are different ways through which we can make our program thread safe.

* Synchronization is the easiest and most widely used tool for thread safety in java.
* Use of Atomic Wrapper classes from java.util.concurrent.atomic package. For example AtomicInteger
* Use of locks from java.util.concurrent.locks package.
* Using thread safe collection classes, check this post for usage of [ConcurrentHashMap](https://www.journaldev.com/122/concurrenthashmap-in-java) for thread safety.
* Using volatile keyword with variables to make every thread read the data from memory, not read from thread cache.

1. String pool is possible only because String is immutable in Java. This way Java Runtime saves a lot of [heap space](https://www.journaldev.com/4098/java-heap-space-vs-stack-memory) because different String variables can refer to the same String variable in the pool.
2. If String is not immutable then it would cause a severe security threat to the application. For example, database username, password are passed as String to get database connection and in [socket programming](https://www.journaldev.com/741/java-socket-programming-server-client) host and port details passed as String. Since String is immutable, its value can’t be changed otherwise any hacker could change the referenced value to cause security issues in the application.
3. Since String is immutable, it is safe for [multithreading](https://www.journaldev.com/1079/multithreading-in-java). A single String instance can be shared across different threads. This avoids the use of synchronization for thread safety. Strings are implicitly thread-safe.
4. Since String is immutable, its **hashcode** is cached at the time of creation and it doesn’t need to be calculated again. This makes it a great candidate for the key in a Map and its processing is faster than other HashMap key objects. This is why String is the most widely used as HashMap keys.

**Immutable Class in Java**An immutable class is good for caching purpose because you don’t need to worry about the value changes. Other benefit of immutable class is that it is inherently [**thread-safe**](https://www.journaldev.com/1061/thread-safety-in-java), so you don’t need to worry about thread safety in case of multi-threaded environment.

<https://www.journaldev.com/129/how-to-create-immutable-class-in-java>

**Deep VS Shallow Copy:**

<https://www.baeldung.com/java-deep-copy>