Define :

Object whose internal state remains constant after it has been entirely created.

Immutability provides the below benefits

1. *Caching*
2. *Security*
3. *Synchronization*
4. *Performance*
5. *Reuse without replication*

* ***String is most widely used data structure***
* Caching the string literals and reusing saves lot of heap space because string variables refer the same object in the string pool. ***String intern pool***serves exactly the same purpose.
* ***Java String Pool*** is special memory region where Strings are stored in JVM.
* ***Interning*** : JVM optimizes the amount of memory allocated for them by storing only one copy of each literal string in the pool.

Security :

**If Strings were mutable, then by the time we execute the update, we can't be sure that the String we received, even after performing security checks, would be safe.** The untrustworthy caller method still has the reference and can change the String between integrity checks. Thus making our query prone to SQL injections in this case. So mutable Strings could lead to degradation of security over time.

Syncronization:

* You should only synchronize on objects that are actually shared between threads. If a String object is only being used by a single thread, there is no need to synchronize access to it.
* You should avoid using the String.class object as a synchronization lock. This can lead to deadlocks, because multiple threads could try to lock the String.class object at the same time.
* If you need to synchronize access to a String object that is being modified, you should use the StringBuffer or StringBuilder class instead. These classes are mutable, so they can be modified by multiple threads without causing race conditions.

Perfromance :

* Immutability: If you need to create a string that will be modified frequently, you should use the StringBuilder class instead of the String class. The StringBuilder class is mutable, which means that its contents can be changed without creating a new object. This can significantly improve the performance of applications that frequently modify strings.
* Interning: If you need to compare strings frequently, you should use the intern() method to get a reference to the canonical representation of a string. This can significantly improve the performance of string comparisons, as it avoids the need to compare two string objects that may be identical.
* String concatenation: If you need to concatenate strings frequently, you should use the StringBuffer class instead of the + operator. The StringBuffer class provides optimized methods for concatenating strings, which can significantly improve the performance of string concatenation.

The String class in Java is designed to be immutable, meaning once a String object is created, its state (i.e., the sequence of characters it represents) cannot be changed. This design decision was made for several important reasons:

Security: Strings are commonly used to store sensitive information, such as passwords or authentication tokens. By being immutable, the content of a String cannot be changed after creation. This property is crucial for security purposes, as it prevents unauthorized modification of sensitive data.

Thread Safety: Immutability ensures that String objects are thread-safe. Since multiple threads can access the same String object simultaneously, there is no risk of one thread modifying the content of the String while another thread is reading it. This makes String handling in multi-threaded environments much simpler and safer.

Caching and Performance: Because String objects are immutable, they can be cached and reused, which can lead to better performance and memory efficiency. When multiple references point to the same String value, the JVM can use the same instance for all the references, saving memory and reducing the overhead of creating new objects.

Hashing and Use in Hash Maps: The immutability of String objects makes them suitable for use as keys in hash-based data structures like HashMaps or HashSet. Since the hash code of an object is calculated based on its content, changing the content of a String after it has been used as a key in a hash map would lead to incorrect retrieval of values.

String Pool: Java maintains a String pool to store unique String literals to reduce memory consumption. The immutability of String objects ensures that they can be safely shared among different parts of the code without fear of unintentional modification.

Because of these advantages, the designers of Java chose to make the String class immutable by design. If you need to perform string manipulation operations that create new modified strings, Java provides other classes like StringBuilder or StringBuffer for mutable string operations.