# Why String is Immutable or Final in Java

The string is Immutable in Java because String objects are cached in String pool. Since cached String literals are shared between multiple clients there is always a risk, where one client's action would affect all another client. For example, if one client changes the value of String "Test" to "TEST", all other clients will also see that value. Since caching of String objects was important from performance reason this risk was avoided by making String class Immutable. At the same time, [String was made final](http://java67.blogspot.com/2014/01/why-string-class-has-made-immutable-or-final-java.html) so that no one can compromise invariant of String class e.g. Immutability, Caching, hashcode calculation etc. by extending and overriding behaviors. Another reason of why String class is immutable could die due to HashMap.

Since Strings are very popular as HashMap key, it's important for them to be immutable so that they can retrieve the value object which was stored in HashMap. Since [HashMap works in the principle of hashing](http://javarevisited.blogspot.com/2011/02/how-hashmap-works-in-java.html), which requires same has value to function properly. Mutable String would produce two different hashcodes at the time of insertion and retrieval if contents of String was modified after insertion, potentially losing the value object in the map.  
  
I think following two reasons make a lot of sense on why String class is made Immutable or final in Java:

1) Imagine String pool facility without making string immutable , its not possible at all because in case of string pool one string object/literal e.g. "Test" has referenced by many reference variables, so if any one of them change the value others will be automatically gets affected i.e. lets say  
  
String A = "Test"  
String B = "Test"  
  
Now String B called, "Test".toUpperCase() which change the same object into "TEST", so A will also be "TEST" which is not desirable

2) String has been widely used as parameter for many Java classes e.g. for opening network connection, you can pass hostname and port number as string, you can pass database URL as a string for opening database connection, you can [open any file in Java](http://javarevisited.blogspot.sg/2012/07/read-file-line-by-line-java-example-scanner.html) by passing the name of the file as argument to File I/O classes.  
  
In case, if String is not immutable, this would lead serious security threat, I mean someone can access to any file for which he has authorization, and then can change the file name either deliberately or accidentally and gain access to that file. Because of immutability, you don't need to worry about that kind of threats. This reason also gels with, Why String is final in Java, by making java.lang.String final, Java designer ensured that no one overrides any behavior of String class.

3)Since String is immutable it can safely share between many threads which is very important for multithreaded programming and to avoid any synchronization issues in Java, Immutability also makes String instance thread-safe in Java, means you don't need to synchronize String operation externally.

4) Another reason is to allow String to cache its hashcode, being immutable String in Java caches its hashcode, and do not calculate every time we call hashcode method of String, which makes it very fast as hashmap key to be used in hashmap in Java. In short because String is immutable, no one can change its contents once created which guarantees hashCode of String to be same on multiple invocations.

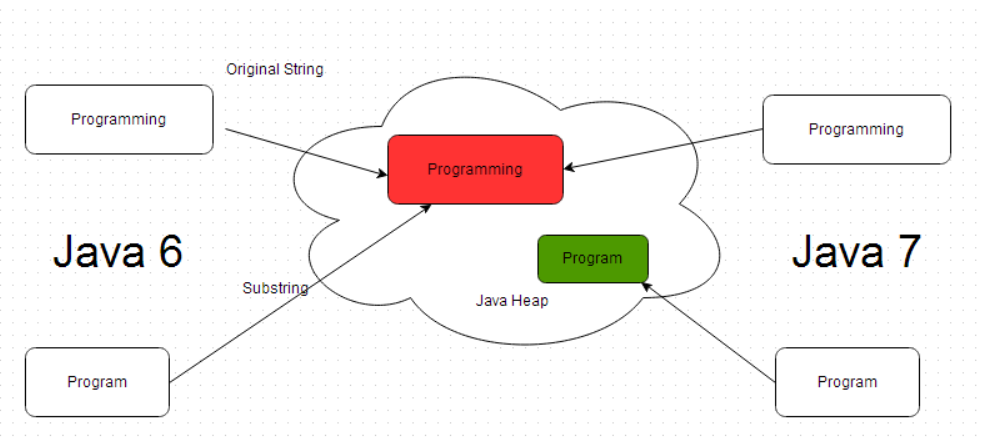
5) Another good reason is that it is used by the [class loading mechanism](http://javarevisited.blogspot.sg/2012/07/when-class-loading-initialization-java-example.html), and thus have profound and fundamental security aspects. Had String been mutable, a request to load "java.io.Writer" could have been changed to load "mil.vogoon.DiskErasingWriter"

# How SubString method works in Java - Memory Leak Fixed in JDK 1.7

Substring method is used to get parts of String in Java. It’s defined in java.lang.String class, and it's an [overloaded method](http://javarevisited.blogspot.com/2011/12/method-overloading-vs-method-overriding.html). One version of substring method takes just beginIndex, and returns part of String started from beginIndex till end, while other takes two parameters, beginIndex and endIndex, and returns part  of String starting from beginIndex to endIndex-1. He also stressed that every time you call  substring() method in Java,  it will return a new String because [String is immutable in Java](http://javarevisited.blogspot.com/2010/10/why-string-is-immutable-in-java.html).

If beginIndex is equal to length in substring(int beginIndex), it won't throw IndexOutOfBoundException instead it will return [empty String](http://javarevisited.blogspot.com/2013/02/5-ways-to-check-if-string-is-empty-in-java-examples.html). Same is the case when beginIndex and endIndex is equal, in case of second method. It will only throw StringIndexBoundException when beginIndex is negative, larger than endIndex or larger than length of String.

If you look substring method inside String class, you will figure out that it calls String (int offset, int count, char value []) [constructor](http://javarevisited.blogspot.com/2012/12/what-is-constructor-in-java-example-chainning-overloading.html) to create new String object. What is interesting here is, value[], which is the same character array used to represent original string. So what's wrong with this?  
  
In case If you have still not figured it out, If the original string is very long, and has array of size 1GB, no matter how small a substring is, it will hold 1GB array.  This will also stop original string to be [garbage collected](http://javarevisited.blogspot.com/2012/10/10-garbage-collection-interview-question-answer.html), in case if doesn't have any live reference. This is clear case of memory leak in Java, where memory is retained even if it's not required. That's how substring method creates memory leak.



This issue has been resolved in java 7.