Strings in Java

<https://www.geeksforgeeks.org/convert-a-string-to-character-array-in-java/>

## Class String

* [java.lang.Object](https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html)
  + java.lang.String
* **All Implemented Interfaces:**

[Serializable](https://docs.oracle.com/javase/7/docs/api/java/io/Serializable.html), [CharSequence](https://docs.oracle.com/javase/7/docs/api/java/lang/CharSequence.html" \o "interface in java.lang), [Comparable](https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html)<[String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html)>

Strings in Java are Objects that are backed internally by a char array. Since arrays are immutable(cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

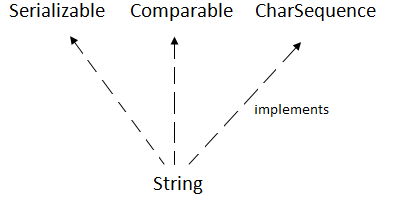
1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

is same as:

1. String s="javatpoint";

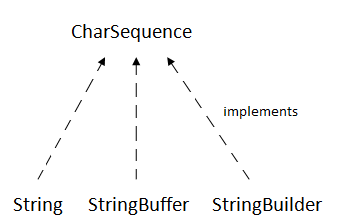
**Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements Serializable, Comparable and CharSequence [interfaces](https://www.javatpoint.com/interface-in-java).



## **CharSequence Interface**

The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

### **What is String in java**

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

### **How to create a string object?**

There are two ways to create String object:

1. By string literal
2. By new keyword

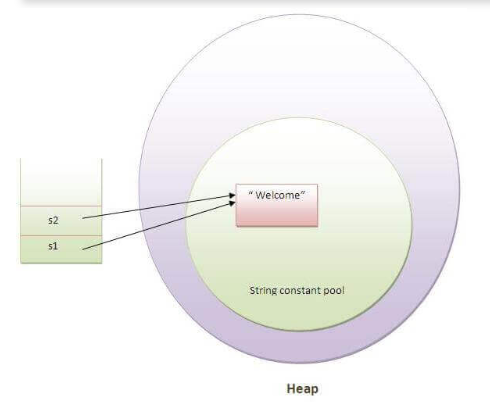
### **1) String Literal**

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, that is why it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance.

#### **String objects are stored in a special memory area known as the "string constant pool".**

### **Why Java uses the concept of String literal?**

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

### **2) By new keyword**

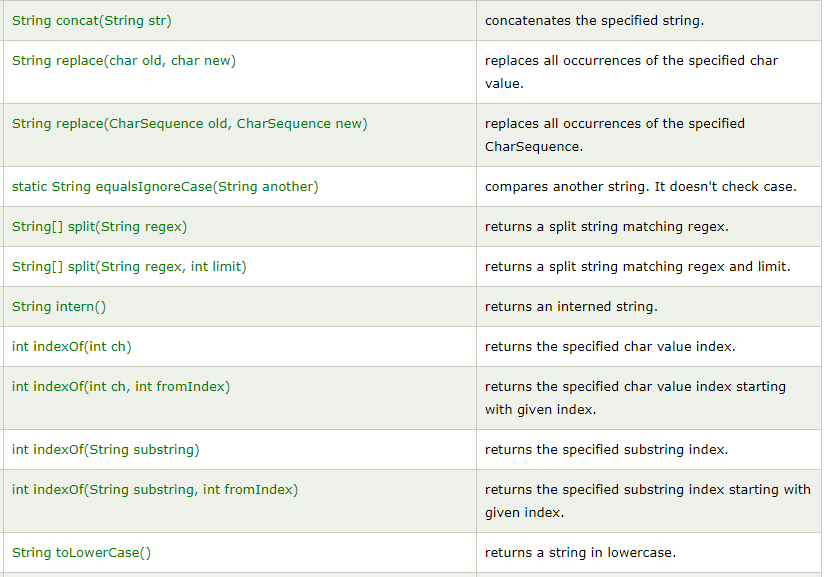
1. String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

### **Java String class methods**

The java.lang.String class provides many useful methods to perform operations on sequence of char values.







# Immutable String in Java

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once string object is created its data or state can't be changed but a new string object is created.

Let's try to understand the immutability concept by the example given below

**class** Testimmutablestring{

**public** **static** **void** main(String args[]){

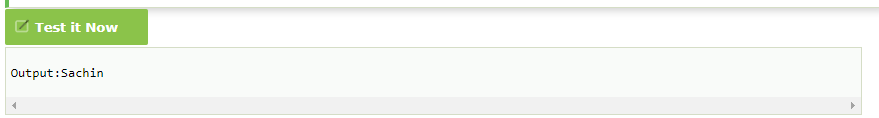
   String s="Sachin";

    s.concat(" Tendulkar");//concat() method appends the string at the end

    System.out.println(s);//will print Sachin because strings are immutable objects

 }

}



As you can see in the above figure that two objects are created but s reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

**class** Testimmutablestring1{

**public** **static** **void** main(String args[]){

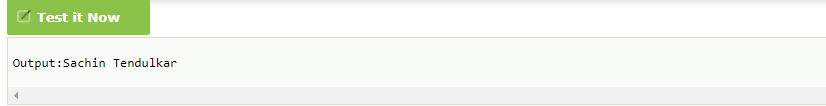
   String s="Sachin";

   s=s.concat(" Tendulkar");

   System.out.println(s);

 }

}



### **Why string objects are immutable in java?**

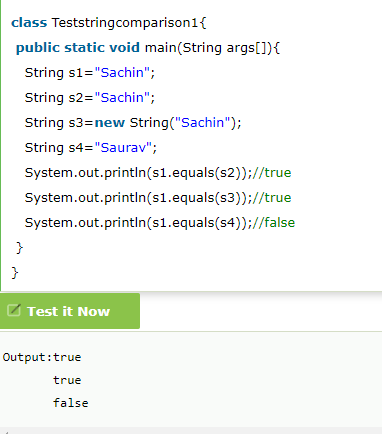
|  |
| --- |
| Because java uses the concept of string literal.Suppose  there are 5 reference variables,all referes to one object "sachin".  If one reference variable changes the value of the object,  it will be affected to all the reference variables. That is why string objects are immutable in java. Java String compare We can compare string in java on the basis of content and reference.  It is used in **authentication** (by equals() method), **sorting** (by compareTo() method)  , **reference matching** (by == operator) etc.  There are three ways to compare string in java: |

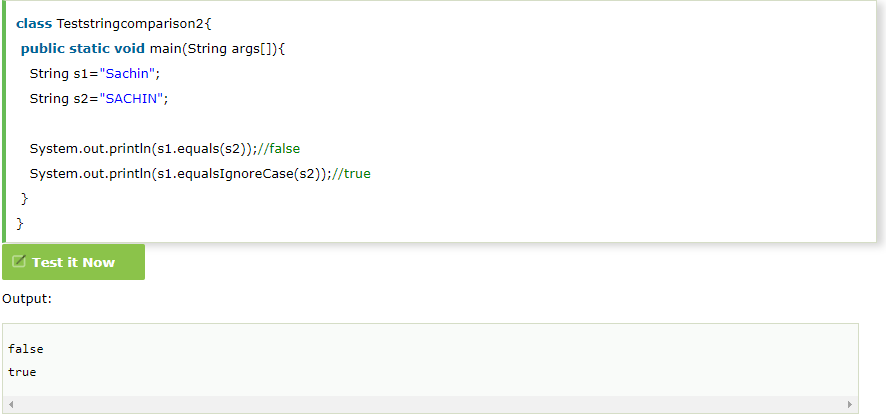
1. By equals() method
2. By = = operator
3. By compareTo() method

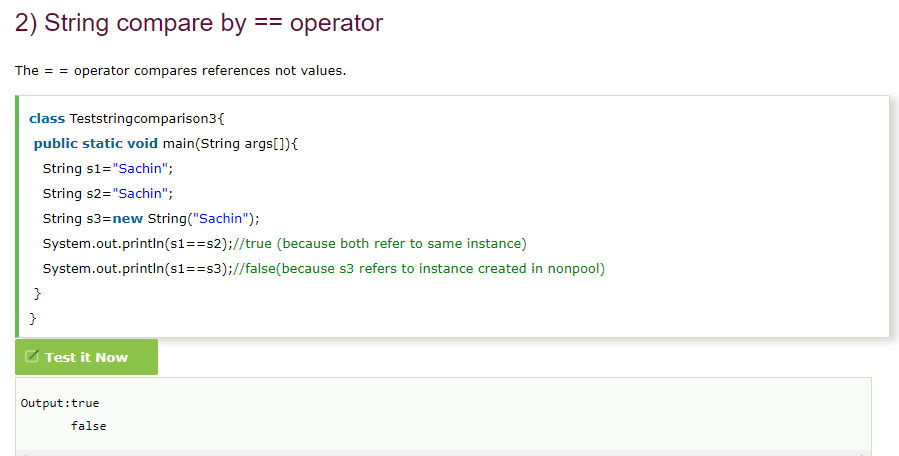
## **String compare by equals() method**

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

* **public boolean equals(Object another)** compares this string to the specified object.
* **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.





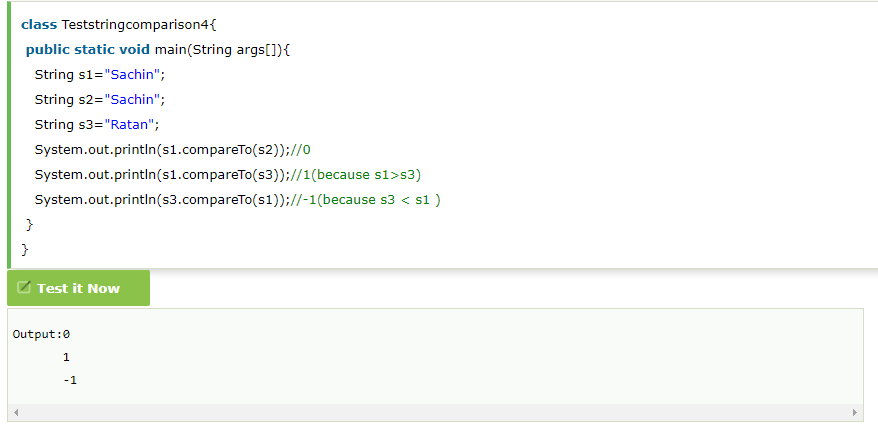


## **String compare by compareTo() method**

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

* **s1 == s2** :0
* **s1 > s2**  :positive value
* **s1 < s2**  :negative value



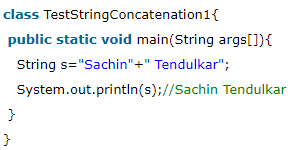
# String Concatenation in Java

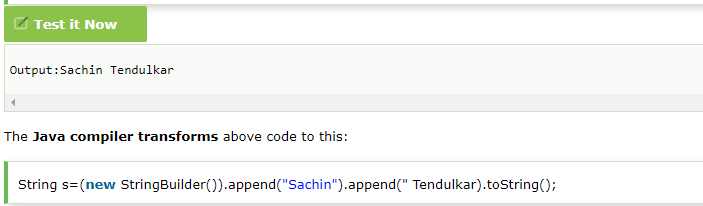
In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string in java:

By + (string concatenation) operator

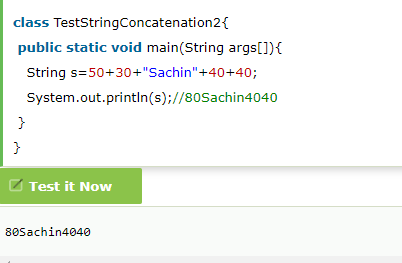
By concat() method

Java string concatenation operator (+) is used to add strings. For Example:





In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concat not only string but primitive values also. For Example:

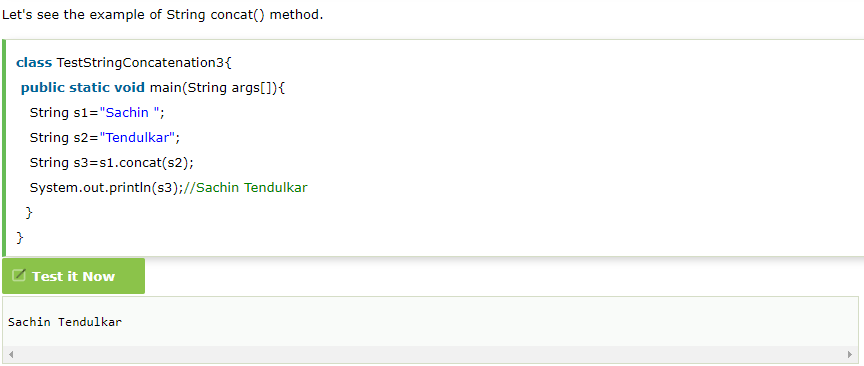


#### **Note: After a string literal, all the + will be treated as string concatenation operator.**

### **String Concatenation by concat() method**

The String concat() method concatenates the specified string to the end of current string. Syntax:

1. **public** String concat(String another)



# Substring in Java

A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

#### **Note: Index starts from 0.**

You can get substring from the given string object by one of the two methods:

1. **public String substring(int startIndex):** This method returns new String object containing the substring of the given string from specified startIndex (inclusive).
2. **public String substring(int startIndex, int endIndex):**This method returns new String object containing the substring of the given string from specified startIndex to endIndex.

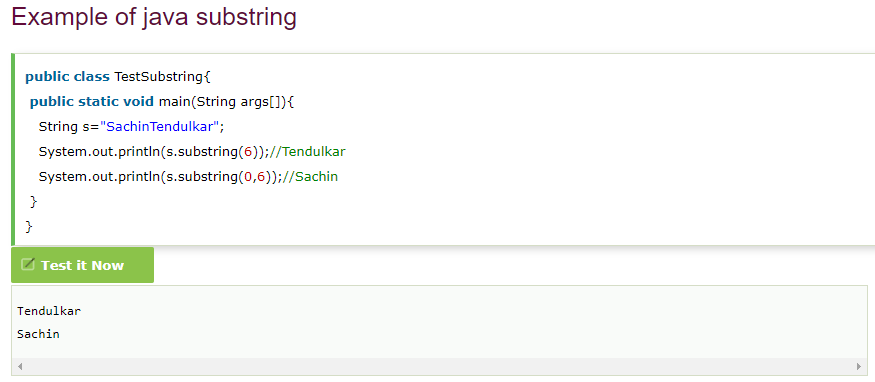
In case of string:

* **startIndex:** inclusive
* **endIndex:** exclusive

Let's understand the startIndex and endIndex by the code given below.

1. String s="hello";
2. System.out.println(s.substring(0,2));//he

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).



# Java String class methods

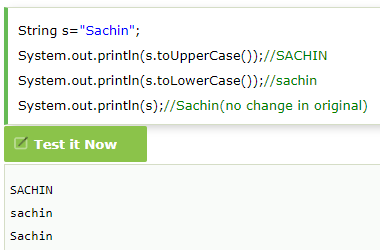
The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc.

Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

Let's see the important methods of String class.

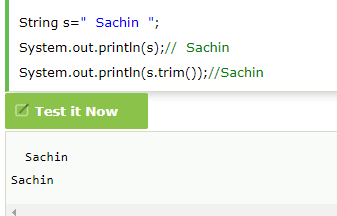
### **Java String toUpperCase() and toLowerCase() method**

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.



### **Java String trim() method**

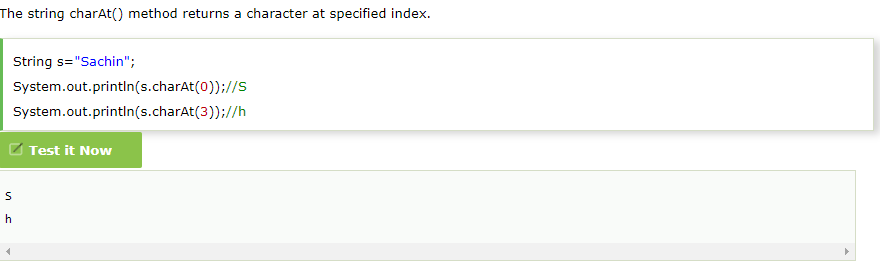
The string trim() method eliminates white spaces before and after string.



### **Java String startsWith() and endsWith() method**

1. String s="Sachin";
2. System.out.println(s.startsWith("Sa"));//true
3. System.out.println(s.endsWith("n"));//true

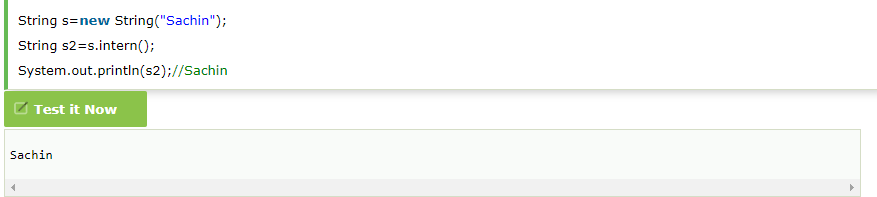
### **Java String charAt() method**



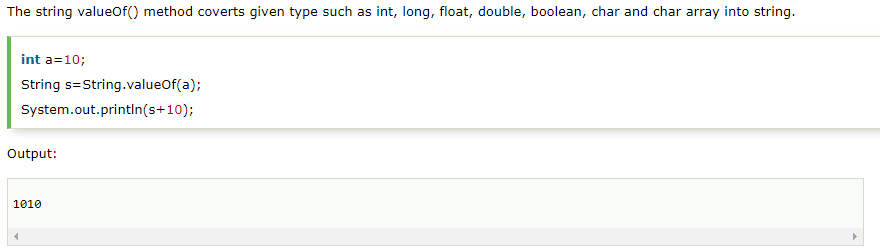
### **Java String intern() method**

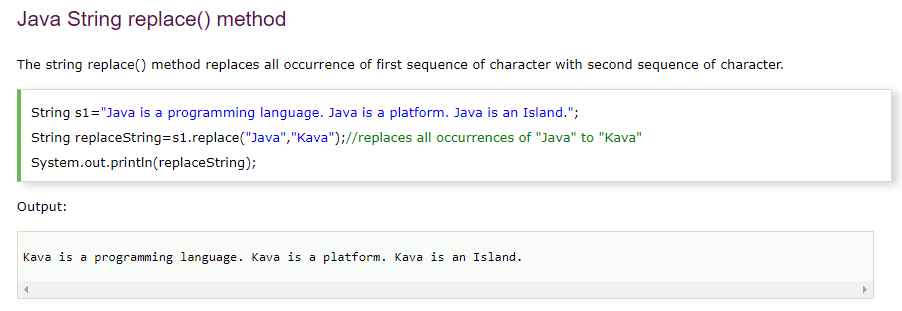
A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.



### **Java String valueOf() method**

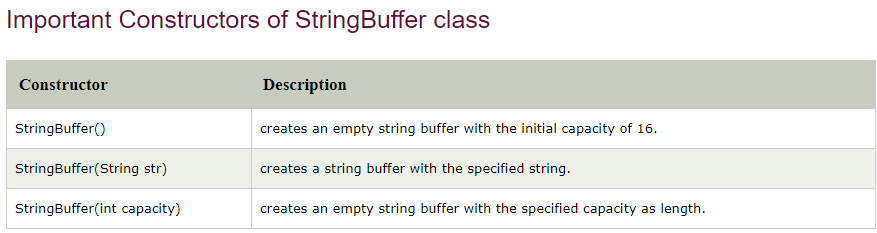




# Java StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

#### **Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.**

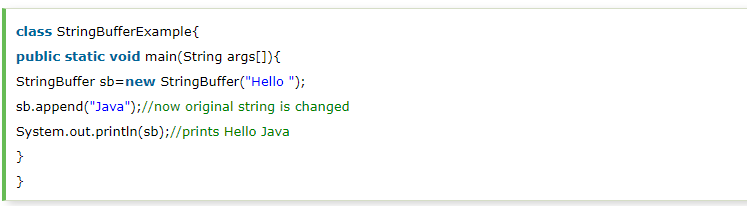


### **What is mutable string**

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

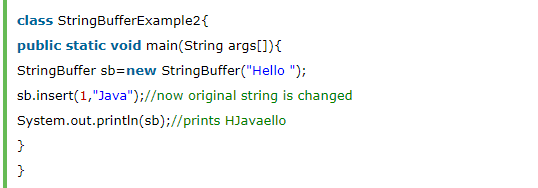
### **1) StringBuffer append() method**

The append() method concatenates the given argument with this string.



### **StringBuffer insert() method**

The insert() method inserts the given string with this string at the given position.



## String vs StringBuffer vs StringBuilder

### String in Java

1. String class represents character strings, we can instantiate String by two ways.  
   String str = "abc"; or String str = new String ("abc");
2. String is [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) in Java, so it’s easy to share it across different threads or functions.
3. When we create a String using double quotes, it first looks for the String with the same value in the JVM string pool, if found it returns the reference else it creates the String object and then places it in the String pool. This way JVM saves a lot of space by using the same String in different threads. But if a new operator is used, it explicitly creates a new String in the heap memory.
4. + operator is overloaded for String and used to concatenate two Strings. Although internally it uses StringBuffer to perform this action.
5. String overrides [equals() and hashCode()](https://www.journaldev.com/21095/java-equals-hashcode) methods, two Strings are equal only if they have the same characters in the same order. Note that equals() method is case sensitive, so if you are not looking for case sensitive checks, you should use equalsIgnoreCase() method.
6. A String represents a string in the UTF-16 format
7. String is a final class with all the fields as final except “private int hash”. This field contains the hashCode() function value and created only when the hashCode() method is called and then cached in this field. Furthermore, the hash is generated using final fields of String class with some calculations, so every time hashCode() method is called, it will result in the same output. For the caller, it’s like calculations are happening every time but internally it’s cached in the hash field.

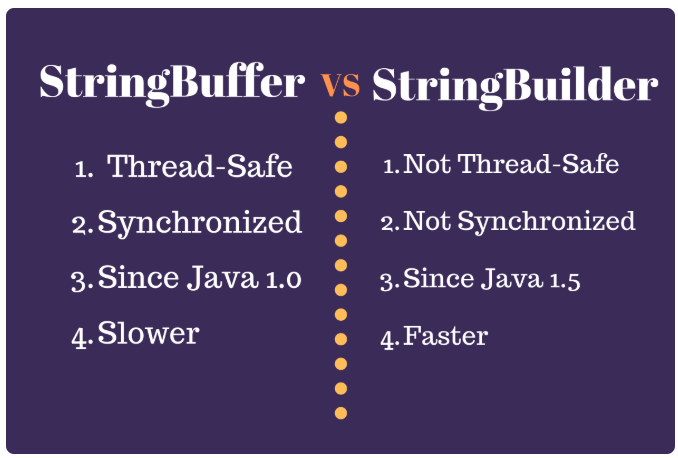
### String vs StringBuffer

Since String is immutable in Java, whenever we do String manipulation like concatenation, substring etc, it generates a new String and discards the older String for garbage collection.

These are heavy operations and generate a lot of garbage in heap. So Java has provided StringBuffer and StringBuilder class that should be used for String manipulation.

StringBuffer and StringBuilder are mutable objects in java and provide append(), insert(), delete() and substring() methods for String manipulation.

### StringBuffer vs StringBuilder



### String vs StringBuffer vs StringBuilder

1. String is immutable whereas StringBuffer and StringBuider are mutable classes.
2. StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why [StringBuilder is more faster than StringBuffer](https://www.journaldev.com/137/stringbuffer-vs-stringbuilder).
3. String concat + operator internally uses StringBuffer or StringBuilder class.
4. For String manipulations in non-multi threaded environment, we should use StringBuilder else use StringBuffer class.

## StringBuffer vs StringBuilder

1. [StringBuffer](https://www.journaldev.com/16827/stringbuffer-in-java) is the **thread safe** utility class to perform several operations on [Strings](https://www.journaldev.com/16928/java-string). It contains append() and insert() methods that are widely used to perform operations on Strings in a multi-thread environment. If you will check the source code, most of its functions are synchronized for [thread safety](https://www.journaldev.com/1061/thread-safety-in-java).

Since most of the String operations, for example, concatenation happens in a single thread environment, Java 1.5 introduced another utility class **[StringBuilder](https://www.journaldev.com/16833/java-stringbuilder" \t "_blank)** to perform similar operations but doesn’t provide thread safety. If you will look into its source code, all the methods are unsynchronized. This is the most important point for StringBuffer vs StringBuilder.

1. StringBuffer has some extra methods such as substring, length, capacity, trimToSize etc. However, these are not required since you have all these present in [String](https://www.journaldev.com/16928/java-string) too. That’s why these methods were never implemented in StringBuilder class.
2. StringBuffer was introduced in Java 1.0 whereas StringBuilder class was introduced in Java 1.5 after looking at shortcomings of StringBuffer.
3. StringBuilder is faster than StringBuffer because of no synchronization.

Note

**Thread safety** is a [computer programming](https://en.wikipedia.org/wiki/Computer_programming) concept applicable to [multi-threaded](https://en.wikipedia.org/wiki/Thread_(computing)) code. Thread-safe code only manipulates shared data structures in a manner that ensures that all threads behave properly and fulfill their design specifications without unintended interaction. There are various strategies for making thread-safe data structures.

## StringBuilder vs StringBuffer Performance

Here, I am trying to check the effect on performance because of synchronization with a sample program that performs append() and insert() on StringBuffer and StringBuilder object for multiple times.