<https://www.javatpoint.com/difference-between-stringbuffer-and-stringbuilder>

**final vs Immutability in Java**

**final :** In Java, [final](https://www.geeksforgeeks.org/final-keyword-java/) is a modifier which is used for class, method and variable also. When a variable is declared with final keyword, it’s value can’t be modified, essentially, a constant.

[**Immutability**](https://www.geeksforgeeks.org/create-immutable-class-java/) **:** In simple terms, immutability means unchanging over time or unable to be changed. In Java, we know that String objects are immutable means we cant change anything to the existing String objects.

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **final** | **finally** | **finalize** |
| 1) | Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed. | Finally is used to place important code, it will be executed whether exception is handled or not. | Finalize is used to perform clean up processing just before object is garbage collected. |
| 2) | Final is a keyword. | Finally is a block. | Finalize is a method. |

**Java final example**

1. class FinalExample{
2. public static void main(String[] args){
3. final int x=100;
4. x=200;//Compile Time Error
5. }}

## String vs StringBuffer vs StringBuilder

[](https://cdn.journaldev.com/wp-content/uploads/2012/10/difference-between-string-and-stringbuffer-and-stringbuilder.png)  
String is one of the most widely used class in java programming and one of the most important topic in core java interview. If you are writing a program which prints something on console, we use String.

This tutorial is aimed to focus on major features of String class and then we will compare **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(), reverse() and substring() methods for String manipulation.

### StringBuffer vs StringBuilder

StringBuffer was the only choice for String manipulation till Java 1.4 but it has one disadvantage that all of its public methods are synchronized. StringBuffer provides Thread safety but on a performance cost.

In most of the scenarios, we don’t use String in a multithreaded environment, so Java 1.5 introduced a new class StringBuilder that is similar to StringBuffer except thread safety and synchronization.

So if you are in a single threaded environment or don’t care about thread safety, you should use StringBuilder else use StringBuffer. See this post for [performance benchmarking between StringBuffer and StringBuilder](https://www.journaldev.com/137/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.
5. StringBuffer and StringBuilder are mutable objects in java and provide append(), insert(), delete(), reverse(), and substring() methods for String manipulation
6. A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |



## StringBuilder vs StringBuffer

1. Lets summarize the differences in detail:
2. 1) **Synchronization**: StringBuffer methods are synchronized while StringBuilder methods are non-synchronized, it means that for thread-safe operations you must choose StringBuffer class instead of StringBuilder.
3. 2) **Performance**: In a synchronized environment a single thread can perform a certain operation rather than distributing the work among multiple threads, which makes StringBuffer low performer as it is synchronized. StringBuilder performance is better than StringBuffer because it is not synchronized.
4. 3) **Which one to use**: Operations (without considering the performance) are almost same in both the classes which means there is nothing in StringBuffer which cannot be done using StringBuilder. As discussed above the main thing which you need to consider while making a choice is **thread-safety**, if you think that the operation should be thread-safe then use StringBuffer, in all other cases StringBuilder is a better choice as it offers you the same functionality with better performance.
5. **Similarities**:  
   Unlike String, both StringBuffer and StringBuilder are mutable (can be modified).
6. The **StringBuffer** and **StringBuilder** classes are used when there is a necessity to make a lot of modifications to Strings of characters.
7. Unlike Strings, objects of type StringBuffer and String builder can be modified over and over again without leaving behind a lot of new unused objects.
8. The StringBuilder class was introduced as of Java 5 and the main difference between the StringBuffer and StringBuilder is that StringBuilders methods are not thread safe (not synchronised).
9. It is recommended to use **StringBuilder** whenever possible because it is faster than StringBuffer. However, if the thread safety is necessary, the best option is StringBuffer objects.

## Example

1. [Live Demo](http://tpcg.io/7tGiXc)
2. public class Test {
3. public static void main(String args[]) {
4. StringBuffer sBuffer = new StringBuffer("test");
5. sBuffer.append(" String Buffer");
6. System.out.println(sBuffer);
7. }
8. }
9. This will produce the following result −

## Output

1. test String Buffer

https://www.geeksforgeeks.org/string-vs-stringbuilder-vs-stringbuffer-in-java/

# String vs StringBuilder vs StringBuffer in Java

**Prerequisites :**[String](https://www.geeksforgeeks.org/string-class-in-java/) , [Initialize a String](https://www.geeksforgeeks.org/how-to-initialize-and-compare-strings-in-java/)

Consider below code with three concatenation functions with three different types of parameters, String, StringBuffer and StringBuilder.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate difference between String,  // StringBuilder and StringBuffer  class Geeksforgeeks  {      // Concatenates to String      public static void concat1(String s1)      {          s1 = s1 + "forgeeks";      }        // Concatenates to StringBuilder      public static void concat2(StringBuilder s2)      {          s2.append("forgeeks");      }        // Concatenates to StringBuffer      public static void concat3(StringBuffer s3)      {          s3.append("forgeeks");      }        public static void main(String[] args)      {          String s1 = "Geeks";          concat1(s1);  // s1 is not changed          System.out.println("String: " + s1);            StringBuilder s2 = new StringBuilder("Geeks");          concat2(s2); // s2 is changed          System.out.println("StringBuilder: " + s2);            StringBuffer s3 = new StringBuffer("Geeks");          concat3(s3); // s3 is changed          System.out.println("StringBuffer: " + s3);      }  } |

**Output:**

String: Geeks

StringBuilder: Geeksforgeeks

StringBuffer: Geeksforgeeks

**Explanation:**  
**1. Concat1**: In this method, we pass a string “Geeks” and perform “s1 = s1 + ”forgeeks”. The string passed from main() is not changed, this is due to the fact that String is **immutable**. Altering the value of string creates another object and s1 in concat1() stores reference of new string. References s1 in main() and cocat1() refer to different strings.

**2. Concat2**: In this method, we pass a string “Geeks” and perform “s2.append(“forgeeks”)” which changes the actual value of the string (in main) to “Geeksforgeeks”. This is due to the simple fact that StringBuilder is **mutable** and hence changes its value.

**2. Concat3**: StringBuffer is similar to StringBuilder except one difference that StringBuffer is thread safe, i.e., multiple threads can use it without any issue. The thread safety brings a penalty of performance.  
**When to use which one :**

* If a string is going to remain constant throughout the program, then use String class object because a String object is immutable.
* If a string can change (example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.
* If a string can change, and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous so you have thread-safety.

**Conversion between types of strings in Java**

Sometimes there is a need of converting a string object of different classes like String, StringBuffer, StringBuilder to one-another. Below are some techniques to do the same.

1. **From String to StringBuffer and StringBuilder :**This one is easy. We can directly pass String class object to StringBuffer and StringBuilder class constructors. As String class is immutable in java, so for editing a string, we can perform same by converting it to StringBuffer or StringBuilder class objects. Below is the java program to demonstrate the same.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate conversion from  // String to StringBuffer and StringBuilder.  public class Test  {      public static void main(String[] args)      {          String str = "Geeks";            // conversion from String object to StringBuffer          StringBuffer sbr = new StringBuffer(str);          sbr.reverse();          System.out.println(sbr);            // conversion from String object to StringBuilder          StringBuilder sbl = new StringBuilder(str);          sbl.append("ForGeeks");          System.out.println(sbl);      }  } |

Output:

skeeG

GeeksForGeeks

1. **From StringBuffer and StringBuilder to String :** This conversions can be perform using toString() method which is overridden in both StringBuffer and StringBuilder classes.  
   Below is the java program to demonstrate the same. Note that while we use toString() method, a new String object(in Heap area) is allocated and initialized to character sequence currently represented by StringBuffer object, that means the subsequent changes to the StringBuffer object do not affect the contents of the String object.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate conversion from  // String to StringBuffer and StringBuilder.  public class Test  {      public static void main(String[] args)      {          StringBuffer sbr = new StringBuffer("Geeks");          StringBuilder sbdr = new StringBuilder("Hello");            // conversion from StringBuffer object to String          String str = sbr.toString();          System.out.println("StringBuffer object to String : ");          System.out.println(str);            // conversion from StringBuilder object to String          String str1 = sbdr.toString();          System.out.println("StringBuilder object to String : ");          System.out.println(str1);            // changing StringBuffer object sbr          // but String object(str) doesn't change          sbr.append("ForGeeks");          System.out.println(sbr);          System.out.println(str);        }  } |

Output:

StringBuffer object to String :

Geeks

StringBuilder object to String :

Hello

GeeksForGeeks

Geeks

1. **From StringBuffer to StringBuilder or vice-versa :**This conversion is tricky.There is no direct way to convert the same. In this case, We can use a String class object. We first convert StringBuffer/StringBuilder object to String using toString() method and then from String to StringBuilder/StringBuffer using constructors.Below is the java program to demonstrate the same.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java program to demonstrate conversion from  // String to StringBuffer and StringBuilder.  public class Test  {      public static void main(String[] args)      {          StringBuffer sbr = new StringBuffer("Geeks");            // conversion from StringBuffer object to StringBuilder          String str = sbr.toString();          StringBuilder sbl = new StringBuilder(str);            System.out.println(sbl);        }  } |

Output:

Geeks

**Conclusion:**

* Objects of String are immutable, and objects of StringBuffer and StringBuilder are mutable.
* StringBuffer and StringBuilder are similar, but StringBuilder is faster and preferred over StringBuffer for single threaded program. If thread safety is needed, then StringBuffer is used.

**Related Article :**  
[Reverse a string in Java (5 Different Ways)](https://www.geeksforgeeks.org/reverse-a-string-in-java-5-different-ways/)

This article is contributed by **Pranjal and Gaurav Miglani**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

## Recommended Posts:

* [Sorting collection of String and StringBuffer in Java](https://www.geeksforgeeks.org/sorting-collection-string-stringbuffer-java/)
* [equals() on String and StringBuffer objects in Java](https://www.geeksforgeeks.org/equals-string-stringbuffer-objects-java/)
* [Matcher appendReplacement(StringBuffer, String) method in Java with Examples](https://www.geeksforgeeks.org/matcher-appendreplacementstringbuffer-string-method-in-java-with-examples/)
* [Matcher appendReplacement(StringBuilder, String) method in Java with Examples](https://www.geeksforgeeks.org/matcher-appendreplacementstringbuilder-string-method-in-java-with-examples/)
* [StringBuffer insert() in Java](https://www.geeksforgeeks.org/stringbuffer-insert-java/)
* [StringBuffer class in Java](https://www.geeksforgeeks.org/stringbuffer-class-in-java/)
* [StringBuffer subSequence() in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-subsequence-in-java-with-examples/)
* [StringBuffer setLength() in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-setlength-in-java-with-examples/)
* [A Java Random and StringBuffer Puzzle](https://www.geeksforgeeks.org/java-random-stringbuffer-puzzle/)
* [StringBuffer substring() method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-substring-method-in-java-with-examples/)
* [StringBuffer indexOf() method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-indexof-method-in-java-with-examples/)
* [StringBuffer append() Method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-append-method-in-java-with-examples/)
* [StringBuffer deleteCharAt() Method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-deletecharat-method-in-java/)
* [StringBuffer appendCodePoint() Method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-appendcodepoint-method-in-java/)
* [StringBuffer delete() Method in Java with Examples](https://www.geeksforgeeks.org/stringbuffer-delete-method-in-java-with-examples/)

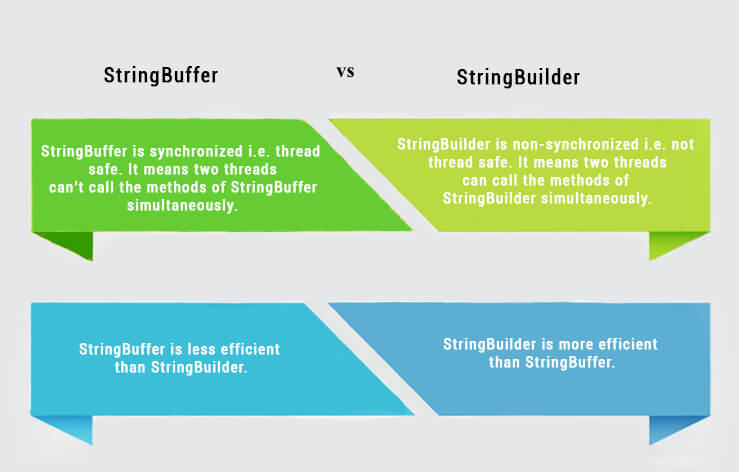
====================================================

Difference between StringBuffer and StringBuilder

Java provides three classes to represent a sequence of characters: String, StringBuffer, and StringBuilder. The String class is an immutable class whereas StringBuffer and StringBuilder classes are mutable. There are many differences between StringBuffer and StringBuilder. The StringBuilder class is introduced since JDK 1.5.

A list of differences between StringBuffer and StringBuilder are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **StringBuffer** | **StringBuilder** |
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |



## StringBuilder vs StringBuffer

Lets summarize the differences in detail:

1) **Synchronization**: StringBuffer methods are synchronized while StringBuilder methods are non-synchronized, it means that for thread-safe operations you must choose StringBuffer class instead of StringBuilder.

2) **Performance**: In a synchronized environment a single thread can perform a certain operation rather than distributing the work among multiple threads, which makes StringBuffer low performer as it is synchronized. StringBuilder performance is better than StringBuffer because it is not synchronized.

3) **Which one to use**: Operations (without considering the performance) are almost same in both the classes which means there is nothing in StringBuffer which cannot be done using StringBuilder. As discussed above the main thing which you need to consider while making a choice is **thread-safety**, if you think that the operation should be thread-safe then use StringBuffer, in all other cases StringBuilder is a better choice as it offers you the same functionality with better performance.

**Similarities**:  
Unlike String, both StringBuffer and StringBuilder are mutable (can be modified).

StringBuffer Example

1. //Java Program to demonstrate the use of StringBuffer class.
2. **public** **class** BufferTest{
3. **public** **static** **void** main(String[] args){
4. StringBuffer buffer=**new** StringBuffer("hello");
5. buffer.append("java");
6. System.out.println(buffer);
7. }
8. }

hellojava

StringBuilder Example

1. //Java Program to demonstrate the use of StringBuilder class.
2. **public** **class** BuilderTest{
3. **public** **static** **void** main(String[] args){
4. StringBuilder builder=**new** StringBuilder("hello");
5. builder.append("java");
6. System.out.println(builder);
7. }
8. }

hellojava

Performance Test of StringBuffer and StringBuilder

Let's see the code to check the performance of StringBuffer and StringBuilder classes.

1. //Java Program to demonstrate the performance of StringBuffer and StringBuilder classes.
2. **public** **class** ConcatTest{
3. **public** **static** **void** main(String[] args){
4. **long** startTime = System.currentTimeMillis();
5. StringBuffer sb = **new** StringBuffer("Java");
6. **for** (**int** i=0; i<10000; i++){
7. sb.append("Tpoint");
8. }
9. System.out.println("Time taken by StringBuffer: " + (System.currentTimeMillis() - startTime) + "ms");
10. startTime = System.currentTimeMillis();
11. StringBuilder sb2 = **new** StringBuilder("Java");
12. **for** (**int** i=0; i<10000; i++){
13. sb2.append("Tpoint");
14. }
15. System.out.println("Time taken by StringBuilder: " + (System.currentTimeMillis() - startTime) + "ms");
16. }
17. }

Time taken by StringBuffer: 16ms

Time taken by StringBuilder: 0ms

Next Topic[How to create immutable class](https://www.javatpoint.com/how-to-create-immutable-class)