1. **String vs StringBuilder vs StringBuffer in Java**

A **string** is a sequence of characters. In Java, String objects are immutable, which simply means once created, their values can not be changed. In Java, String, **StringBuilder**, and **StringBuffer** are used for handling strings. The main difference is:

* [String](https://www.geeksforgeeks.org/java/string-class-in-java/)**:** Immutable, meaning its value cannot be changed once created. It is thread-safe but less memory-efficient.
* [StringBuilder](https://www.geeksforgeeks.org/java/stringbuilder-class-in-java-with-examples/): Mutable, not thread-safe, and more memory-efficient compared to String. Best used for single-threaded operations.
* [StringBuffer](https://www.geeksforgeeks.org/java/stringbuffer-class-in-java/): Mutable and thread-safe due to synchronization, but less efficient than StringBuilder in terms of performance.

Difference Between String, StringBuilder, and StringBuffer

|  |  |  |
| --- | --- | --- |
| Feature | Action Class | JavaScriptExecutor |
| **Purpose** | Simulate real user keyboard/mouse actions | Execute JavaScript directly in browser |
| **Keyboard Support** | ✅ Yes (e.g., sendKeys(Keys.ENTER)) | ⚠️ Limited (manual event dispatching) |
| **Real User Simulation** | ✅ Yes | ❌ No |
| **DOM Manipulation** | ❌ No | ✅ Yes |
| **Use Case** | Form filling, keyboard shortcuts | Hidden elements, custom JS events |
| **Reliability** | ✅ High (if element is interactable) | ⚠️ Depends on JS and browser behavior |

Let us consider the code below with three concatenation functions with three different types of parameters, String, StringBuffer, and StringBuilder. Let us clear out the understanding between them via a single Java program below, from which we will be drawing out conclusions from the output generated, to figure out the main differences between String vs StringBuilder, vs StringBuffer in Java.

class Geeks {

// Method 1

// Concatenates to String

public static void concat1(String s1)

{

s1 = s1 + "forgeeks";

}

// Method 2

// Concatenates to StringBuilder

public static void concat2(StringBuilder s2)

{

s2.append("forgeeks");

}

// Method 3

// Concatenates to StringBuffer

public static void concat3(StringBuffer s3)

{

s3.append("forgeeks");

}

// Method 4

// Main driver method

public static void main(String[] args)

{

// Custom input string

// String 1

String s1 = "Geeks";

// Calling above defined method

concat1(s1);

// s1 is not changed

System.out.println("String: " + s1);

// String 1

StringBuilder s2 = new StringBuilder("Geeks");

// Calling above defined method

concat2(s2);

// s2 is changed

System.out.println("StringBuilder: " + s2);

// String 3

StringBuffer s3 = new StringBuffer("Geeks");

// Calling above defined method

concat3(s3);

// s3 is changed

System.out.println("StringBuffer: " + s3);

}

}

**Output**

String: Geeks

StringBuilder: Geeksforgeeks

StringBuffer: Geeksforgeeks

**Explanation:**

* **Concat1**: In this method, the string "Geeks" is passed, and we perform s1 = s1 + "forgeeks". Since String is immutable, a new string is created, and s1 in concat1() points to it. The original string in main() remains unchanged.
* **Concat2:** Here, s2.append("forgeeks") modifies the original StringBuilder object. Since StringBuilder is mutable, it updates the string directly in main() to "Geeksforgeeks".
* **Concat3**: StringBuffer and StringBuilder are similar, but StringBuffer is thread-safe due to synchronized methods, while StringBuilder is not, making it thread-unsafe.

**When to Use Which?**

Geeks now you must be wondering when to use which one, do refer below as follows:

* If a string is going to remain constant throughout the program, then use the String class object because a String object is immutable.
* If a string can change (for 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.
* If you don't want thread-safety than you can also go with StringBuilder class as it is not synchronized.

**Conversion Between String, StringBuilder, and StringBuffer**

Sometimes there is a need for converting a string object of different classes like String, StringBuffer, StringBuilder to one another. Below are some techniques to do the same. Let's cover all use cases as follows:

* From String to StringBuffer and StringBuilder
* From StringBuffer and StringBuilder to String
* From StringBuffer to StringBuilder or vice-versa

**Case 1: Convert String to StringBuffer and StringBuilder**

This one is an easy way out as we can directly pass the String class object to StringBuffer and StringBuilder class constructors. As the String class is immutable in java, so for editing a string, we can perform the same by converting it to StringBuffer or StringBuilder class objects.

API:

A screenshot of a computer program

AI-generated content may be incorrect.

Read Data from JSON file

A screen shot of a computer code

AI-generated content may be incorrect.

GET Request

A screenshot of a computer code

AI-generated content may be incorrect.

**POST Request**

1. Send payload directly to Body

A screen shot of a computer code

AI-generated content may be incorrect.

1. Reading date from JSON fine

A screenshot of a computer program

AI-generated content may be incorrect.

1. Send payload from PayLoads class

A computer screen shot of a code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Compare two JSONS

A computer screen shot of a computer code

AI-generated content may be incorrect.

A computer code on a white background

AI-generated content may be incorrect.