String vs StringBuilder vs StringBuffer in Java

Difficulty Level : Easy Last Updated : 10 Nov, 2021

A string is a sequence of characters. In java, objects of String are immutable which means a constant and cannot be changed once created. Initialize a String is one of the important pillars required as a pre-requisite with deeper understanding. For now, we will be justifying let us do consider the below code 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 differences between String vs StringBuilder vs StringBuffer in Java.

Example

```
// Java program to demonstrate difference between
// String, StringBuilder and StringBuffer

// Main class
class GFG {

    // 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
```

Output explanation:

- **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 the new string. References s1 in main() and cocat1() refer to different strings.
- **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.

- **Concat3**: StringBuilder is similar and can be compatible at all places to StringBuffer except for the key difference of thread safety. StringBuffer is thread-safe while StringBuilder does not guarantee thread safety which means synchronized methods are present in StringBuffer making control of one thread access at a time while it is not seen in StringBuilder, hence thread-unsafe.

Note: *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 types of strings in Java

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 us do cover all use cases s follows:

1. From String to StringBuffer and StringBuilder
2. From StringBuffer and StringBuilder to String
3. From StringBuffer to StringBuilder or vice-versa

Case 1: From 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.

Example

```
// Java program to demonstrate conversion from
// String to StringBuffer and StringBuilder

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Custom input string
        String str = "Geeks";

        // Converting String object to StringBuffer object
        // by
        // creating object of StringBuffer class
        StringBuffer sbr = new StringBuffer(str);

        // Reversing the string
        sbr.reverse();

        // Printing the reversed string
        System.out.println(sbr);

        // Converting String object to StringBuilder object
        StringBuilder sbl = new StringBuilder(str);

        // Adding it to string using append() method
        sbl.append("ForGeeks");

        // Print and display the above appended string
        System.out.println(sbl);
    }
}
```

Output

```
skeeG
GeeksForGeeks
```

Case 2: From StringBuffer and StringBuilder to String

This conversion can be performed 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 the character sequence currently represented by the StringBuffer object, which means the subsequent changes to the StringBuffer object do not affect the contents of the String object.

Example

```
// Java Program to Demonstrate Conversion from
// String to StringBuffer and StringBuilder

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating objects of StringBuffer class
        StringBuffer sbr = new StringBuffer("Geeks");
        StringBuilder sbdr = new StringBuilder("Hello");

        // Converting StringBuffer object to String
        // using toString() method
        String str = sbr.toString();

        // Printing the above string
        System.out.println(
            "StringBuffer object to String : ");
        System.out.println(str);

        // Converting StringBuilder object to String
        String str1 = sbdr.toString();

        // Printing the above string
        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");

        // Printing the above two strings on console
```

```
        System.out.println(sbr);
        System.out.println(str);
    }
}
```

Output

```
StringBuffer object to String :
Geeks
StringBuilder object to String :
Hello
GeeksForGeeks
Geeks
```

Case 3: 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 the StringBuffer/StringBuilder object to String using ***toString()* method** and then from String to StringBuilder/StringBuffer using constructors.

Example

```
// Java program to Demonstrate conversion from
// String to StringBuffer and StringBuilder

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating object of StringBuffer class and
        // passing our input string to it
        StringBuffer sbr = new StringBuffer("Geeks");

        // Storing value StringBuffer object in String and
        // henceforth converting StringBuffer object to
        // StringBuilder class
        String str = sbr.toString();
        StringBuilder sbl = new StringBuilder(str);
```

```
        // Printing th StringBuilder object on console
        System.out.println(sbl);
    }
}
```

Output

```
Geeks
```

From the above three use-cases we can conclude out below pointers:

- 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 the single-threaded program. If thread safety is needed, then StringBuffer is used.

Related Article: 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 review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.