StringBuffer vs StringBuilder in Java

Last Updated : 15 Oct, 2024

Java provides three classes to represent the sequence of characters i.e. String, StringBuffer, and StringBuilder. A string is immutable in Java whereas, both **StringBuffer** and **StringBuilder** are mutable. This means they allow modifications to the character sequence without creating new objects.

The main difference between `StringBuffer` and `StringBuilder` is that `StringBuffer` is **thread-safe** due to synchronization and this is slower. On the other hand, StringBuilder is faster but not thread-safe and is introduced in JDK 1.5. If we do not need thread safety, `StringBuilder` is the better choice due to its speed. We typically do not need thread safety when solving interview questions, competitive coding questions, and single-threaded applications. Use `StringBuffer` only when dealing with multiple threads modifying the same string.

Now let's deep dive into **StringBuffer vs StringBuilder in Java**.

StringBuilder vs StringBuffer in Java

Below is the key differences table of StringBuffer and StringBuilder.

StringBuffer	StringBuilder
StringBuffer is present since early versions of Java.	StringBuilder was introduced in Java 5 (JDK 1.5).
StringBuffer is synchronized. This means that multiple threads cannot	StringBuilder is asynchronized. This means that multiple threads can

StringBuffer	StringBuilder
call the methods of StringBuffer simultaneously.	call the methods of StringBuilder simultaneously.
Due to synchronization, StringBuffer is called a thread safe class.	Due to its asynchronous nature, StringBuilder is not a thread safe class.
Due to synchronization, StringBuffer is lot slower than StringBuilder.	Since there is no preliminary check for multiple threads, StringBuilder is a lot faster than StringBuffer.
Use in multi-threaded environments.	Use in single-threaded or non-concurrent environments.

Table of Content

- StringBuilder vs StringBuffer in Java
- StringBuffer Class
- StringBuilder Class
- Conversion from StringBuffer to StringBuilder
- Conversion from StringBuilder to StringBuffer

StringBuffer Class

`StringBuffer` is a class that allows us to create and modify a flexible, expandable sequence of characters. It is **thread-safe.** Thismeans that it can be safely used when multiple threads are accessing or modifying the string concurrently. But due to synchronization, it is slower than `StringBuilder`.

```
StringBuffer str = new StringBuffer();
```

Example:

Below is an example which implements the StringBuffer class.

```
// Java program to demonstrate
// the StringBuffer class
```

```
public class GFG {

    // Driver code
    public static void main(String args[])
    {

        // Creating a new StringBuffer
        StringBuffer str
            = new StringBuffer("Hello");

        str.append(" World!");

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

Output

```
Hello World!
```

StringBuilder Class

`StringBuilder` is similar to `StringBuffer`, but it is **not synchronized.** Thismeans it is not thread-safe. But it is much faster than `StringBuffer` and is suitable for use in scenarios where thread safety is not required (e.g., competitive coding, interview questions).

```
StringBuilder str = new StringBuilder();
```

Example:

Below is an example which implements the StringBuilder class.

```
// Java program to demonstrate
// the StringBuilder class

public class GFG {

    // Driver code
    public static void main(String args[])
    {

        // Creating a new StringBuilder
        StringBuilder str
            = new StringBuilder("Hello");

        str.append(" World!");

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

```
    }
}
```

Output

```
Hello World!
```

Conversion from StringBuffer to StringBuilder

The StringBuffer cannot be directly converted to StringBuilder. We first need to convert the StringBuffer to a String object by using the inbuilt method **toString()** method. After converting it to a string object, we can simply create a StringBuilder using the constructor of the class.

For example:

```
// Java program to demonstrate
// the conversion between the
// StringBuffer and StringBuilder
// class

public class GFG {

    // Driver code
    public static void main(String args[])
    {
        StringBuffer sbr
            = new StringBuffer("Geeks");

        // Conversion from StringBuffer
        // object to the String object
        String str = sbr.toString();

        // Creating a new StringBuilder
        // using the constructor
        StringBuilder sbl
            = new StringBuilder(str);

        System.out.println(sbl);
    }
}
```

Output

```
Geeks
```

Conversion from StringBuilder to StringBuffer

Similar to the above conversion, the StringBuilder cannot be converted to the StringBuffer directly. We first need to convert the StringBuilder to the String object by using the inbuilt method **toString().**

Now, we can create a StringBuilder using the constructor.

For example:

```
// Java program to demonstrate
// the conversion between the
// StringBuilder and StringBuffer
// class

public class GFG {

    // Driver code
    public static void main(String args[])
    {
        StringBuilder sbr
            = new StringBuilder("Geeks");

        // Conversion from StringBuilder
        // object to the String object
        String str = sbr.toString();

        // Creating a new StringBuffer
        // using the constructor
        StringBuffer sbl
            = new StringBuffer(str);

        System.out.println(sbl);
    }
}
```

Output

```
Geeks
```

Conclusion

When deciding between `StringBuffer` and `StringBuilder`, the key factor is **thread safety**. If thread safety is not required, prefer `StringBuilder` for its speed and performance. For multi-threaded applications where the same String is modified by multiple threads, `StringBuffer` is a safer option but it is slower due to synchronization.