# StringBuffer in Java

**1. What is StringBuffer?**

* In Java, String objects are **immutable** (once created, cannot be changed).
* To solve this, Java introduced **StringBuffer** (and later StringBuilder) which are **mutable**.
* **StringBuffer** is a **thread-safe, mutable sequence of characters**.
* Belongs to the package: java.lang.

👉 In short:

* String → Immutable
* StringBuffer → Mutable + Thread-safe

**2. Why StringBuffer? (Purpose)**

* When you are performing many modifications (append, insert, delete, replace) on a string, using String will create new objects every time → inefficient.
* StringBuffer avoids creating new objects by modifying the **same object in memory**.
* Useful when working in **multi-threaded** environments because it is **synchronized** (thread-safe).

**3. When to use StringBuffer?**

* Use StringBuffer when:
  + You need to **modify strings frequently** (append, reverse, replace, delete, insert).
  + Your code is **multi-threaded** (multiple threads accessing same string).
* If thread-safety is not needed, prefer StringBuilder (faster).

**4. Where is StringBuffer used?**

* Real-time use cases:
  1. **Banking / Transaction systems**  
     Maintaining logs or generating dynamic statements where many threads access the same string buffer.
  2. **Chat applications**  
     Building a chat message dynamically by multiple users at the same time.
  3. **Compiler design / text processing**  
     When large text or code needs to be edited continuously.
  4. **Reports generation**  
     Concatenating strings dynamically with safe multi-thread access.

**5. How StringBuffer works?**

* Internally, it uses a **character array** (like char[] value) to hold the sequence.
* When you modify (append, insert, etc.), it updates the array **without creating new object**.
* Capacity increases automatically if needed (default = 16 chars).

**6. Common Methods in StringBuffer**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | Method | Description | Example | | append(String s) | Adds text at end | "Hello".append("World") → HelloWorld | | insert(int offset, String s) | Inserts text at position | "Hello".insert(2, "Java") → HeJavallo | | replace(int start, int end, String s) | Replaces text | "Hello".replace(1,3,"Java") → HJavalo | | delete(int start, int end) | Deletes chars | "Hello".delete(1,3) → Ho | | reverse() | Reverses string | "Hello".reverse() → olleH | | capacity() | Returns buffer size | default 16 + string length | | length() | Returns number of characters | "Hello".length() → 5 | |

**7. Code Examples:**

public class SBuffer {

    public static void main(String[] args){

        // Creating StringBuffer

        StringBuffer s = new StringBuffer();

        //append() Method

        StringBuffer sb = new StringBuffer("Hello ");

        sb.append("Java"); // now original string is changed

        System.out.println(sb);

        // Adding elements in StringBuffer

        s.append("Hello");

        s.append(" ");

        s.append("world");

        // String with the StringBuffer value

        System.out.println("String Buffer-------"+s);

        String str = s.toString();

        System.out.println("String Buffer to string-------"+str);

    // 1. Using default constructor

    StringBuffer sb1 = new StringBuffer();

        sb1.append("Hello");

        System.out.println("Default Constructor: " + sb1);

    // 2. Using constructor with specified capacity

    StringBuffer sb2 = new StringBuffer(50);

        sb2.append("Java Programming");

        System.out.println("With Capacity 50: " + sb2);

    // 3. Using constructor with String

    StringBuffer sb3 = new StringBuffer("Welcome");

        sb3.append(" to Java");

        System.out.println("With String: " + sb3);

        //insert() Method

        StringBuffer sb4 = new StringBuffer("Hello ");

        sb4.insert(1, "Java");

        // Now original string is changed

        System.out.println(sb4);

        //replace() Method

        StringBuffer sb5 = new StringBuffer("Hello");

        sb5.replace(1, 3, "Java");

        System.out.println(sb5);

        //delete() Method

        StringBuffer sb6 = new StringBuffer("Hello");

        sb6.delete(1, 3);

        System.out.println(sb6);

        // reverse() Method

        StringBuffer sb7 = new StringBuffer("Hello");

        sb7.reverse();

        System.out.println(sb7);

// capacity() Method

        StringBuffer sb8 = new StringBuffer();

        // default 16

        System.out.println(sb8.capacity());

        sb8.append("Hello");

        // now 16

        System.out.println(sb8.capacity());

        sb8.append("java is my favourite language");

        // (oldcapacity\*2)+2

        System.out.println(sb8.capacity());

        // length()

        // Creating and storing string by creating object of StringBuffer

        StringBuffer sb9 = new StringBuffer("Astinil Technologies");

        // Getting the length of the string

        int p = sb9.length();

        // Getting the capacity of the string

        System.out.println("Length of string Astinil Technologies =" + p);

}

}

**Output:**

Hello Java

String Buffer-------Hello world

String Buffer to string-------Hello world

Default Constructor: Hello

With Capacity 50: Java Programming

With String: Welcome to Java

HJavaello

HJavalo

Hlo

olleH

16

16

34

Length of string Astinil Technologies =20

**8.In Real-Time Scenarios:**

Suppose you are building a **Banking Application** where multiple users (threads) perform transactions like **Deposit, Withdraw, Transfer, Balance Enquiry** at the same time.

You need to **log all transactions** safely in one place.  
If you use String → every change will create a new object (memory waste, slow).  
If you use StringBuilder → it is not synchronized (two users updating at same time may corrupt data).  
👉 Best choice here = **StringBuffer (Thread-safe)**.

9.**Block Diagram – How StringBuffer Works:**

StringBuffer

char[] value (array)

int count (length)

synchronized methods

Updates same character array

append() insert() replace() delete() reverse()

**10.Difference Between String, StringBuffer:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | | **Feature** | **String (Immutable)** | **StringBuffer (Mutable + Thread-safe)** | | --- | --- | --- | | Mutability | ❌ No | ✅ Yes | | Performance | Slow (creates new) | Moderate (synchronized = slower) | | Thread-safe | N/A | ✅ Yes | | Best Use | Fixed data | Multi-threaded apps |  | | |

**Summary**:

* StringBuffer is a **mutable, thread-safe string class**.
* Use it in **multi-threaded** scenarios where frequent modifications are required.
* Provides methods like append, insert, replace, delete, and reverse.
* Avoids overhead of creating new objects like String.