# StringBuffer vs StringBuilder in Java 🚀

---

## Why Do We Need Them?

- **Strings in Java are immutable**, meaning any modification creates a new object.
- **StringBuffer** and **StringBuilder** help us modify strings efficiently **without creating new objects**.
- **Key difference**: StringBuffer is thread-safe, while StringBuilder is faster but not thread-safe.

## How They Work (With Diagram)

Both use a **char array** to store data and expand dynamically.

### Internal Memory Structure:

```
Initial Capacity: |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_| (16 empty slots)
After appending "Hello": |H|e|l|l|o|_|_|_|_|_|_|_|_|_|_|_| (capacity remains 16)
After exceeding capacity: |H|e|l|l|o|...new_data...| (new capacity: 34)
```

- Both StringBuffer and StringBuilder **store characters internally in a mutable char array**.

- The **default capacity is 16**, and when exceeded, it **doubles the size + 2** to optimize performance.

---

## How Java Stores StringBuffer & StringBuilder Internally 🧠

StringBuffer sb = new StringBuffer("Hello");

Internally:

```
char[] value = {'H', 'e', 'l', 'l', 'o', '_', '_', '_', '_',
'_', '_', '_', '_', '_', '_', '_'};
```

- Unlike String (which is **final** and stored in the **String Pool**), these objects exist in the **Heap Memory**.

## Thread-Safety and Performance

### StringBuffer (Thread-Safe ✅)

- **Uses Synchronization**: Only one thread can modify it at a time.
- **Best for Multi-Threaded Environments**.
- **Slightly Slower** due to thread safety.

### StringBuilder (Not Thread-Safe ❌)

- **No Synchronization**: Multiple threads can modify it simultaneously.
- **Faster** in single-threaded applications.
- **Risk of Data Corruption** in multi-threaded environments.

---

## Visual Representation:

StringBuffer (Thread-Safe):

[Thread 1] → append("A") → No Corruption

[Thread 2] → append("B") → No Corruption

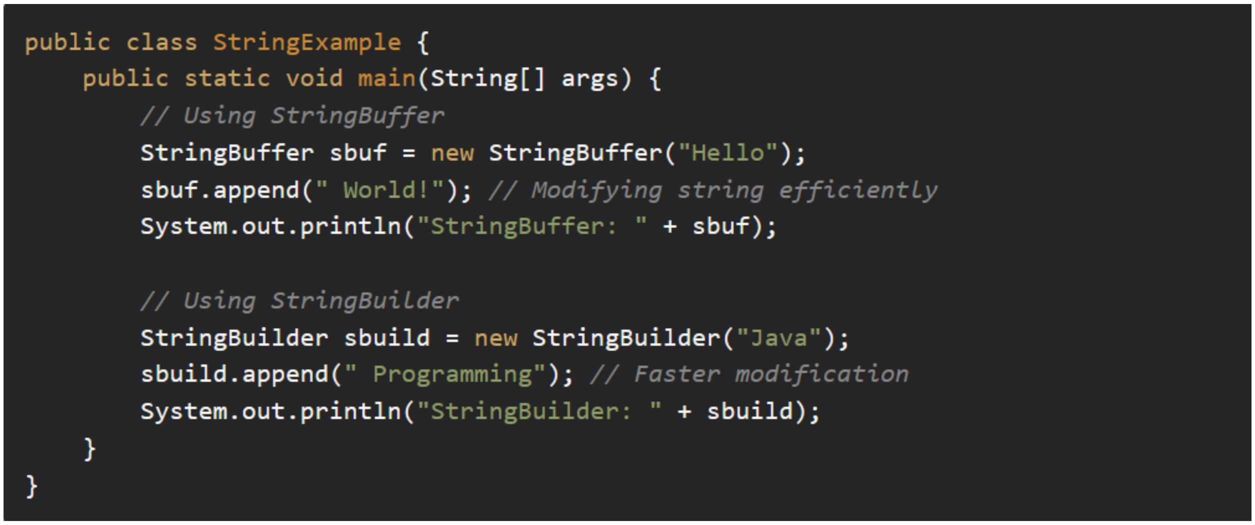
StringBuilder (Not Thread-Safe):

[Thread 1] → append("A")

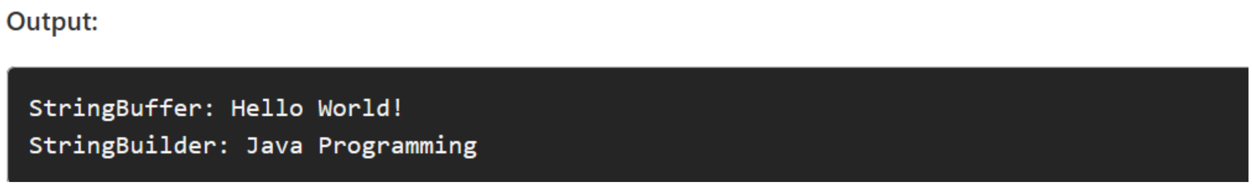
[Thread 2] → append("B") → May Corrupt Data

## Code Examples

### 1. Basic Usage



```java
public class StringExample {
    public static void main(String[] args) {
        // Using StringBuffer
        StringBuffer sbuf = new StringBuffer("Hello");
        sbuf.append(" World!"); // Modifying string efficiently
        System.out.println("StringBuffer: " + sbuf);

        // Using StringBuilder
        StringBuilder sbuild = new StringBuilder("Java");
        sbuild.append(" Programming"); // Faster modification
        System.out.println("StringBuilder: " + sbuild);
    }
}
```

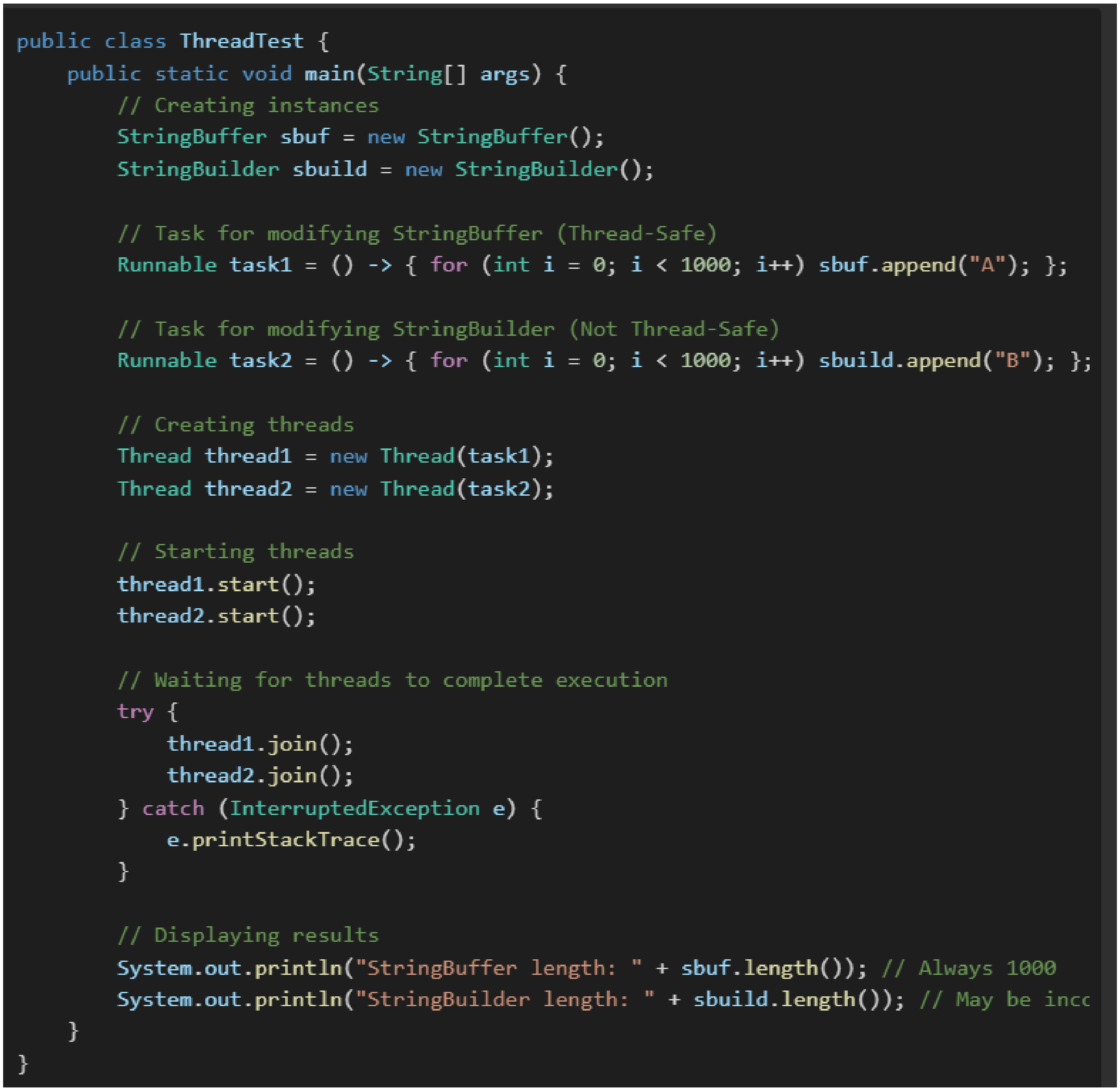


**Output:**

```
StringBuffer: Hello World!
StringBuilder: Java Programming
```

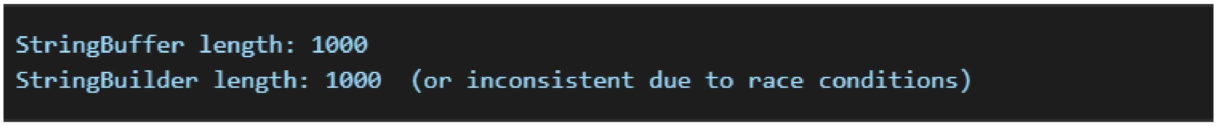
---

## 2. Multi-Threading Test



```
public class ThreadTest {
    public static void main(String[] args) {
        // Creating instances
        StringBuffer sbuf = new StringBuffer();
        StringBuilder sbuild = new StringBuilder();

        // Task for modifying StringBuffer (Thread-Safe)
        Runnable task1 = () -> { for (int i = 0; i < 1000; i++) sbuf.append("A"); };

        // Task for modifying StringBuilder (Not Thread-Safe)
        Runnable task2 = () -> { for (int i = 0; i < 1000; i++) sbuild.append("B"); };

        // Creating threads
        Thread thread1 = new Thread(task1);
        Thread thread2 = new Thread(task2);

        // Starting threads
        thread1.start();
        thread2.start();

        // Waiting for threads to complete execution
        try {
            thread1.join();
            thread2.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Displaying results
        System.out.println("StringBuffer length: " + sbuf.length()); // Always 1000
        System.out.println("StringBuilder length: " + sbuild.length()); // May be incc
    }
}
```

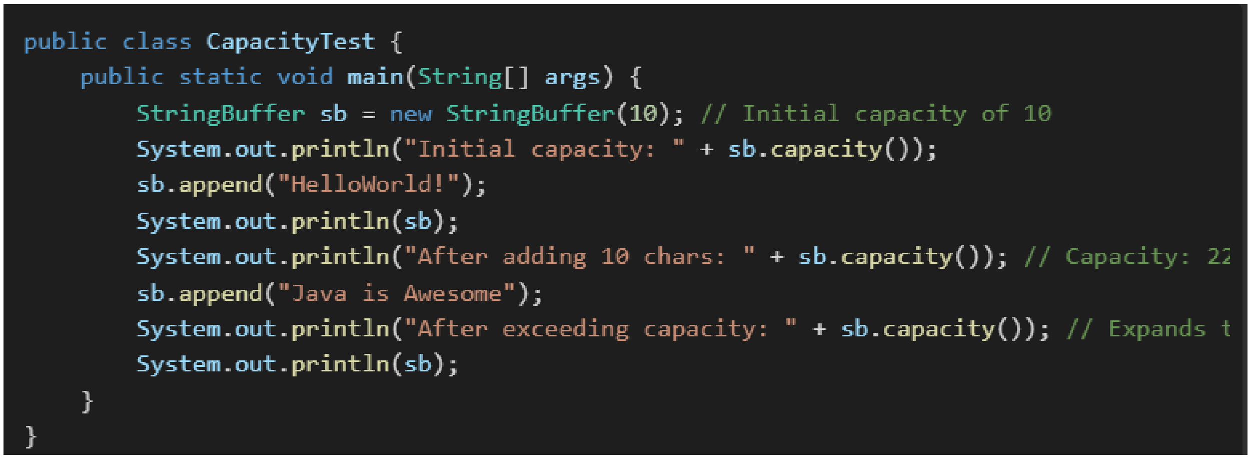
### Output:



```
StringBuffer length: 1000
StringBuilder length: 1000  (or inconsistent due to race conditions)
```

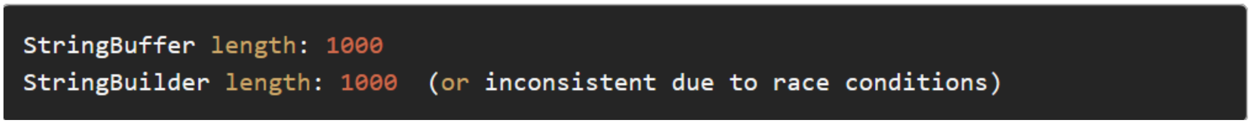
---

## 3. Expanding Capacity Manually



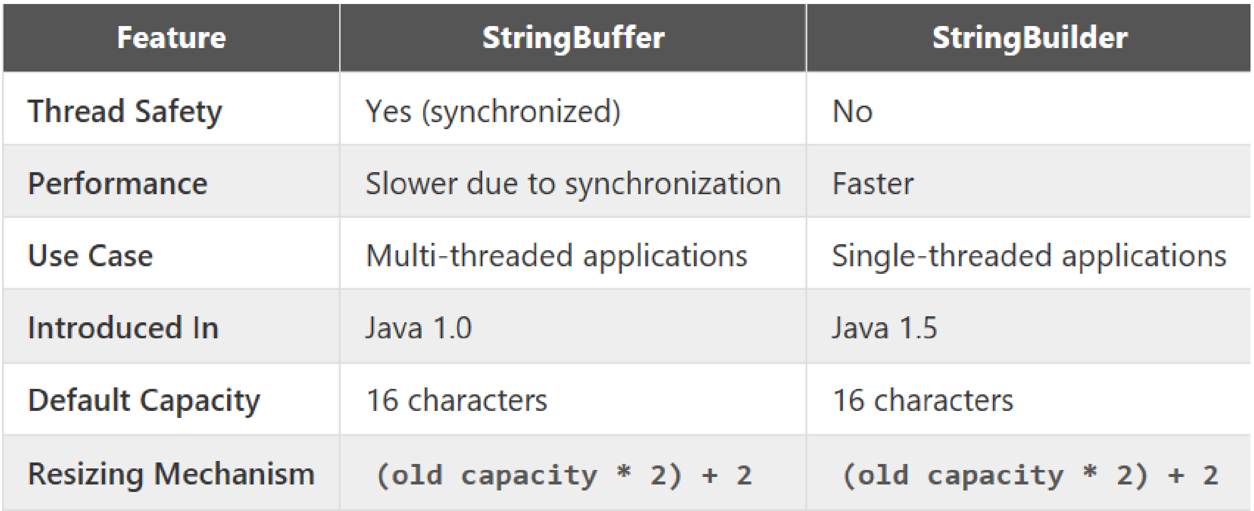
```
public class CapacityTest {
    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer(10); // Initial capacity of 10
        System.out.println("Initial capacity: " + sb.capacity());
        sb.append("HelloWorld!");
        System.out.println(sb);
        System.out.println("After adding 10 chars: " + sb.capacity()); // Capacity: 22
        sb.append("Java is Awesome");
        System.out.println("After exceeding capacity: " + sb.capacity()); // Expands t
        System.out.println(sb);
    }
}
```

**Output:**



```
StringBuffer length: 1000
StringBuilder length: 1000  (or inconsistent due to race conditions)
```

## Differences Between StringBuffer and StringBuilder



| Feature | StringBuffer | StringBuilder |
|---|---|---|
| Thread Safety | Yes (synchronized) | No |
| Performance | Slower due to synchronization | Faster |
| Use Case | Multi-threaded applications | Single-threaded applications |
| Introduced In | Java 1.0 | Java 1.5 |
| Default Capacity | 16 characters | 16 characters |
| Resizing Mechanism | `(old capacity * 2) + 2` | `(old capacity * 2) + 2` |

## Advanced Use Cases 💡

- **Buffering Large Data**: When dealing with logs, JSON/XML data, and network streams, `StringBuffer` is preferred for stability.
- **Dynamic UI Updates**: In Swing/JavaFX apps, `StringBuilder` is used for rapid changes.
- **Cryptographic Hashing**: Since cryptographic operations require thread safety, `StringBuffer` is used with security algorithms.
- **Text Processing & Parsers**: `StringBuilder` is used for parsing and constructing responses in web applications.

## Interview Questions 📝

1. **What is the key difference between String, StringBuffer, and StringBuilder?**
   - `String` is immutable, while `StringBuffer` and `StringBuilder` are mutable.
   - `StringBuffer` is synchronized (thread-safe), while `StringBuilder` is not.
2. **Why is StringBuffer thread-safe but StringBuilder is not?**
   - `StringBuffer` methods are synchronized, ensuring only one thread accesses it at a time.
   - `StringBuilder` does not use synchronization, making it faster but unsafe for multi-threading.
3. **What is the default capacity of StringBuffer/StringBuilder? How does it grow?**
   - Default capacity is **16 characters**.
   - When exceeded, capacity grows by `(old capacity * 2) + 2`.
4. **When should you use StringBuffer over StringBuilder?**
   - Use `StringBuffer` in multi-threaded environments where multiple threads modify the same object.
   - Use `StringBuilder` for better performance in single-threaded applications.
5. **Can we make StringBuilder thread-safe? How?**
   - Yes, by **explicitly synchronizing** critical sections using `synchronized` blocks or using `Collections.synchronizedList()`.

## Summary 📌

☑ Use StringBuilder for fast performance in single-threaded applications.

☑ Use StringBuffer if multiple threads modify the same object to prevent data corruption. ☑ Set an initial capacity to avoid frequent resizing for better efficiency.

☑ Understand internal storage to optimize memory usage.