In Java, **String**, **StringBuffer**, and **StringBuilder** are classes used to handle and manipulate sequences of characters. They serve different purposes based on mutability, thread-safety, and performance requirements.

**1. String**

* **Definition**: A String is an immutable sequence of characters.
* **Mutability**: Immutable (cannot be changed after creation).
* **Thread-Safety**: Thread-safe due to immutability.
* **Performance**: Slower for modifications because every operation creates a new String object.
* **Use Case**: Use String when the content does not change frequently or when thread-safety is required without extra synchronization.

**Example:**

public class StringExample {

public static void main(String[] args) {

String s1 = "Hello";

String s2 = s1.concat(" World");

System.out.println(s1); // Output: Hello (unchanged)

System.out.println(s2); // Output: Hello World

}

}

**2. StringBuffer**

* **Definition**: A StringBuffer is a mutable sequence of characters.
* **Mutability**: Mutable (can be modified after creation).
* **Thread-Safety**: Thread-safe because its methods are synchronized.
* **Performance**: Slower than StringBuilder because of synchronization overhead.
* **Use Case**: Use StringBuffer when multiple threads may access and modify the string.

**Example:**

public class StringBufferExample {

public static void main(String[] args) {

StringBuffer sb = new StringBuffer("Hello");

sb.append(" World");

System.out.println(sb); // Output: Hello World

}

}

**3. StringBuilder**

* **Definition**: A StringBuilder is also a mutable sequence of characters.
* **Mutability**: Mutable (can be modified after creation).
* **Thread-Safety**: Not thread-safe because its methods are not synchronized.
* **Performance**: Faster than StringBuffer because there is no synchronization overhead.
* **Use Case**: Use StringBuilder for single-threaded applications or when thread-safety is not a concern.

**Example:**

public class StringBuilderExample {

public static void main(String[] args) {

StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

System.out.println(sb); // Output: Hello World

}

}

**Comparison Table**

| **Feature** | **String** | **StringBuffer** | **StringBuilder** |
| --- | --- | --- | --- |
| **Mutability** | Immutable | Mutable | Mutable |
| **Thread-Safety** | Thread-safe | Thread-safe | Not thread-safe |
| **Performance** | Slower (new object for every modification) | Moderate (synchronized methods) | Faster (no synchronization) |
| **Usage** | For immutable strings or frequent read-only operations | For mutable strings in multi-threaded environments | For mutable strings in single-threaded environments |

**Choosing Between Them**

1. **Use String** when:
   * The value will not change.
   * You prioritize simplicity and readability.
2. **Use StringBuffer** when:
   * You are working in a multi-threaded environment.
   * You need mutable strings and thread safety is required.
3. **Use StringBuilder** when:
   * You are working in a single-threaded environment.
   * You need mutable strings and prioritize performance.

**Practical Example Comparing All Three**

public class StringComparison {

public static void main(String[] args) {

// String Example

String s = "Hello";

s.concat(" World"); // Creates a new String object

System.out.println("String: " + s); // Output: Hello

// StringBuffer Example

StringBuffer sb = new StringBuffer("Hello");

sb.append(" World"); // Modifies the original object

System.out.println("StringBuffer: " + sb); // Output: Hello World

// StringBuilder Example

StringBuilder sbuilder = new StringBuilder("Hello");

sbuilder.append(" World"); // Modifies the original object

System.out.println("StringBuilder: " + sbuilder); // Output: Hello World

}

}

**Performance Comparison Example**

public class PerformanceTest {

public static void main(String[] args) {

long startTime, endTime;

// Testing String

startTime = System.currentTimeMillis();

String s = "Hello";

for (int i = 0; i < 10000; i++) {

s = s + " World";

}

endTime = System.currentTimeMillis();

System.out.println("String Time: " + (endTime - startTime) + "ms");

// Testing StringBuffer

startTime = System.currentTimeMillis();

StringBuffer sb = new StringBuffer("Hello");

for (int i = 0; i < 10000; i++) {

sb.append(" World");

}

endTime = System.currentTimeMillis();

System.out.println("StringBuffer Time: " + (endTime - startTime) + "ms");

// Testing StringBuilder

startTime = System.currentTimeMillis();

StringBuilder sbuilder = new StringBuilder("Hello");

for (int i = 0; i < 10000; i++) {

sbuilder.append(" World");

}

endTime = System.currentTimeMillis();

System.out.println("StringBuilder Time: " + (endTime - startTime) + "ms");

}

}

**Expected Output**

* String Time: Much higher because of immutable nature (new object created every time).
* StringBuffer Time: Moderate due to synchronization overhead.
* StringBuilder Time: Fastest due to lack of synchronization.