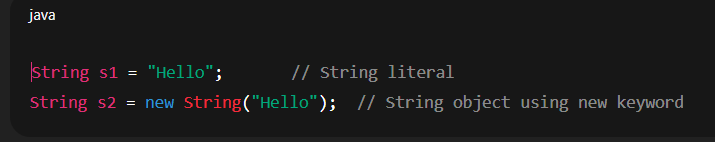
## 🧠 ****1. What is String in Java****

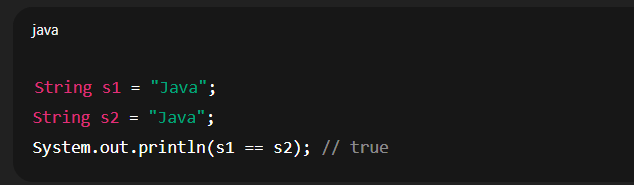
* String is a **class** in java.lang package.
* It represents a **sequence of characters**.
* Strings are **immutable** (cannot be changed once created).
* It is **final**, so it cannot be subclassed.



## 🧩 ****2. Memory Management (String Constant Pool)****

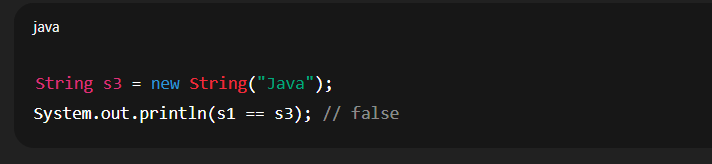
* When you create a string using **string literal**, it is stored in the **String Constant Pool (SCP)** inside the heap.

Example:



✅ Both s1 and s2 refer to the same object from the **String Pool**.

If you use **new keyword**, a new object is created in the **heap memory**, not in the pool:



* Diagram of representing **Heap** and **SCP** area of memory

## 

· s1 → refers to SCP object

· s2 → it checks first same object is present in SCP ,if yes then refers to existing object

· s3 → refers to Heap object

· s4 → refers to Heap object,if already present in heap or not

✅ **Summary:**

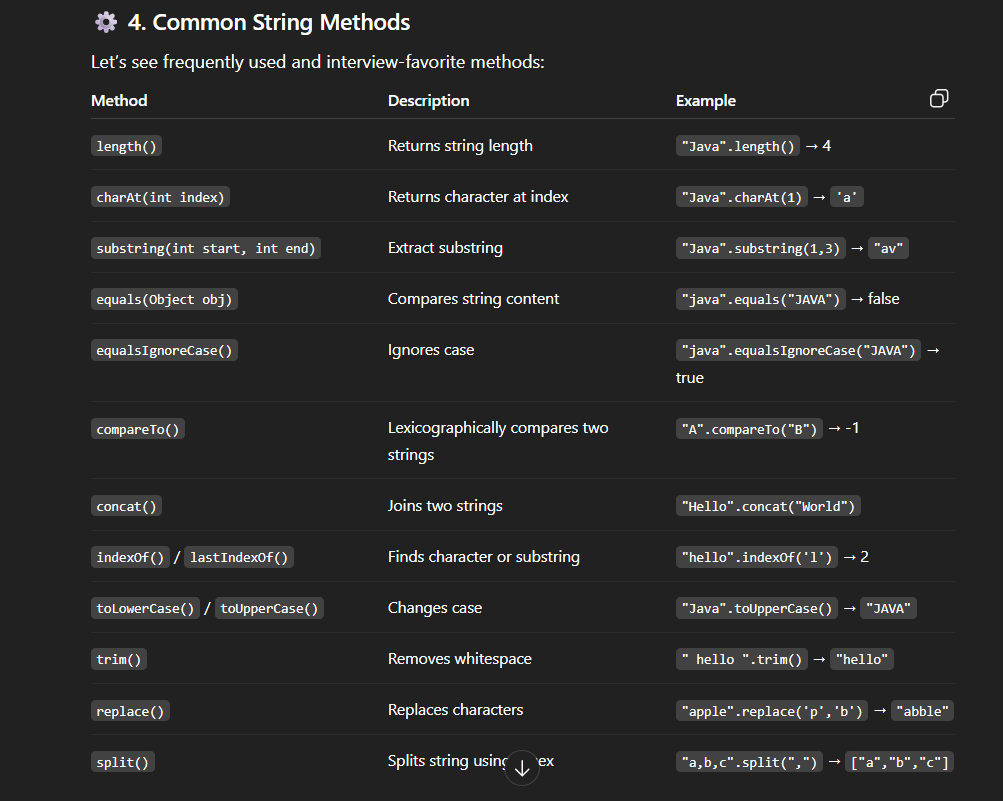
| **Code** | **SCP Objects** | **Heap Objects** | **Total** |
| --- | --- | --- | --- |
| String s1="Java"; | 1 | 0 | 1 |
| String s2="Java"; | 0 | 0 | 0 (it refers to s1 object which is already present) |

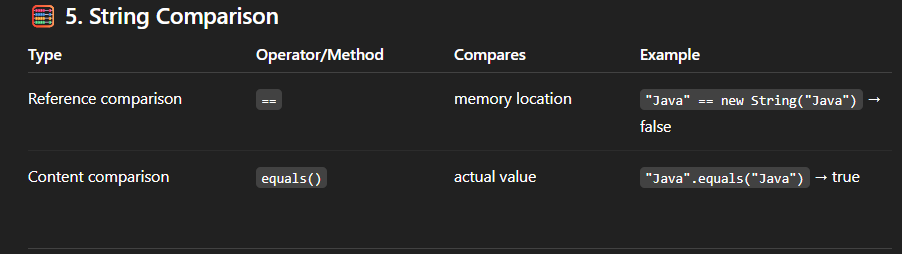
|  |  |  |  |
| --- | --- | --- | --- |
| Code | SCP | Heap | Total |
| String s3 = new String("Java"); | 0 | 1 | 1(always create a new obj.in heap) |
| String s4 = new String("Java"); | 0 | 1 | 1 |

## ****3. Why String is Immutable****

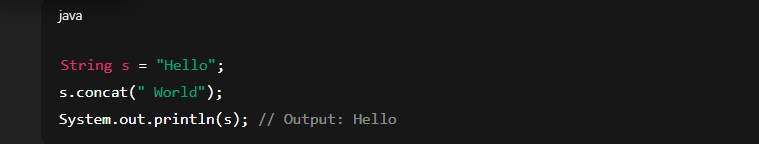
### Reasons:

1. **Security** — Strings are used in sensitive areas like database URLs, usernames, passwords. Immutability prevents tampering.
2. **Thread safety** — Immutable objects can be shared safely among threads.
3. **Caching** — The string pool benefits from immutability (same string reused).
4. **Hashcode caching** — String stores its hashcode after first calculation, improving performance in hash-based collections (like HashMap).



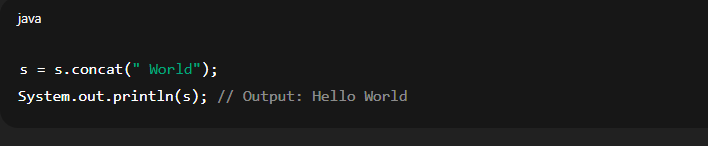


## ****6. String Immutability Example****



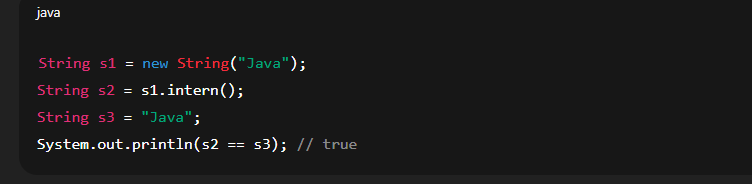
➡ concat() creates a **new string**, but since not stored in any variable, it’s lost.

If we store:



## ⚡ ****7. String Interning****

When you use intern() method, it ensures that the string is added to the pool if not already present.

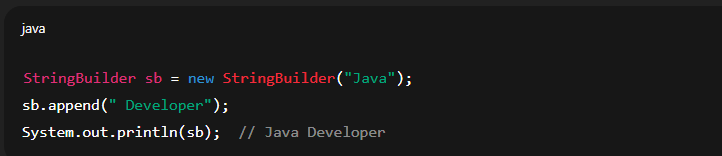


# 

# 2.****What is**** StringBuilder****?****

### 💡 Definition:

* StringBuilder is a **mutable class** introduced in **Java 5** (JDK 1.5).  
  It allows you to **modify strings (append, insert, delete, reverse)** without creating new objects every time.

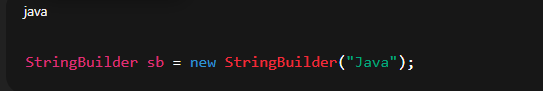


Here, the **same object** is updated in memory — not a new one.



## 🧠 Internal Working:

When you create:



It creates a **char[] array** internally with default capacity **16 + string length**.

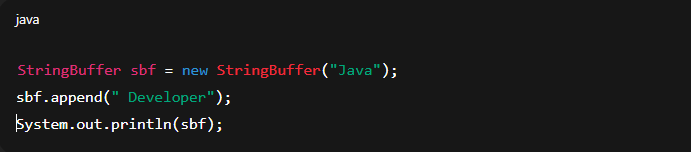
Example:  
"Java" → length = 4  
So total capacity = **20** (16 + 4).

If you exceed capacity, it automatically increases to:  
(oldCapacity \* 2) + 2

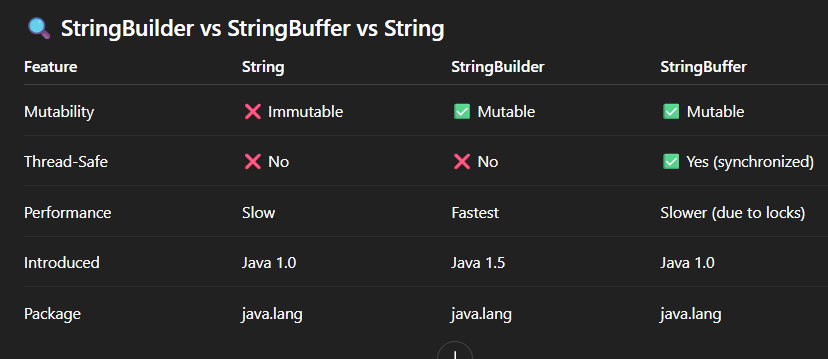
# 🧱 ****3️⃣ What is**** StringBuffer****?****

### 💡 Definition:

StringBuffer is **similar to** StringBuilder — also **mutable** —  
but it is **thread-safe** (synchronized).

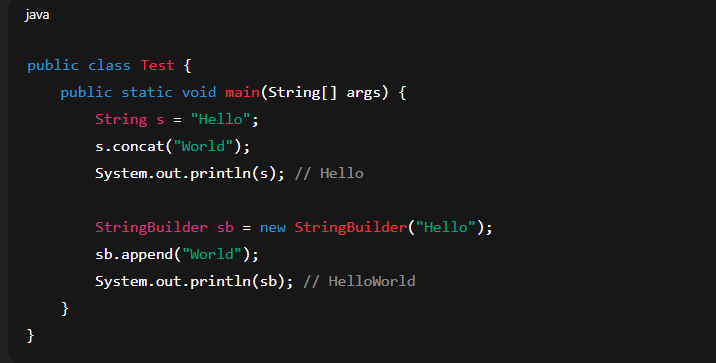


✅ Use StringBuffer when multiple threads modify the same string.  
✅ Use StringBuilder in single-threaded environments (faster).

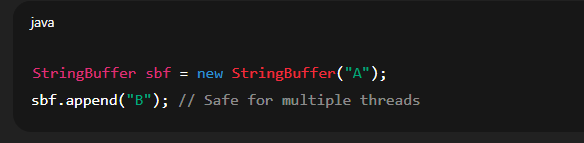


# ⚡ ****4️⃣ Interview-Level Examples****

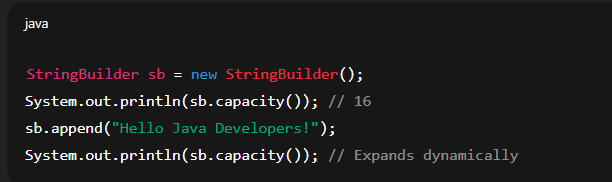
### 🧮 Example 1 — Difference in Behavior:



### 🧩 Example 2 — Thread Safety:



### 🧩 Example 3 — Capacity Behavior:



# 🧠 ****5️⃣ Interview Tips****

✅ Always mention:

* String → Immutable, stored in SCP.
* StringBuilder → Mutable, not thread-safe, fast.
* StringBuffer → Mutable, thread-safe, slower.

✅ If asked: “Which one should you use?”

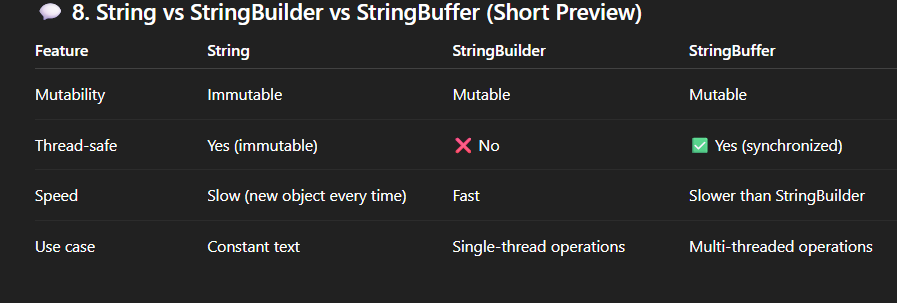
* For **read-only text** → String
* For **frequent modifications in single-thread** → StringBuilder
* For **multi-threaded modification** → StringBuffer

✅ Common tricky question:

* Why is String immutable in Java?

**Answer:**

* Security (used in class loading, URLs, etc.)
* Thread safety
* Cache & performance optimization (hashcode reuse)



## ****Common Interview Questions****

1. Why is String immutable in Java?
2. What is String Constant Pool?
3. Difference between == and equals() in String?
4. What is intern() method?
5. How is String stored in memory?
6. Difference between String, StringBuilder, and StringBuffer?
7. Why is String final?
8. How to reverse a String without using reverse() method?
9. What happens when you do s1 + s2?
10. Explain String.intern() with example.