🧵 What is a String in Java?

In Java, a **String** is an object that represents a sequence of characters. It is part of the `java.lang` package and is **immutable**, meaning once a String object is created, its value cannot be changed.

🇮🇳 Real-Life Analogy:

Think of a **nameplate** outside a house:

- The name is fixed (immutable).
- If you want to change it, you replace the whole plate (create a new string).
- You can read letters from it, count them, or compare it with others.

📦 Heap Memory & Non-Primitive Type Explained

The `String` is a class in Java, which means it is a non-primitive data type. Since it is a non-primitive data type, it resides in the heap memory.

🧠 Java String Pool Memory Allocation

If you declare any `String` without using the `new` keyword, that string will be stored in the **String Pool** (also known as the **intern pool**), which resides inside the **heap memory**.

🧠 Java String Pool vs `new` Keyword

Consider the **String Pool** in Java as a **special area inside the heap memory**, just like **7 RCR** (the Prime Minister's residence) is a special place within **Delhi**. Everyone lives in Delhi, but the PM lives at 7 RCR — a reserved and unique location. Similarly, the String Pool is a reserved part of the heap — **dedicated for storing unique String literals**.

🔁 String Literals and Reuse (Without `new` Keyword)

When you create a String like this:

```
String s1 = "hello";
String s2 = "hello";
```

Java checks the **String Pool**. If `"hello"` already exists, **no new object is created**. Instead:

- `s1` points to memory address `2K`
- `s2` also points to the **same memory address** `2K`

✅ **Result:** One `"hello"` string in memory, **shared by both variables**. This avoids duplication — a clever memory optimization.

🆕 Creating String Objects with `new`

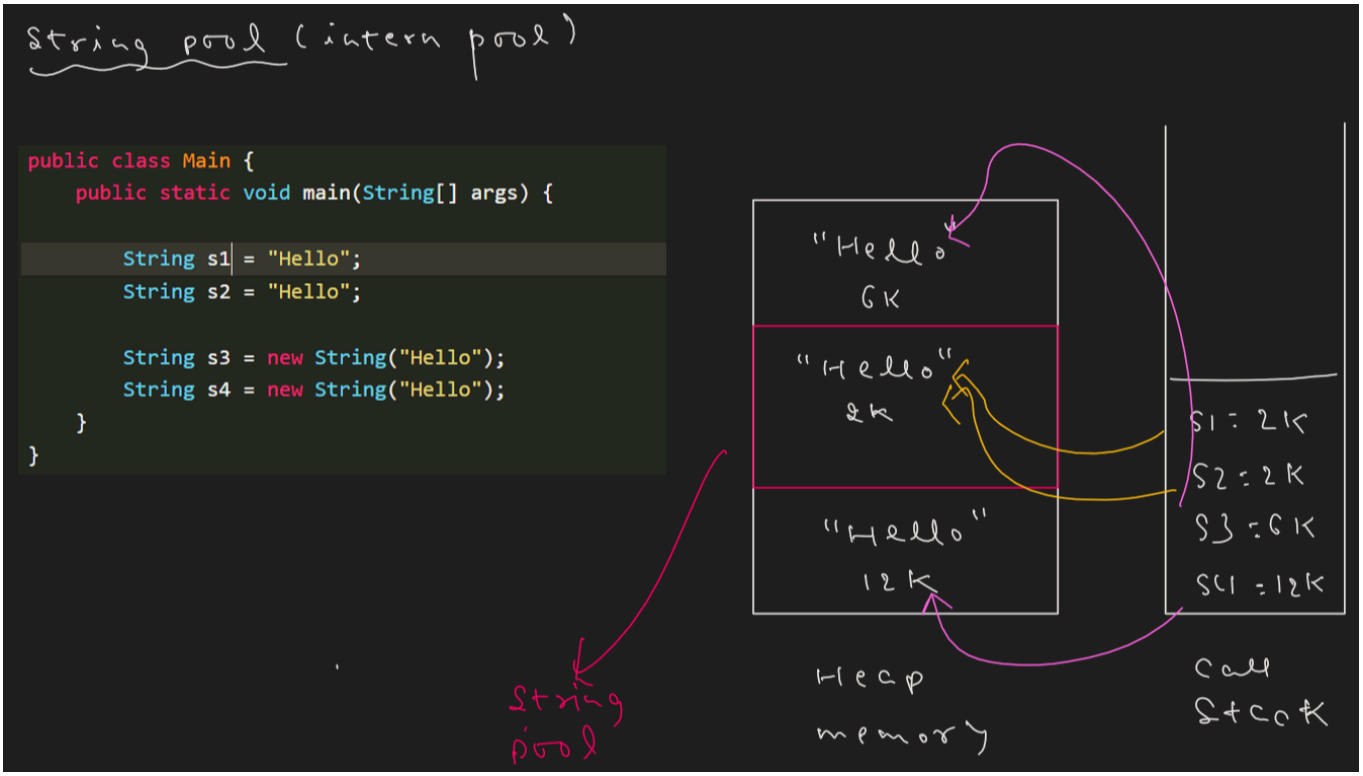
When you explicitly use the `new` keyword:

```
String s1 = new String("hello");
String s2 = new String("hello");
```

- `s1` points to a new memory location, say `6K`
- `s2` points to a different location, say `12K`

Even though both contain the same text `"hello"`, Java creates **separate objects** in the heap (outside the String Pool). So, **no memory optimization happens here**.

🛑 **Duplicate checking only applies to Strings in the String Pool**, not to objects created with `new`.




```java
String s1 = "Hello"; // string pool
String s2 = "Hello"; // string pool
String s3 = new String("Hello");
String s4 = new String("Hello");
System.out.println(s1); // toString() method called
System.out.println(s1 == s2); // comparing address => true
System.out.println(s3 == s4); // comparing address => false
System.out.println(s3 == s1); // comparing address => false
```

📏 Finding Length of a String

```java
int[] arr = new int[7];
System.out.println(arr.length); // variable -> attribute
System.out.println(s1.length()); // method
```

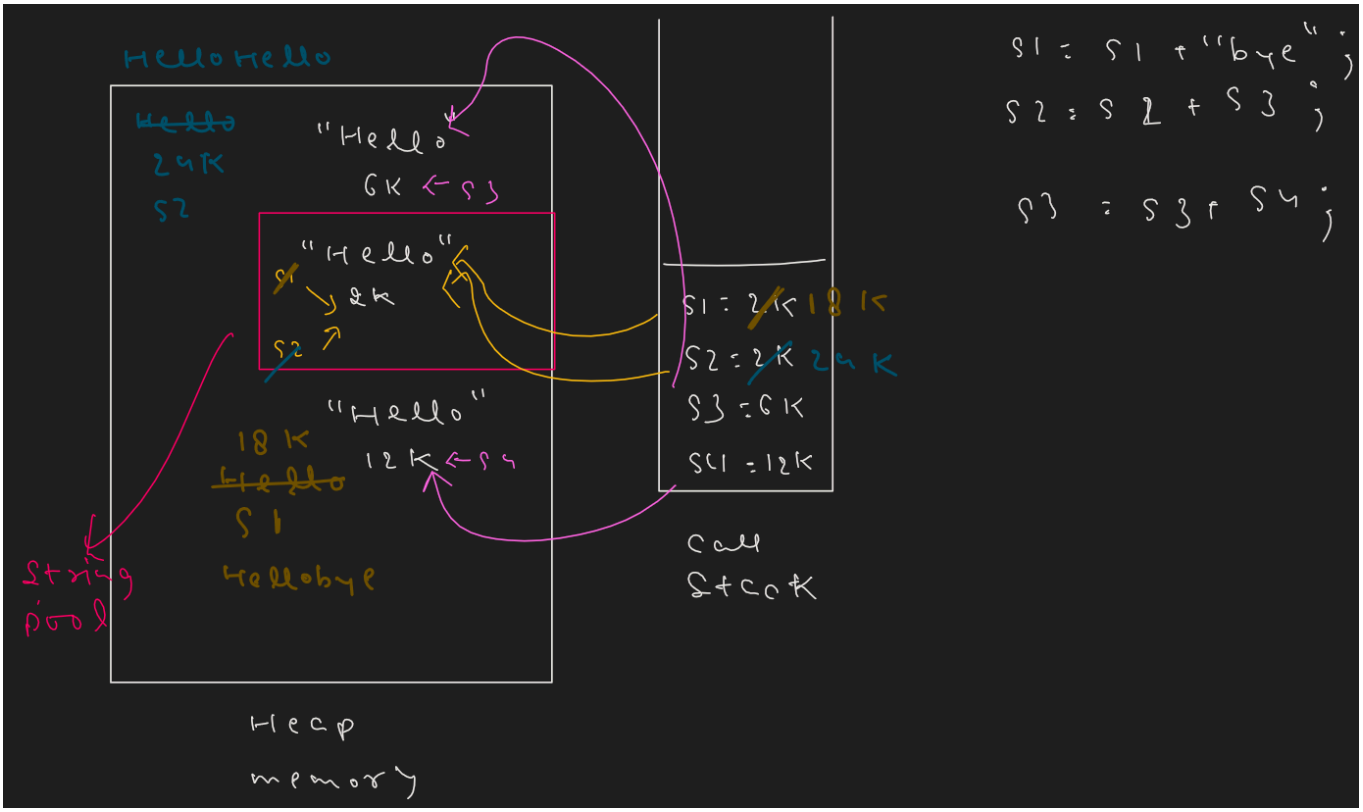
➕ String Concatenation in Java

Concatenation means combining two or more strings into one.

🧠 What Happens When You Modify a String in Java?

The moment you make any change inside a string, a **copy of this string will be formed outside of the pool**, and then the change will happen.

If you have made any change — like **concatenation** — on a string, it will be **outside of the pool**, and a **new address will be allocated** to that string.




```java
public class Main {
	public static void main(String[] args) {
		String s1 = "Hello";
		String s2 = "Hello";
		String s3 = new String("Hello");
		String s4 = new String("Hello");

		s1 = s1 + "bye"; // s1 not in pool
		s2 = s2 + s3; // s2 not in pool
		// now the “Hello” in pool is getting pointed by no one,
		// so it will be cleaned automatically by garbage collector in Java.
		s3 = s3 + s4; // not in pool

		s2 = s2.concat("okay"); // s2= s2+ "okay"; // not in pool

		String s5 = "hello" + "bye"; // pool

		String h1 = "hello"; // pool
		String h2 = h1 + new String(" bye"); // not in pool
		String h3 = h1 + new String(" bye"); // not in pool
		System.out.println(h2==h3);
	}
}
```

String + [any data type] = String

```java
System.out.println("Hey" + 10 + 20 + "Bye"); //Hey1020Bye
System.out.println("Hey" + (10 + 20) + "Bye"); //Hey30Bye
System.out.println(10 + 20 + "Hey" + "Bye"); //30HeyBye
```

🔡 Accessing Characters in a String

The `charAt(int index)` method in Java is used to **retrieve a specific character** from a string, based on its **index**.

```
String s = "akarsh";
System.out.println(s.length());
System.out.println(s.charAt(4));
System.out.println(s.charAt(s.length()-1));
```

✅ Checking if Two Strings Are Equal in Java

```
public class Main {
	public static void main(String[] args) {
		String s1 = "Akarsh";
		String s2 = new String("Akarsh");
		System.out.println(s1.equals(s2));
		System.out.println(equals(s1, s2));
	}

	public static boolean equals(String s1, String s2) {
		if (s1 == s2) {
			return true;
		}
	  if (s1.length() != s2.length()) {
			return false;
		}
		for (int i = 0; i < s1.length(); i++) {
			if (s1.charAt(i) != s2.charAt(i)) {
				return false;
			}
		}
		return true;
	}
}
```

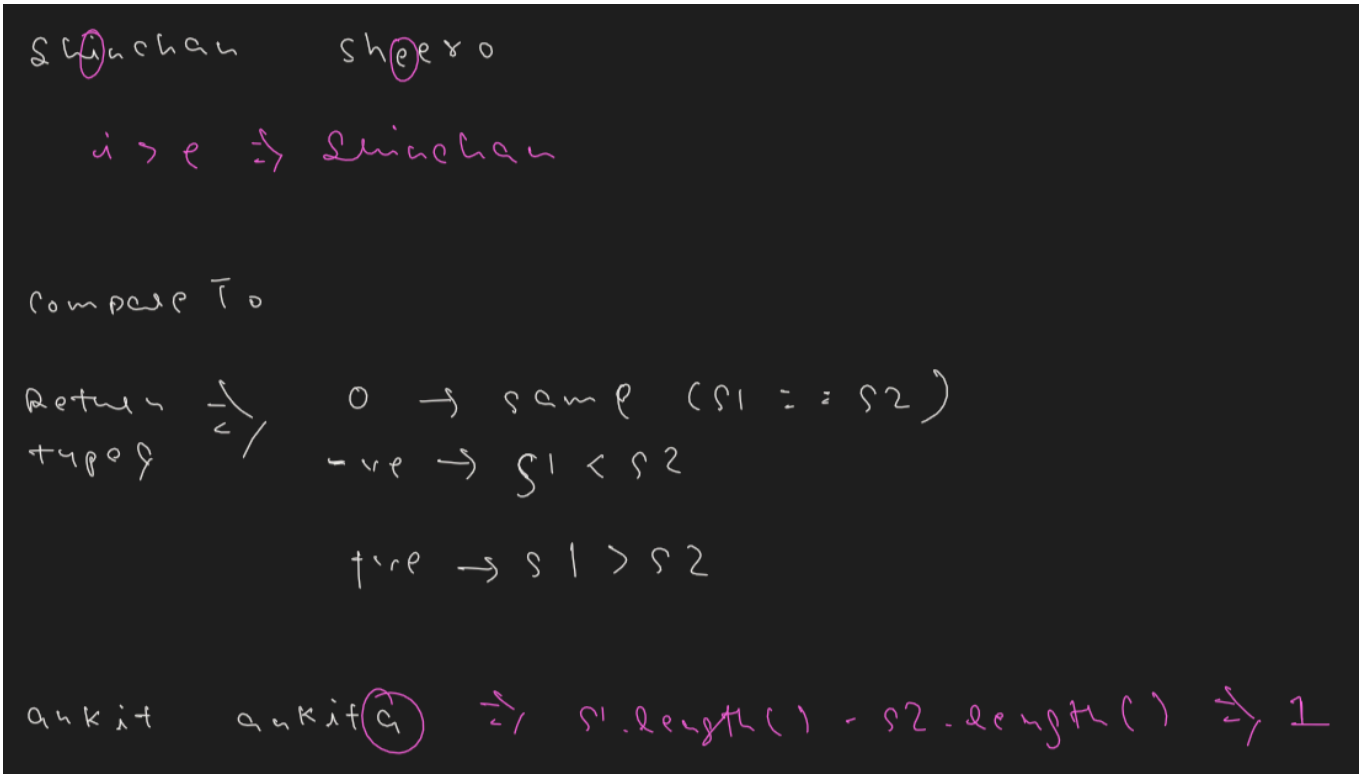
🔁 `compareTo()` in Java — Comparing Strings Lexicographically

It is used to compare two strings lexicographically (i.e., dictionary order).

```
int result = string1.compareTo(string2);
```

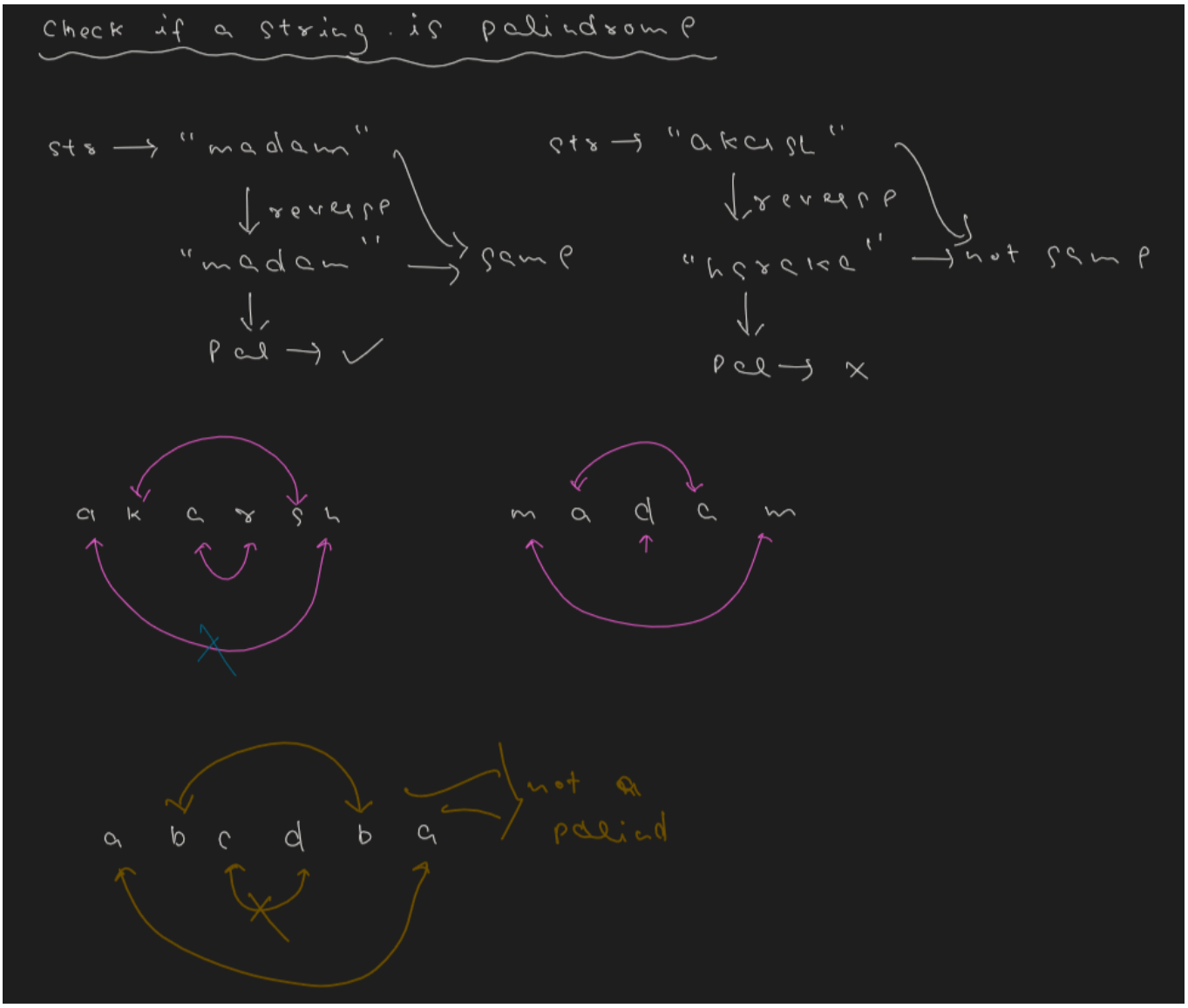
📋 **Return Values:**

- `0` → if both strings are **equal**
- `< 0` → if `string1` comes **before** `string2`
- `> 0` → if `string1` comes **after** `string2`




```java
public class Main {
	public static void main(String[] args) {
		String s1 = "shinchan";
		String s2 = "sheero";
		System.out.println(s1.compareTo(s2));
		// -ve s1<s2 | +ve s1>s2 | 0 s1==s2

		String s3 = "ankita";
		String s4 = "ankit";
		System.out.println(s3.compareTo(s4));

		System.out.println(compareTo(s1, s2));
		System.out.println(compareTo(s3, s4));
	}

	public static int compareTo(String s1, String s2) {
		if (s1 == s2) {
			return 0;
		}
		int n = Math.min(s1.length(), s2.length());
		for (int i = 0; i < n; i++) {
			if (s1.charAt(i) != s2.charAt(i)) {
				return s1.charAt(i) - s2.charAt(i);
			}
		}
		return s1.length() - s2.length();
	}
}
```

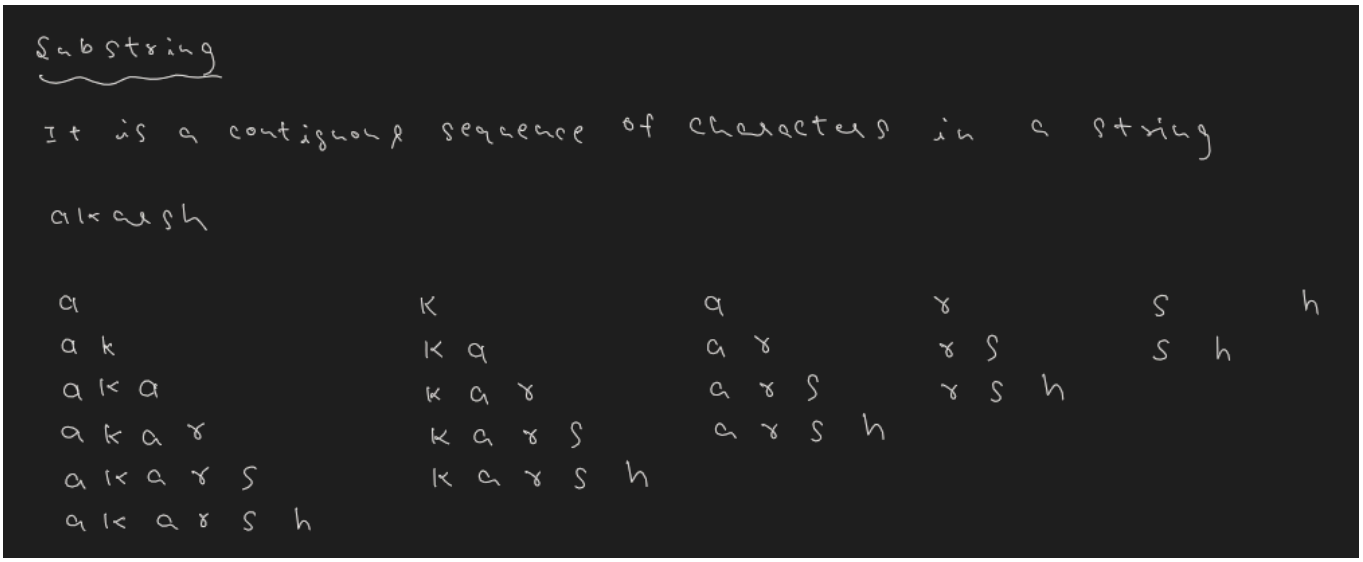
🔁 Check if a String is a Palindrome




```java
public class Main {
	public static void main(String[] args) {
		String s = "madam";
		System.out.println(isPalindrome(s));
	}

	public static boolean isPalindrome(String s) {
		int i = 0;
		int j = s.length() - 1;
		while (i < j) {
			if (s.charAt(i) != s.charAt(j)) {
				return false;
			}
			i++;
			j--;
		}
		return true;
	}
}
```

✂️ What is a Substring?

A substring is a contiguous sequence of characters within a string.
For example, in the word `"akarsh"`, `"kar"`, `"ak"`, and `"sh"` are substrings.



✂️ `substring()`

This method is used to **extract a portion** of a string based on index values.

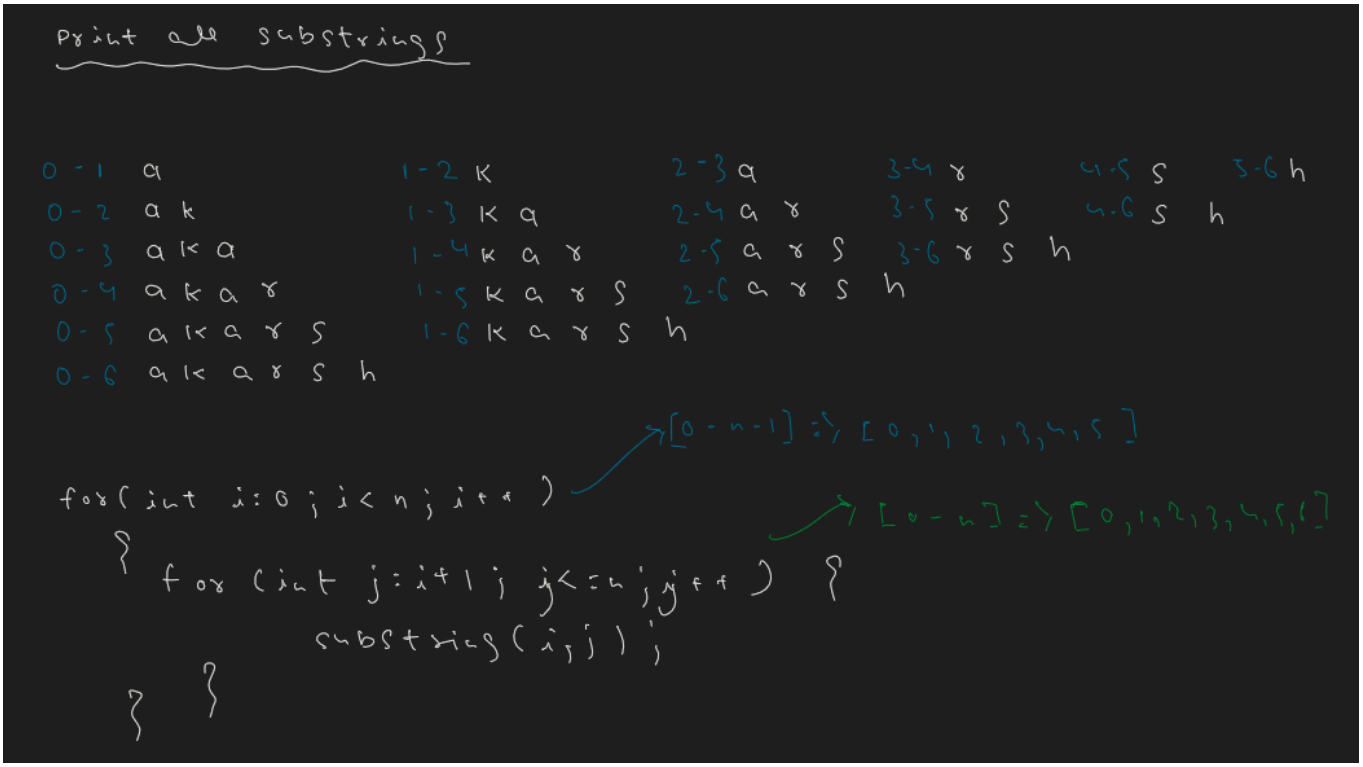
✅ Syntax:

```
string.substring(startIndex)
string.substring(startIndex, endIndex)
```

- `startIndex`: the **starting index** (inclusive)
- `endIndex`: the **ending index** (exclusive)
- Indexing starts at **0**

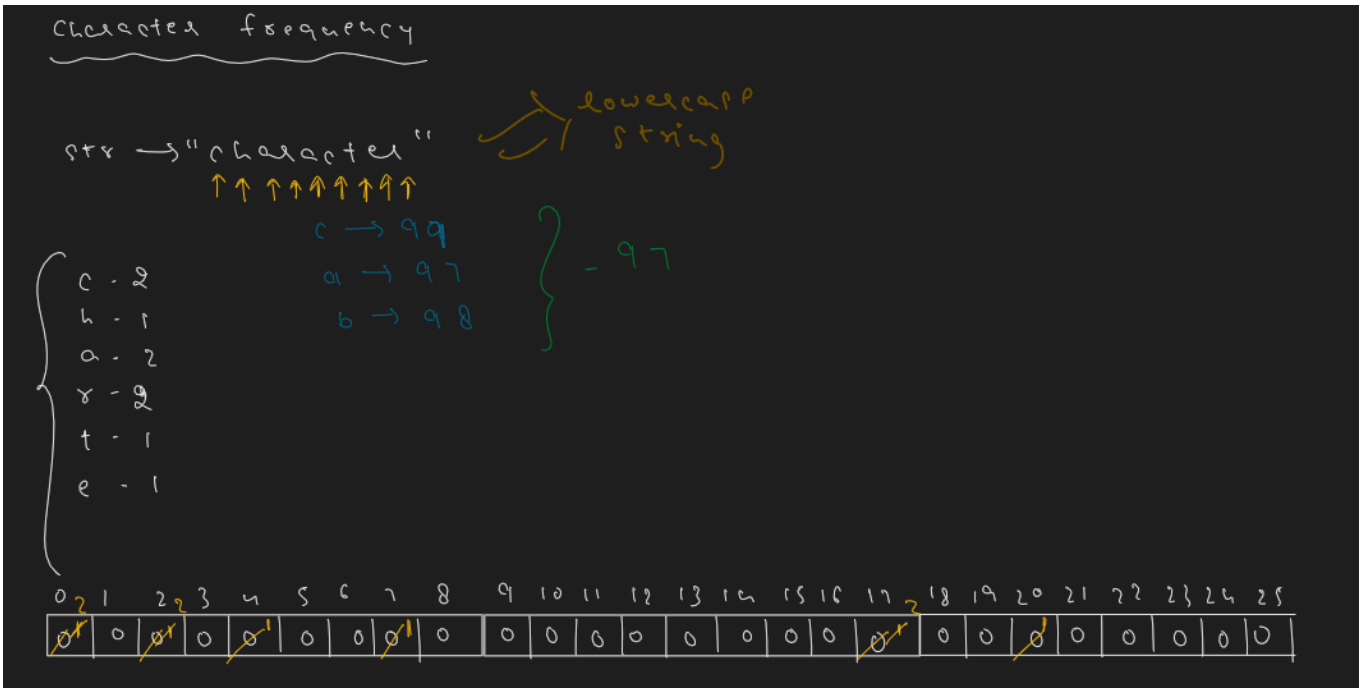
```java
public class Main {
    public static void main(String[] args) {
        String str = "akarsh";

        // Substring from index 1 to the end
        String part1 = str.substring(1);  // Starts from 'k' at index 1
        System.out.println(part1);  // Output: karsh

        // Substring between index 1 and 4
        String part2 = str.substring(1, 4);  // From index 1 to 3 (4 is excluded)
        System.out.println(part2);  // Output: kar

        // Last 3 characters
        String part3 = str.substring(str.length()-3);  // len is 6, so starts at
index 3
        System.out.println(part3);  // Output: rsh

        // String errorExample = str.substring(0, 10);  // str has only 6
characters
        // System.out.println(errorExample); // StringIndexOutOfBoundsException
    }
}
```

✅ Print All Substrings




```java
public class Main {
	public static void main(String[] args) {
		String str = "akarsh";
		printAllSubstrings(str);
	}

	public static void printAllSubstrings(String str) {
		for (int i = 0; i < str.length(); i++) {
			for (int j = i + 1; j <= str.length(); j++) {
				System.out.println(str.substring(i, j));
			}
		}
	}
}
```

🔡 Character Frequency Counter in a String




```java
public class Main {
	public static void main(String[] args) {
		String s = "akarsh";
		int[] freq = new int[26];
		for (int i = 0; i < s.length(); i++) {
			int idx = s.charAt(i) - 97; // s.charAt(i) - 'a';
			freq[idx] = freq[idx] +1;
		}

		for (int i = 0; i < freq.length; i++) {
			if (freq[i] != 0) {
				char ch = (char) (97 + i); // (char) ('a' + i);
				System.out.println(ch + " -> " + freq[i]);
			}
		}
	}
}
```

🎯 First Unique Character in a String

https://leetcode.com/problems/first-unique-character-in-a-string/

```java
class Solution {
    public int firstUniqChar(String s) {
        int[] freq = new int[26];
        for (int i = 0; i < s.length(); i++) {
            int idx = s.charAt(i) - 97;
            freq[idx] = freq[idx] +1;
        }

        for (int i = 0; i < s.length(); i++) {
            int idx = s.charAt(i) - 97;
            if (freq[idx] == 1) {
                return i;
```

```
            }
        }
        return -1;
    }
}
```