📘 JAVA STRING — COMPLETE NOTES

1. Definition

- A String in Java is a sequence of characters enclosed in double quotes (" ").
- In Java, String is not a primitive data type — it is an object of the java.lang.String class.
- Strings are used to store and manipulate **textual data** like names, messages, or IDs.
-

String name = “Rahul";

2. Key Features of Strings

1. **Immutable:** Once created, string values cannot be changed.
2. **Stored in Memory:** Strings are stored in **Heap memory**, and literals go into the **String Constant Pool (SCP)**.
3. **Indexed:** Each character has a position (index starts from 0).
4. **Predefined Methods:** Java provides many built-in methods to work with strings.
5. **Final Class:** The String class is final, meaning it cannot be extended.
6. **Widely Used:** Strings are used in almost all automation test validations (UI & API).

3. Ways to Create Strings

1. Using String Literal
String s1 = "Java";
String s2 = "Java";

Stored in String Constant Pool (SCP).
If the same literal already exists, it will be **reused** (memory optimization).

2. Using new Keyword

String s3 = new String("Java");
String s4 = new String(“Java”);

4. String Memory Storage

Even if the same value exists, each new keyword creates a **new object**.

Type	Memory Area	Description
String Literal	SCP (inside Heap)	Reused if already exists
new String()	Heap Memory	Always creates new object

5. String Immutability

- Once a String object is created, its value **cannot be changed**.
- Any operation like concat(), replace(), or substring() creates a **new object** in memory.

```
String name = "Java";
name.concat(" Programming");
System.out.println(name); // Output: Java
name = name.concat(" Programming");
System.out.println(name); // Output: Java Programming
```

Strings are immutable — once created, they cannot be changed.
New strings are created for every modification.

6. String Constant Pool (SCP)

- SCP is a special memory area inside Heap that stores all unique string literals.
- If a new literal matches an existing one, Java **reuses** it instead of creating a new object.
- It helps in saving memory and improving performance.

7. Difference Between == and .equals()

Comparison Type	Operator / Method	Checks	Example	Output
Reference Comparison		Memory location	`s1 == s2`	true / false
Content Comparison	`.equals()`	Actual text value	`s1.equals(s2)`	TRUE

```
String s1 = new String("Java");
String s2 = new String("Java");
System.out.println(s1 == s2);      // false (different objects)
System.out.println(s1.equals(s2)); // true (same content)
```

What is String in Java?

In Java, a **String** is a **sequence of characters** enclosed in **double quotes (" ")**.
It is **not a primitive data type**, but a **class** present in the java.lang package.
Every string in Java is an **object** of the String class.

Why is String Immutable in Java?

A **String** in Java is **immutable**, meaning **once created, its value cannot be changed**.
Any operation that appears to modify a string (like concat(), replace(), or substring()) actually **creates a new String object** in memory, leaving the original string unchanged.

Reasons for Immutability:

1. **Security:**
   Strings are used in sensitive areas like URLs, usernames, passwords.
   If strings were mutable, malicious code could alter these values.
2. **Caching and Reusability (SCP):**
   Since strings are immutable, Java can safely store and reuse them in the **String Constant Pool** — improving performance.

3. **Thread Safety:**
   Multiple threads can share the same string safely because its value can’t change.
4. **HashCode Consistency:**
   Strings are often used as **keys in HashMaps**.
   If they were mutable, changing the value would break hashing logic.

   Strings are immutable to ensure security, memory optimization, and thread safety.

What is Allure Report?

Allure Report is a **beautiful, interactive test report** that provides:

- ✅ Test status (passed, failed, skipped)
- 🧩 Step details, screenshots, and logs
- 📈 Execution history & trend

Step 1: Install Allure dependencies
npm install --save-dev allure-playwright
npm install -g allure-commandline --save-dev

Or
npm install --save-dev allure-commandline

```
// playwright.config.js
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './tests',
  reporter: [
    ['list'], // default console view
    ['allure-playwright'], // allure report generation
  ],
  use: {
    screenshot: 'only-on-failure',
    video: 'retain-on-failure',
    trace: 'retain-on-failure',
  },
});
```

Step 3: Run Your Tests

Step 4: Generate Allure Report
npx allure generate allure-results --clean -o allure-report

Step 5: Open Allure Report

npx allure open allure-report

```
package Day_29_25_11_12_String;

public class Program1 {

    public static void main(String[] args) {

        // ---------------------------------------------------------
        // STRING METHODS IN JAVA
        // ---------------------------------------------------------

        // 1. length() – returns length of the string
        String str1 = "Automation";
        System.out.println(str1.length());

        int len = str1.length();
        System.out.println(len);

        // 2. toUpperCase() – converts all characters to uppercase
        String name = "Python";
        System.out.println(name.toUpperCase());

        // 3. toLowerCase() – converts all characters to lowercase
        String country = "INDIA";
        System.out.println(country.toLowerCase());

        // 4. charAt(index) – returns character at given index (0-based index)
        String word = "Hello";
        System.out.println(word.charAt(0)); // H
        System.out.println(word.charAt(3)); // l
        // System.out.println(word.charAt(5)); // Error: Index out of range

        String lang = "java";
        System.out.println(lang.charAt(3)); // a

        // 5. equals() – compares two strings (case-sensitive)
        String s1 = "Java";
        String s2 = "Java";
        System.out.println(s1.equals(s2)); // true

        String s3 = "java";
        System.out.println(s1.equals(s3)); // false

        // 6. equalsIgnoreCase() – compares ignoring case
        String s4 = "JAVA";
        String s5 = "java";
        String s6 = "Automation";
        String s7 = "automation";
```

```
System.out.println(s4.equalsIgnoreCase(s5)); // true
System.out.println(s6.equalsIgnoreCase(s7)); // true

// 7. contains() – checks if substring exists
String msg = "Welcome to Java world";
System.out.println(msg.contains("Java"));   // true
System.out.println(msg.contains("Python")); // false

// 8. startsWith() – checks if string starts with given value
String str = "Automation Testing";
System.out.println(str.startsWith("Auto")); // true
System.out.println(str.startsWith("Test")); // false

// 9. endsWith() – checks if string ends with given value
System.out.println(str.endsWith("ing")); // true
System.out.println(str.endsWith("Auto")); // false

// 10. indexOf() – returns first occurrence of given character
String s = "Programmingr";
System.out.println(s.indexOf('r')); // first r = index 1
System.out.println(s.indexOf('g')); // first g = index 3

// find 2nd occurrence of 'r'
System.out.println(s.indexOf('r', s.indexOf('r') + 1));

// find 3rd occurrence of 'r'
System.out.println(s.indexOf('r',
      s.indexOf('r', s.indexOf('r') + 1) + 1));

// 11. lastIndexOf() – returns last occurrence of substring
String name1 = "Programming";
System.out.println(name1.lastIndexOf('g'));  // index of last g
System.out.println(name1.lastIndexOf("mm")); // index of "mm"

// 12. substring() – returns part of string
String word1 = "Programming";
System.out.println(word1.substring(2));     // from index 2 to end
System.out.println(word1.substring(4));     // from index 4 to end

System.out.println(word1.substring(2, 7));   // index 2 to 6

// 13. replace() – replace character or substring
String msg1 = "Java is fun";
System.out.println(msg1.replace("fun", "powerful"));
System.out.println(msg1.replace('J', 'P'));

// 14. trim() – removes leading and trailing spaces
String data = "     Java   ";
```

```
System.out.println(data.length());
System.out.println(data);
System.out.println(data.trim());          // remove start/end spaces
System.out.println(data.stripLeading());  // remove only leading spaces
System.out.println(data.stripTrailing()); // remove only trailing spaces

// 15. concat() – join two strings
String first = "Hello";
String second = "World";
System.out.println(first.concat(" " + second));

// 16. isEmpty() – checks if string is empty
String s10 = "";
System.out.println(s10.isEmpty()); // true
System.out.println(first.isEmpty()); // false

// 17. compareTo() – compares lexicographically
String s11 = "Amit";
String s12 = "Amit";
System.out.println(s11.compareTo(s12)); // 0 (equal)

// 18. toCharArray() – converts string to char array
String word11 = "Java";
char[] arr1 = word11.toCharArray();

System.out.println(arr1); // prints as a char sequence

// Enhanced for loop
for (char c : arr1) {
   System.out.print(c + " ");
}
System.out.println();

// Normal for loop
for (int i = 0; i < arr1.length; i++) {
   System.out.println(arr1[i]);
}

// 19. split() – splits string based on delimiter

String sentence = "Java is easy to learn";
String[] word2 = sentence.split(" ");

for (String w : word2) {
   System.out.println(w);
}

String fruits = "Apple,Mango,Banana,Orange";
```

```
        String[] fruitsList = fruits.split(",");

        for (String f : fruitsList) {
            System.out.println(f);
        }

        String date = "12/11/2025";
        String[] part = date.split("/");

        System.out.println("Day: " + part[0]);
        System.out.println("Month: " + part[1]);
        System.out.println("Year: " + part[2]);
    }
}
```