Java - String

String in Java

1. Introduction

- In Java, **String** is a sequence of characters.
- Unlike other programming languages, in Java, a string is not a primitive data type, but an **object** of the `String` class.
- Strings are widely used for text manipulation, data exchange, and user interaction in applications.

---

2. Characteristics of Strings

- **Immutable:** Once created, a string cannot be changed. Any modification creates a new object.
- **Stored in String Pool:** Java maintains a special memory area called the **String Constant Pool (SCP)** to optimize memory usage.
- **Final Class:** The `String` class is declared as `final`, so it cannot be inherited.
- **Implements Interfaces:** `String` implements `Serializable`, `Comparable<String>`, and `CharSequence`.

---

3. Creating Strings

There are two common ways:

3.1 Using String Literals

```
String s1 = "Hello";
String s2 = "Hello"; // Reuses from String Pool
```

- Both `s1` and `s2` refer to the same object in the **String Pool**.

3.2 Using `new` Keyword

```
String s3 = new String("Hello");
```

- Creates a new string object in the heap memory, not reused from the pool.

4. Memory Management (String Pool vs Heap)

- **String Pool (SCP):** Stores unique string literals for memory efficiency.
- **Heap:** New objects created using `new` keyword are stored here.
- JVM optimizes memory by reusing string literals in the pool.

5. Important String Methods

The `String` class provides numerous built-in methods.

Method	Use	Example	Output
`length()`	Returns length of string	`"Hello".length()`	`5`
`charAt(i)`	Returns character at index	`"Java".charAt(2)`	`v`
`substring(i)`	Part of string from index	`"Programming".substring(3)`	`gramming`
`substring(i, j)`	Part between indices	`"Programming".substring(0, 6)`	`Progra`
`equals()`	Compares 2 strings (case-sensitive)	`"Java".equals("java")`	`false`
`equalsIgnoreCase()`	Compares ignoring case	`"Java".equalsIgnoreCase("java")`	`true`
`compareTo()`	Lexicographic comparison	`"Apple".compareTo("Banana")`	`-1`
`concat()`	Joins 2 strings	`"Hello".concat(" World")`	`Hello World`
`toUpperCase()`	Converts to uppercase	`"java".toUpperCase()`	`JAVA`
`toLowerCase()`	Converts to lowercase	`"JAVA".toLowerCase()`	`java`
`trim()`	Removes spaces from ends	`" Hello ".trim()`	`Hello`
`replace()`	Replaces character/substring	`"Java".replace('a','@')`	`J@v@`
`contains()`	Checks substring present	`"Hello Java".contains("Java")`	`true`
`startsWith()`	Checks beginning	`"Hello".startsWith("He")`	`true`
`endsWith()`	Checks ending	`"Hello".endsWith("lo")`	`true`
`indexOf()`	First occurrence position	`"programming".indexOf("g")`	`3`

Method	Use	Example	Output
`lastIndexOf()`	Last occurrence position	`"programming".lastIndexOf("g")`	`10`
`isEmpty()`	Checks empty string	`"".isEmpty()`	`true`
`isBlank()` (Java 11+)	Empty or only spaces	`" ".isBlank()`	`true`
`split()`	Splits string by regex	`"a,b,c".split(",")`	`["a","b","c"]`
`valueOf()`	Converts to string	`String.valueOf(100)`	`"100"`
`toCharArray()`	Converts to char array	`"Hi".toCharArray()`	`['H','i']`
`format()`	Returns formatted string	`String.format("Age: %d", 20)`	`Age: 20`

6. String Comparison

- **Using `==` operator:** Compares references (memory addresses).
- **Using `.equals()`:** Compares actual content of strings.

```
String a = "Java";
String b = "Java";
String c = new String("Java");

System.out.println(a == b);     // true (same pool reference)
System.out.println(a == c);     // false (different object)
System.out.println(a.equals(c)); // true (same content)
```

7. Mutable Alternatives

Since `String` is immutable, Java provides two mutable classes:

7.1 StringBuilder

- Non-synchronized (faster).
- Used in single-threaded environments.

7.2 StringBuffer

- Synchronized (thread-safe).
- Used in multi-threaded environments.

Example:

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
```

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

8. Advantages of Strings

- Easy to use and powerful built-in methods.
- Immutability ensures security, thread safety, and caching.
- String Pool optimizes memory usage.

9. Disadvantages of Strings

- Immutability can cause performance issues when performing frequent modifications (solution: use `StringBuilder` or `StringBuffer`).
- Memory overhead if too many string objects are created unnecessarily.

Summary

- **String in Java is an object, not a primitive.**
- **Immutable and final**, stored in **String Pool** for efficiency.
- Provides **rich methods** for text manipulation.
- Use `StringBuilder` or `StringBuffer` for mutable string operations.

Escape Sequence Character in Java

An **escape sequence character** in Java is a **special character combination** that begins with a **backslash (`\`)** followed by one or more characters.

It is used to represent characters that **cannot be typed directly** or would otherwise conflict with the syntax of Java code.

For example:

- Writing a newline inside a string → `\n`
- Printing double quotes inside a string → `\"`

Commonly Used Escape Sequences

Escape Sequence	Meaning	Example	Output
`\n`	New Line	`System.out.println("Hello\nWorld");`	Hello World
`\t`	Tab (horizontal)	`System.out.println("Hello\tWorld");`	Hello World
`\\`	Backslash	`System.out.println("C:\\Program Files\\Java");`	C:\Program Files\Java
`\"`	Double Quote	`System.out.println("He said, \"Java is fun\"");`	He said, "Java is fun"

Escape Sequence	Meaning	Example	Output
`\'`	Single Quote	`System.out.println('\'');`	'
`\r`	Carriage Return (moves cursor to start of line)	`System.out.println("Hello\rWorld");`	World (overwrites Hello)
`\b`	Backspace	`System.out.println("Java\b!");`	Jav!
`\f`	Form Feed (new page in printing, rarely used)	`System.out.println("Hello\fWorld");`	Hello (form feed) World
`\uXXXX`	Unicode Character	`System.out.println("\u0041");`	A

Example Code

```
public class EscapeSequenceDemo {
    public static void main(String[] args) {
        System.out.println("Line1\nLine2");         // New line
        System.out.println("Column1\tColumn2");     // Tab
        System.out.println("This is a backslash: \\"); // Backslash
        System.out.println("He said, \"Java is fun!\""); //Double Quote
        System.out.println("Single quote: \'"); // Single Quote
        System.out.println("Carriage\rReturn"); // Carriage Return
        System.out.println("Backspace\bDemo"); // Backspace
        System.out.println("Hello\fWorld");//Form Feed (new page in printing,rare
ly used)
        System.out.println("Unicode A: \u0041"); // Unicode Character
    }
}
```