ROHIT PATKAR

String in Java – A Detailed & Simple Guide with Examples

1. What is a String in Java?

In Java, a String is a sequence of characters enclosed in double quotes. It is used to store and manipulate text. Java provides the String class, which offers many useful methods for handling text data.

✓ Key Features:

- Immutable by default → Once created, its value cannot be changed.
- Stored in the String Pool → Memory-efficient storage.
- Provides various built-in methods for text operations.

2. How to Create a String in Java

There are two ways to create a String:

A) Using String Literals

```
String name = "Alice";
```

- Direct assignment using double quotes.
- Stored in the **String Pool**, which optimizes memory usage by reusing the same object if the same literal is used again.

✓ **Example:**

```
String str1 = "Java";

String str2 = "Java";

System.out.println(str1 == str2);  // true (both refer to the same object in the String Pool)
```

B) Using new Keyword

```
String name = new String("Alice");
```

- Creates a **new object** in **heap memory**.
- Even if the content is the same, it creates a separate object.

✓ **Example:**

```
String str1 = new String("Java");

String str2 = new String("Java");


System.out.println(str1 == str2);    // false (different memory locations)

System.out.println(str1.equals(str2));  // true (content is the same)
```

2. Mutable vs. Immutable Strings

✅ Immutable Strings

A String in Java is immutable, meaning once created, its value cannot be changed. When you modify a string, it creates a new object instead of modifying the original.

✓ Example:

```
String text = "Hello";
text = text + " World";   // Creates a new String object
System.out.println(text); // Output: Hello World
```

- The original text (Hello) is not modified; a **new object** is created with Hello World.

✓ **Why use immutable strings?**

- **Security** → Immutable strings prevent data from being changed unintentionally.
- **Caching & Optimization** → Reusing literals from the **String Pool** saves memory.

✅ Mutable Strings

Java provides two classes for **mutable strings**:

1. StringBuilder → Faster, not thread-safe.
2. StringBuffer → Slower, thread-safe.

Mutable strings allow modifications without creating new objects, making them more memory-efficient.

✔ StringBuilder Example:

```
StringBuilder sb = new StringBuilder("Hello");

sb.append(" World");

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

✔ StringBuffer Example:

```
StringBuffer sb = new StringBuffer("Java");

sb.append(" Rocks");

System.out.println(sb);  // Output: Java Rocks
```

🚦 4. Differences Between String, StringBuilder, and StringBuffer

Feature	`String`	`StringBuilder`	`StringBuffer`
Mutability	Immutable	Mutable	Mutable
Thread-safe	Yes (by being immutable)	No	Yes
Performance	Slow (new object each time)	Fast (no synchronization)	Slower (synchronized)
Memory usage	High (due to immutability)	Low (reuses the same object)	Low

5. Useful String Methods in Java

1. length() → Get the length of the string

```
String text = "Java";

System.out.println(text.length());  // Output: 4
```

2. concat() → Join two strings

```
String first = "Hello";

String second = "World";

System.out.println(first.concat(" ").concat(second));  // Output: Hello World
```

3. equals() and == → Compare strings

- equals() → compares content.
- == → compares memory reference.

```
String str1 = "Java";

String str2 = new String("Java");


System.out.println(str1.equals(str2));   // true (content is the same)

System.out.println(str1 == str2);        // false (different memory locations)
```

4. substring() → Extract part of a string

```
String msg = "Hello, World!";

System.out.println(msg.substring(7));       // World!

System.out.println(msg.substring(0, 5));    // Hello
```

5. replace() → Replace characters

```
String str = "Java is fun!";

System.out.println(str.replace("fun", "awesome"));  // Java is awesome!
```

6. split() → Split a string into an array

```
String fruits = "apple,orange,banana";

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


for (String fruit : arr) {

   System.out.println(fruit);

}

// Output:

// apple

// orange

// banana
```

ROHIT PATKAR

6. Where to Use Strings in Programming?

✅ 1. Data Storage & Display

- To display messages or labels.
- Storing textual data in variables.

✅ 2. Input and Output

- Reading data from the console or files.
- Displaying output messages.

✅ 3. Data Manipulation

- Extracting, formatting, or replacing parts of text.
- Concatenating dynamic strings.

✅ 4. Logging and Debugging

- Displaying logs or error messages.
- Capturing and formatting exception messages.

ROHIT PATKAR

7. Use of Strings in Automation Testing

✅ 1. Handling Test Data

- Strings are used to store test data like usernames, passwords, and URLs.

✅ 2. Validating Responses

- Comparing expected and actual output (e.g., JSON or HTML responses).

```
String actual = "Welcome to Test Automation";

String expected = "Welcome to Test Automation";

assert actual.equals(expected) : "Test failed!";
```

✅ 3. Extracting Values

- Extracting data from API responses or HTML elements.

```
String response = "User: Alice, Age: 30";

String[] data = response.split(",");

System.out.println(data[0]);  // User: Alice
```

✅ 4. String Matching

- Using contains(), matches(), and regex for validation.

```
String pageTitle = "Test Automation Platform";

assert pageTitle.contains("Automation");   // Test passes
```

8. Key Takeaways

1. String is **immutable**, while StringBuilder and StringBuffer are **mutable**.
2. Use StringBuilder for **better performance** in single-threaded applications.
3. Use StringBuffer in **multi-threaded** programs.
4. In **automation testing**, strings are heavily used for validation, data extraction, and comparisons.

Practice Challenge

Write a program that:

1. Accepts a full name (e.g., "Alice Johnson").
2. Splits it into first and last name.
3. Prints them separately.
4. Reverses the full name using StringBuilder.
5.

✅ Top 10 Automation Testing Interview Questions on Strings in Java

1. What is the difference between == and equals() when comparing strings in Java?
2. Why are Strings immutable in Java, and how does it benefit automation testing?
3. How can you reverse a string in Java without using StringBuilder or StringBuffer?
4. What is the difference between String, StringBuilder, and StringBuffer?
5. How do you validate if a string contains a specific substring during automation testing?
6. How do you split a string based on a delimiter in Java?
7. How can you replace part of a string in Java during test execution?
8. How can you convert a String to int and vice versa in Java?
9. How can you compare two strings while ignoring case sensitivity?
10. How can you extract numbers from a string in Java?