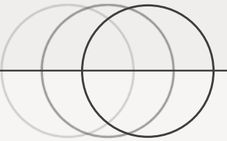
EMIR TOTIĆ - JAVA SOFTWARE ENGINEER

Strings in Java – Deep Dive

Understanding immutability, memory and efficient manipulation

OCTOBER 2025

What is a String in Java?



A String in Java represents a sequence of characters.
Unlike primitive types (int, char, boolean), String is an object defined in the java.lang package.
Internally, it stores data as a character array (char[]), and its behavior is controlled by methods from the String class.

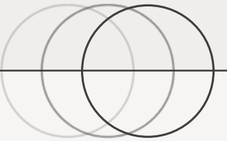
- String language = "Java"; → stored in String Constant Pool.
- new String("Spring Boot") → creates a new object in the heap.

The String class is:

- final → cannot be subclassed
- immutable → once created, cannot be modified
- widely used → for identifiers, configuration keys, JSON, logs, etc.

In Java, a String is not just text — it’s a carefully optimized object for safety and efficiency.

Immutability Explained



Why Strings are Immutable?

Once created, a String object in Java cannot be modified.

Every change produces a new object in memory, while the original remains untouched.

```
String s = "Java";
s.concat(" Rocks!");
System.out.println(s);
```

The output will be *Java*.

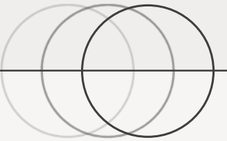
concat() didn't modify s; it created a new String "Java Rocks!".

If we wrote s = s.concat(" Rocks!");, then s would point to the new object.

Reasons for Immutability:

1. Security – Strings are used in sensitive places (file paths, class names, network URLs).
2. Immutability prevents malicious modification after creation.
3. Thread-Safety – Since Strings can't change, multiple threads can safely share them.
4. Caching & Reuse – Enables String Pool, which stores one copy of each literal.
5. Hash Consistency – A String's hashCode() is cached, making it efficient for use in HashMap and HashSet.

The String Pool



Understanding the String Constant Pool

When you create a string literal in Java, it's stored in a special memory region called the String Constant Pool — part of the heap that's optimized for reuse.

If two variables contain the same literal, they point to the same object in the pool, not separate copies.

```
String a = "Java";
String b = "Java";
System.out.println(a == b);
```

Output: TRUE

```
String c = new String("Java");
System.out.println(a == c);
```

Output: FALSE

Key Points

- String literals are automatically interned by the JVM.
- The pool helps save memory and boost performance.
- You can manually add a String to the pool using .intern():
- The pool is part of the Java heap, not the permanent generation (since Java 7).

The String Pool ensures that identical strings are stored only once, making Java applications memory-efficient.

Everyday String Operations

Operation	Example	Output
Concatenation	"Java" + "17"	Java17
Substring	"SpringBoot".substring(6)	Boot
Equals	"Code".equals("code")	false
Ignore Case	"Code".equalsIgnoreCase ("code")	true
Index Of	"Backend".indexOf("end")	4
Length	"Hello".length()	5
Contains	"Microservice".contains("s ervice")	true
Replace	"Java".replace('a', 'o')	Jovo

Chaining Example:

```
String result = "  microservice  "
    .trim()
    .toUpperCase()
    .replace("SERVICE", "SYSTEM");
System.out.println(result);
```

The output will be: MICROSYSTEM

These methods are optimized for readability and performance — learn to chain them effectively for clean, expressive code.

StringBuilder vs StringBuffer

Quick Comparison:

Feature	StringBuilder	StringBuffer
Mutability	Yes	Yes
Thread-Safe	No	Yes
Synchronization	No	Yes
Performance	Faster	Slower (due to locking)
Use Case	Single-threaded	Multi-threaded

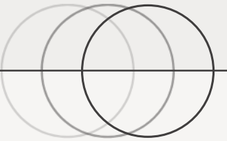
Example:

```
StringBuilder sb = new StringBuilder("Java");
sb.append(" Rocks!");
System.out.println(sb);
```

Output: Java Rocks!

Use StringBuilder for local text processing, and StringBuffer only when working with shared data across threads.

Common Pitfalls & Hidden Details



== vs .equals()

== compares references, while .equals() compares content.

```
String a = "Code";
String b = new String("Code");
System.out.println(a == b);      // false
System.out.println(a.equals(b)); // true
```

Compile-Time Concatenation

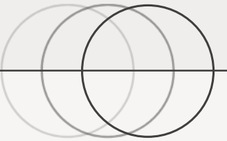
String literals joined at compile time are stored as one object in the pool:

```
String x = "Ja" + "va";
String y = "Java";
System.out.println(x == y); // true
```

But concatenation with variables happens at runtime, producing a new object:

```
String p = "Ja";
String q = p + "va";
System.out.println(q == y); // false
```

Common Pitfalls & Hidden Details



Strings in Loops

Avoid using += in loops — it creates many temporary objects.

Inefficient:

```
String result = "";
for (int i = 0; i < 1000; i++) {
    result += i;
}
```

Efficient:

```
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 1000; i++) {
    sb.append(i);
}
String result = sb.toString();
```

In a loop, using += on strings causes a new String object to be created on each iteration because strings are immutable. This leads to unnecessary memory allocation and slower performance. StringBuilder, on the other hand, modifies the same object in memory, making it much faster and more efficient for repeated concatenations.

Modern String Features (Java 11+)

Modern Java versions added several useful String methods that make code cleaner and more expressive.

Method	Description	Example
isBlank()	Checks if the string is empty or contains only whitespace.	" ".isBlank() → true
strip()	Removes leading and trailing whitespace (Unicode-aware, unlike trim()).	" dev ".strip() → "dev"
repeat(int n)	Returns a new string repeated n times.	"Java".repeat(3) → "JavaJavaJava"
lines()	Splits a string into a stream of lines.	"one\ntwo\nthree".lines().count() → 3
formatted()	A cleaner alternative to String.format().	"Hello, %s!".formatted("World") → "Hello, World!"

Summary & Key Takeaways

Key Points to Remember:

- A String in Java is an immutable object — every change creates a new instance
- The String Pool enables memory efficiency and reuse of identical literals
- Use .equals() for content comparison — not ==
- Prefer StringBuilder for heavy concatenation and loops
- Modern Java versions introduced helpful methods like isBlank(), strip(), and formatted() for cleaner code

Tips to choose the right tool:

String	For static or constant text that doesn't change.
StringBuilder	For dynamic text operations inside a single thread (e.g. loops, concatenations).
StringBuffer	For text operations shared across multiple threads.

Thank you for reading

Let’s connect


www.linkedin.com/in/emirtotic


https://emirtotic.github.io/portfolio-site


emirtotic@gmail.com