Strings in JAVA

Strings in Java represent sequences of characters and are instances of the `String` class. Once created, a string's value cannot be changed. Any operation that modifies a string results in a new string. Java maintains a pool of strings to optimize memory usage. Literal strings are interned and stored in this pool. Strings in Java are internally represented using UTF-16 encoding, allowing for the representation of Unicode characters.

Syntax:

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

Example:

```
public class StringExample {
    public static void main(String[] args) {
        String greeting = "Hello, World!";
        System.out.println(greeting);
    }
}
```

Two Types of Creation of String

1. **String Literal**: Created by directly assigning a value in double quotes. These strings are interned and stored in the string pool.

   ```
   String str1 = "Hello";
   ```

2. **Using `new` Keyword**: Created by instantiating the `String` class with the `new` keyword. These strings are stored in the heap memory.

   ```
   String str2 = new String("Hello");
   ```

Memory Efficiency: String literals are more memory-efficient due to interning, whereas strings created with `new` occupy more memory. The string pool helps save memory by storing only one copy of each distinct string literal. ‘==‘ checks for reference equality, while `.equals()` checks for value equality between strings.

String Class in Java

`String` is a final class in Java, which means it cannot be subclassed. Strings are immutable, meaning their values cannot be changed once created. This ensures thread safety. The `String` class provides multiple constructors to create strings from byte arrays, character arrays, and other strings. It Includes methods such as `length()`, `charAt(int index)`, `substring(int beginIndex, int endIndex)`, `indexOf(String str)`, and `concat(String str)`. It Utilizes a string pool for efficient memory management of string literals.

String Constructors in JAVA

Constructor	Description	Example
`String(byte[] byte_arr)`	Construct a new String by decoding the byte array using the platform's default character set.	`byte[] b_arr = {71, 101, 101, 107, 115}; String s_byte = new String(b_arr); // "Geeks"`
`String(byte[] byte_arr, Charset char_set)`	Construct a new String by decoding the byte array using the specified `char_set`.	`byte[] b_arr = {71, 101, 101, 107, 115}; Charset cs = Charset.defaultCharset(); String s_byte_char = new String(b_arr, cs); // "Geeks"`
`String(byte[] byte_arr, String char_set_name)`	Construct a new String by decoding the byte array using the specified `char_set_name`.	`byte[] b_arr = {71, 101, 101, 107, 115}; String s = new String(b_arr, "US-ASCII"); // "Geeks"`
`String(byte[] byte_arr, int start_index, int length)`	Construct a new String from the bytes array starting at `start_index` and for `length` characters.	`byte[] b_arr = {71, 101, 101, 107, 115}; String s = new String(b_arr, 1, 3); // "eek"`
`String(byte[] byte_arr, int start_index, int length, Charset char_set)`	Construct a new String from the bytes array starting at `start_index` and for `length` characters using the specified `char_set`.	`byte[] b_arr = {71, 101, 101, 107, 115}; Charset cs = Charset.defaultCharset(); String s = new String(b_arr, 1, 3, cs); // "eek"`
`String(byte[] byte_arr, int start_index, int length, String char_set_name)`	Construct a new String from the bytes array starting at `start_index` and for `length` characters using the specified `char_set_name`.	`byte[] b_arr = {71, 101, 101, 107, 115}; String s = new String(b_arr, 1, 4, "US-ASCII"); // "eeks"`
`String(char[] char_arr)`	Allocate a new String from the given character array.	`char char_arr[] = {'G', 'e', 'e', 'k', 's'}; String s = new String(char_arr); // "Geeks"`

`String(char[] char_array, int start_index, int count)`	Allocate a String from the given character array starting at `start_index` and for `count` characters.	`char char_arr[] = {'G', 'e', 'e', 'k', 's'}; String s = new String(char_arr , 1, 3); // "eek"`
`String(int[] uni_code_points, int offset, int count)`	Allocate a String from the given Unicode code points starting at `offset` and for `count` characters.	`int[] uni_code = {71, 101, 101, 107, 115}; String s = new String(uni_code, 1, 3); // "eek"`
`String(StringBuffer s_buffer)`	Allocate a new String from the string in `s_buffer`.	`StringBuffer s_buffer = new StringBuffer("Geeks"); String s = new String(s_buffer); // "Geeks"`
`String(StringBuilder s_builder)`	Allocate a new String from the string in `s_builder`.	`StringBuilder s_builder = new StringBuilder("Geeks"); String s = new String(s_builder); // "Geeks"`

String Methods in JAVA

Method	Description
`int length()`	Returns the number of characters in the String.
`char charAt(int i)`	Returns the character at the specified index `i`.
`String substring(int i)`	Returns the substring from the specified index `i` to the end.
`String substring(int i, int j)`	Returns the substring from the specified index `i` to `j-1`.
`String concat(String str)`	Concatenates the specified string `str` to the end of this string.
`int indexOf(String s)`	Returns the index of the first occurrence of the specified string `s`. Returns `-1` if `s` is not found.
`int indexOf(String s, int i)`	Returns the index of the first occurrence of the specified string `s`, starting at the specified index `i`.
`int lastIndexOf(String s)`	Returns the index of the last occurrence of the specified string `s`. Returns `-1` if `s` is not found.
`boolean equals(Object otherObj)`	Compares this string to the specified object `otherObj`.
`boolean equalsIgnoreCase(String anotherString)`	Compares this string to another string, ignoring case considerations.
`int compareTo(String anotherString)`	Compares two strings lexicographically.
`int compareToIgnoreCase(String anotherString)`	Compares two strings lexicographically, ignoring case considerations.
`String toLowerCase()`	Converts all the characters in the String to lower case.
`String toUpperCase()`	Converts all the characters in the String to upper case.
`String trim()`	Returns a copy of the String, with leading and trailing whitespaces removed.
`String replace(char oldChar, char newChar)`	Returns a new string by replacing all occurrences of `oldChar` with `newChar`.

`boolean contains(String str)`	Returns `true` if this string contains the specified string `str`.
`char[] toCharArray()`	Converts this String to a new character array.
`boolean startsWith(String prefix)`	Returns `true` if this string starts with the specified prefix.