# Strings in Java

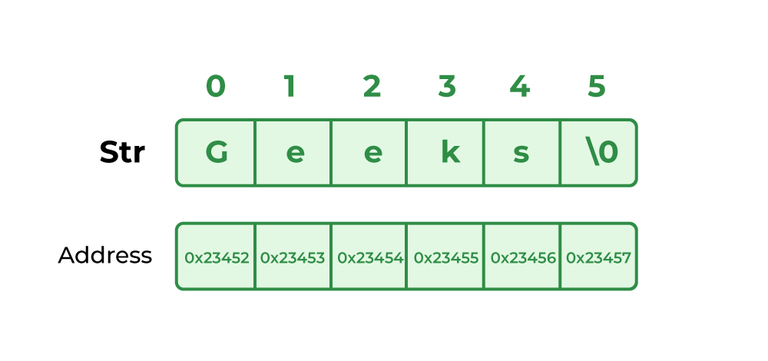
In the given example only one object will be created. Firstly JVM will not find any string object with the value “Welcome” in the string constant pool, so it will create a new object. After that it will find the string with the value “Welcome” in the pool, it will not create a new object but will return the reference to the same instance. In this article, we will learn about Java Strings.

## What are Strings in Java?

Strings are the type of objects that can store the character of values and in Java, every character is stored in 16 bits i,e using UTF 16-bit encoding. A string acts the same as an array of characters in Java.

**Example:**

String name = "Geeks";



*String Example in Java*

**Below is an example of a String in Java:**

// Java Program to demonstrate

// String

public class StringExample {

// Main Function

public static void main(String args[])

{

String str = new String("example");

// creating Java string by new keyword

// this statement create two object i.e

// first the object is created in heap

// memory area and second the object is

// created in String constant pool.

System.out.println(str);

}

}

## ****Ways of Creating a String****

There are two ways to create a string in Java:

* String Literal
* Using new Keyword

### ****1. String literal****

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

**Example:**

String demoString = “GeeksforGeeks”;

### ****2. Using new keyword****

* String s = new String(“Welcome”);
* In such a case, JVM will create a new string object in normal (non-pool) heap memory and the literal “Welcome” will be placed in the string constant pool. The variable s will refer to the object in the heap (non-pool)

**Example:**

String demoString = new String (“GeeksforGeeks”);

## Immutable String in Java

In Java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. Once a string object is created its data or state can’t be changed but a new string object is created.

**Below is the implementation of the topic:**

// Java Program to demonstrate Immutable String in Java

import java.io.\*;

class GFG {

public static void main(String[] args)

{

String s = "Sachin";

// concat() method appends

// the string at the end

s.concat(" Tendulkar");

// This will print Sachin

// because strings are

// immutable objects

System.out.println(s);

}

}

**Output**

Sachin

Here Sachin is not changed but a new object is created with “Sachin Tendulkar”. That is why a string is known as immutable.

As you can see in the given figure that two objects are created but s reference variable still refers to “Sachin” and not to “Sachin Tendulkar”. But if we explicitly assign it to the reference variable, it will refer to the “Sachin Tendulkar” object.

// Java Program to demonstrate Explicitly assigned strings

import java.io.\*;

class GFG {

public static void main(String[] args)

{

String name = "Sachin";

name = name.concat(" Tendulkar");

System.out.println(name);

}

}

## ****Memory Allotment of String****

Whenever a String Object is created as a literal, the object will be created in the String constant pool. This allows JVM to optimize the initialization of String literal.

**Example:**

String demoString = "Geeks";

The string can also be declared using a **new** operator i.e. dynamically allocated. In case of String are dynamically allocated they are assigned a new memory location in the heap. This string will not be added to the String constant pool.

**Example:**

String demoString = new String("Geeks");

If you want to store this string in the constant pool then you will need to “intern” it.

**Example:**

String internedString = demoString.intern();   
// this will add the string to string constant pool.

It is preferred to use String literals as it allows JVM to optimize memory allocation.

An example that shows how to declare a String

// Java code to illustrate String

import java.io.\*;

import java.lang.\*;

class Test {

public static void main(String[] args)

{

// Declare String without using new operator

String name = "GeeksforGeeks";

// Prints the String.

System.out.println("String name = " + name);

// Declare String using new operator

String newString = new String("GeeksforGeeks");

// Prints the String.

System.out.println("String newString = " + newString);

}

}

## Frequently Asked Questions

### 1. What are strings in Java?

Strings are the types of objects which can store characters as elements.

### 2. Why string objects are immutable in Java?

Because java uses the concept of string literal. Suppose there are 5 reference variables, all refer to one object “Sachin”. If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in Java.

### 3. Why Java uses the concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

# Java Strings

In Java, a string is a sequence of characters. For example, "hello" is a string containing a sequence of characters 'h', 'e', 'l', 'l', and 'o'.

We use **double quotes** to represent a string in Java. For example,

// create a string

String type = "Java programming";

Here, we have created a string variable named type. The variable is initialized with the string Java Programming.

### Example: Create a String in Java

class Main {

public static void main(String[] args) {

// create strings

String first = "Java";

String second = "Python";

String third = "JavaScript";

// print strings

System.out.println(first); // print Java

System.out.println(second); // print Python

System.out.println(third); // print JavaScript

}

}

In the above example, we have created three strings named first, second, and third. Here, we are directly creating strings like primitive types.

**Note**: Strings in Java are not primitive types (like int, char, etc). Instead, all strings are objects of a predefined class named String.

And, all string variables are instances of the String class.

## Java String Operations

Java String provides various methods to perform different operations on strings. We will look into some of the commonly used string operations.

### 1. Get length of a String

To find the length of a string, we use the length() method of the String. For example,

class Main {

public static void main(String[] args) {

// create a string

String greet = "Hello! World";

System.out.println("String: " + greet);

// get the length of greet

int length = greet.length();

System.out.println("Length: " + length);

}

}

**Output**

String: Hello! World

Length: 12

### 2. Join Two Java Strings

We can join two strings in Java using the concat() method. For example,

class Main {

public static void main(String[] args) {

// create first string

String first = "Java ";

System.out.println("First String: " + first);

// create second

String second = "Programming";

System.out.println("Second String: " + second);

// join two strings

String joinedString = first.concat(second);

System.out.println("Joined String: " + joinedString);

}

}

**Output**

First String: Java

Second String: Programming

Joined String: Java Programming

We can also join two strings using the + operator in Java.

### 3. Compare two Strings

In Java, we can make comparisons between two strings using the equals() method. For example,

class Main {

public static void main(String[] args) {

// create 3 strings

String first = "java programming";

String second = "java programming";

String third = "python programming";

// compare first and second strings

boolean result1 = first.equals(second);

System.out.println("Strings first and second are equal: " + result1);

// compare first and third strings

boolean result2 = first.equals(third);

System.out.println("Strings first and third are equal: " + result2);

}

}

**Output**

Strings first and second are equal: true

Strings first and third are equal: false

**Note**: We can also compare two strings using the == operator in Java. However, this approach is different than the equals() method.

## Escape character in Java Strings

The escape character is used to escape some of the characters present inside a string.

Suppose we need to include double quotes inside a string.

// include double quote

String example = "This is the "String" class";

Since strings are represented by **double quotes**, the compiler will treat "This is the " as the string. Hence, the above code will cause an error.

To solve this issue, we use the escape character \ in Java. For example,

// use the escape character

String example = "This is the \"String\" class.";

## Java Strings are Immutable

In Java, strings are **immutable**. This means, once we create a string, we cannot change that string.

To understand it more deeply, consider an example:

// create a string

String example = "Hello! ";

Here, we have created a string variable named example. The variable holds the string "Hello! ".

Now suppose we want to change the string.

// add another string "World"

// to the previous tring example

example = example.concat(" World");

Here, we are using the concat() method to add another string World to the previous string.

It looks like we are able to change the value of the previous string. However, this is not true.

Let's see what has happened here,

1. JVM takes the first string "Hello! "
2. creates a new string by adding "World" to the first string
3. assign the new string "Hello! World" to the example variable
4. the first string "Hello! " remains unchanged

## Creating strings using the new keyword

So far we have created strings like primitive types in Java.

Since strings in Java are objects, we can create strings using the new keyword as well. For example,

// create a string using the new keyword

String name = new String("Java String");

In the above example, we have created a string name using the new keyword.

Here, when we create a string object, the String() constructor is invoked.

### Example: Create Java Strings using the new keyword

class Main {

public static void main(String[] args) {

// create a string using new

String name = new String("Java String");

System.out.println(name); // print Java String

}

}

### Create String using literals vs new keyword

Now that we know how strings are created using string literals and the new keyword, let's see what is the major difference between them.

In Java, the JVM maintains a **string pool** to store all of its strings inside the memory. The string pool helps in reusing the strings.

1. While creating strings using string literals,

String example = "Java";

Here, we are directly providing the value of the string (Java). Hence, the compiler first checks the string pool to see if the string already exists.

* **If the string already exists**, the new string is not created. Instead, the new reference, example points to the already existed string (Java).
* **If the string doesn't exist**, the new string (Java is created.

2. While creating strings using the new keyword,

String example = new String("Java");

Here, the value of the string is not directly provided. Hence, a new "Java" string is created even though "Java" is already present inside the memory pool.

## Methods of Java String

|  |  |
| --- | --- |
| Methods | Description |
| [contains()](https://www.programiz.com/java-programming/library/string/contains) | checks whether the string contains a substring |
| [substring()](https://www.programiz.com/java-programming/library/string/substring) | returns the substring of the string |
| [join()](https://www.programiz.com/java-programming/library/string/join) | join the given strings using the delimiter |
| [replace()](https://www.programiz.com/java-programming/library/string/replace) | replaces the specified old character with the specified new character |
| [replaceAll()](https://www.programiz.com/java-programming/library/string/replaceall) | replaces all substrings matching the regex pattern |
| [replaceFirst()](https://www.programiz.com/java-programming/library/string/replacefirst) | replace the first matching substring |
| [charAt()](https://www.programiz.com/java-programming/library/string/charat) | returns the character present in the specified location |
| [getBytes()](https://www.programiz.com/java-programming/library/string/getbytes) | converts the string to an array of bytes |
| [indexOf()](https://www.programiz.com/java-programming/library/string/indexof) | returns the position of the specified character in the string |
| [compareTo()](https://www.programiz.com/java-programming/library/string/compareto) | compares two strings in the dictionary order |
| [compareToIgnoreCase()](https://www.programiz.com/java-programming/library/string/comparetoignorecase) | compares two strings ignoring case differences |
| [trim()](https://www.programiz.com/java-programming/library/string/trim) | removes any leading and trailing whitespaces |
| [format()](https://www.programiz.com/java-programming/library/string/format) | returns a formatted string |
| [split()](https://www.programiz.com/java-programming/library/string/split) | breaks the string into an array of strings |
| [toLowerCase()](https://www.programiz.com/java-programming/library/string/tolowercase) | converts the string to lowercase |
| [toUpperCase()](https://www.programiz.com/java-programming/library/string/touppercase) | converts the string to uppercase |
| [valueOf()](https://www.programiz.com/java-programming/library/string/valueof) | returns the string representation of the specified argument |
| [toCharArray()](https://www.programiz.com/java-programming/library/string/tochararray) | converts the string to a char array |
| [matches()](https://www.programiz.com/java-programming/library/string/matches) | checks whether the string matches the given regex |
| [startsWith()](https://www.programiz.com/java-programming/library/string/startswith) | checks if the string begins with the given string |
| [endsWith()](https://www.programiz.com/java-programming/library/string/endswith) | checks if the string ends with the given string |
| [isEmpty()](https://www.programiz.com/java-programming/library/string/isempty) | checks whether a string is empty of not |
| [intern()](https://www.programiz.com/java-programming/library/string/intern) | returns the canonical representation of the string |
| [contentEquals()](https://www.programiz.com/java-programming/library/string/contentequals) | checks whether the string is equal to charSequence |
| [hashCode()](https://www.programiz.com/java-programming/library/string/hashcode) | returns a hash code for the string |
| [subSequence()](https://www.programiz.com/java-programming/library/string/subsequence) | returns a subsequence from the string |

# Java String contains()

The contains() method checks whether the specified string (sequence of characters) is present in the string or not.

### Example

class Main {

public static void main(String[] args) {

String str1 = "Java String contains()";

// check if str1 contains "Java"

boolean result = str1.contains("Java");

System.out.println(result);

}

}

// Output: true

## Example 1: Java String contains()

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

Boolean result;

// check if str1 contains "Java"

result = str1.contains("Java");

System.out.println(result); // true

// check if str1 contains "Python"

result = str1.contains("Python");

System.out.println(result); // false

// check if str1 contains ""

result = str1.contains("");

System.out.println(result); // true

}

}

## Example 2: Using contains() With if...else

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Java";

String str3 = "java";

Boolean result;

// true because "Learn Java" contains "Java"

if (str1.contains(str2)) {

System.out.println(str1 + " contains " + str2);

}

else {

System.out.println(str1 + " doesn't contains " + str2);

}

// contains() is case-sensitive

// false because "Learn Java" doesn't contains "java"

if (str1.contains(str3)) {

System.out.println(str1 + " contains " + str3);

}

else {

System.out.println(str1 + " doesn't contain " + str3);

}

}

}

# Java String equalsIgnoreCase()

string.equalsIgnoreCase(String str)

## equalsToIgnoreCase() Return Value

* **returns true** if the strings are equal, ignoring case considerations
* **returns false** if the strings are not equal
* **returns false** if the str argument is null

## Example 1: Java String equalsIgnoreCase()

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "learn java";

String str3 = "Learn Kolin";

Boolean result;

// comparing str1 with str2

result = str1.equalsIgnoreCase(str2);

System.out.println(result); // true

// comparing str1 with str3

result = str1.equalsIgnoreCase(str3);

System.out.println(result); // false

// comparing str3 with str1

result = str3.equalsIgnoreCase(str1);

System.out.println(result); // false

}

}

## Example 2: Check if Two Strings are Equal

class Main {

public static void main(String[] args) {

String str1 = "LEARN JAVA";

String str2 = "Learn Java";

// if str1 and str2 are equal (ignoring case differences),

// the result is true

if (str1.equalsIgnoreCase(str2)) {

System.out.println("str1 and str2 are equal");

}

else {

System.out.println("str1 and str2 are not equal");

}

}

}

# Java String compareTo()

The compareTo() method compares two strings lexicographically (in the dictionary order). The comparison is based on the Unicode value of each character in the strings.

### Example

class Main {

public static void main(String[] args) {

String str1 = "Learn Java";

String str2 = "Learn Kolin";

int result;

// comparing str1 with str2

result = str1.compareTo(str2);

System.out.println(result);

}

}

// Output: -1

## Example 2: Check if Two Strings are Equal

class Main {

public static void main(String[] args) {

String str1 = "Learn Python";

String str2 = "Learn Java";

// if str1 and str2 are equal, the result is 0

if (str1.compareTo(str2) == 0) {

System.out.println("str1 and str2 are equal");

}

else {

System.out.println("str1 and str2 are not equal");

}

}

}

**Output**

str1 and str2 are not equal

# Java String split()

The split() method divides the string at the specified regex and returns an array of substrings.

### Example

The split() method divides the string at the specified regex and returns an array of substrings.

### Example

class Main {

public static void main(String[] args) {

String text = "Java is a fun programming language";

// split string from space

String[] result = text.split(" ");

System.out.print("result = ");

for (String str : result) {

System.out.print(str + ", ");

}

}

}

// Output: result = Java, is, a, fun, programming, language,

## Example 1: split() Without limit Parameter

// importing Arrays to convert array to string

// used for printing arrays

import java.util.Arrays;

class Main {

public static void main(String[] args) {

String vowels = "a::b::c::d:e";

// splitting the string at "::"

// storing the result in an array of strings

String[] result = vowels.split("::");

// converting array to string and printing it

System.out.println("result = " + Arrays.toString(result));

}

}

# Java String substring()

The Java substring() method extracts a part of the [string](https://www.programiz.com/java-programming/string) (substring) and returns it.

### Example

class Main {

public static void main(String[] args) {

String str1 = "java is fun";

// extract substring from index 0 to 3

System.out.println(str1.substring(0, 4));

}

}

// Output: java

## substring() Argument

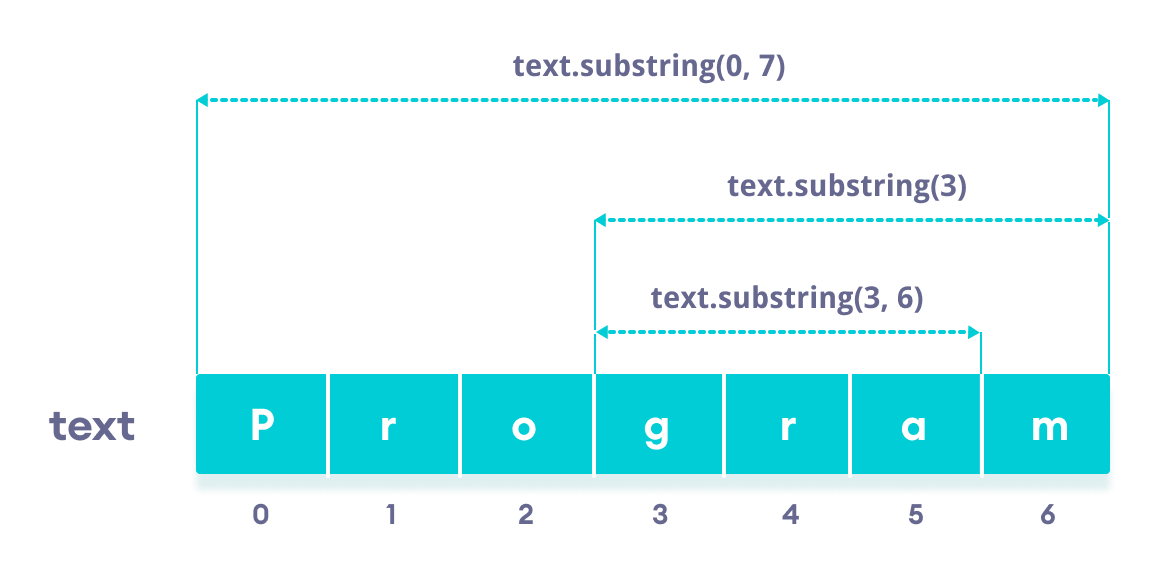
The substring() method can take a maximum of two arguments.

* **startIndex** - the beginning index
* **endIndex** (optional) - the ending index

## substring() Return Value

The substring() method returns a substring from the given string.

* The substring begins with the character at the startIndex and extends to the character at index endIndex - 1.
* If the endIndex is not passed, the substring begins with the character at the specified index and extends to the end of the string.

Working of Java String substring() method

**Note:** You will get an error if

* startIndex/endIndex is negative or greater than string's length
* startIndex is greater than endIndex

## Example 1: Java substring() With Only Start Index

class Main {

public static void main(String[] args) {

String str1 = "program";

// 1st character to the last character

System.out.println(str1.substring(0)); // program

// 4th character to the last character

System.out.println(str1.substring(3)); // gram

}

}

## Example 2: Java substring() With Start and End Index

class Main {

public static void main(String[] args) {

String str1 = "program";

// 1st to the 7th character

System.out.println(str1.substring(0, 7)); // program

// 1st to the 5th character

System.out.println(str1.substring(0, 5)); // progr

// 4th to the 5th character

System.out.println(str1.substring(3, 5)); // gr

}

}

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



Data types – primitive data type (int, float, double, long, short, Boolean,char)

Non –primitive data – arrays, strings, class

String – sequence of characters

String str=”Hello” -- primitive type

2 bytes of space – 16 bits

String literal

String str = new str(“Hello”)

Literal – numeric , string

A=10 – 10 is called numeric literal (value)

Name=”Hello”

Immutable – whose value cannot be changed

Name=”Raju”

Name=”Helln”

String is group of characters

Char [] = [‘h’,’e’,’l’,’l’,’o’]

0 1 2 3 4

Operations

Equal

uppercase