
Java String

In this tutorial, we will learn about Java Strings, how to create them, and various methods of String with the help of examples.

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`.

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.

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.

However, there is another way of creating Java strings (using the `new` keyword). We will learn about that later in this tutorial.

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
```

In the above example, the `length()` method calculates the total number of characters in the string and returns it. To learn more, visit Java String length().

---

2. Join two 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
```

In the above example, we have created two strings named `first` and `second`. Notice the statement,

---

```
String joinedString = first.concat(second);
```

Here, we the `concat()` method joins `first` and `second` and assigns it to the `joinedString` variable.

We can also join two strings using the `+` operator in Java. To learn more, visit Java String concat().

---

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
```

In the above example, we have created 3 strings named `first`, `second`, and `third`. Here, we are using the `equal()` method to check if one string is equal to another.

The `equals()` method checks the content of strings while comparing them. To learn more, visit Java String equals().

> **Note**: We can also compare two strings using the `==` operator in Java. However, this approach is different than the `equals()` method. To learn more, visit Java String == vs equals().

Methods of Java String

Besides those mentioned above, there are various string methods present in Java. Here are some of those methods:

Methods	Description
substring()	returns the substring of the string
replace()	replaces the specified old character with the specified new character
charAt()	returns the character present in the specified location
getBytes()	converts the string to an array of bytes
indexOf()	returns the position of the specified character in the string
compareTo()	compares two strings in the dictionary order
trim()	removes any leading and trailing whitespaces
format()	returns a formatted string
split()	breaks the string into an array of strings

---

toLowerCase()	converts the string to lowercase
toUpperCase()	converts the string to uppercase
valueOf()	returns the string representation of the specified argument
toCharArray()	converts the string to a `char` array

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.";
```

Now escape characters tell the compiler to escape **double quotes** and read the whole text.

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. To learn more about constructor, visit Java Constructor.

> **Note**: The `String` class provides various other constructors to create strings. To learn more, visit Java String (official Java documentation).

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.

While creating strings using string literals, the value of the string is directly provided. 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 points to the existing string.
- **If the string doesn't exist**, the new string is created.

However, **while creating strings using the new keyword**, the value of the string is not directly provided. Hence the new string is created all the time.
