Java String

In [Java](https://www.javatpoint.com/java-tutorial) , string is basically an object that represents sequence of char values. An [array](https://www.javatpoint.com/array-in-java)

of characters works same as Java string. For example:

1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

is same as:

1. String s="javatpoint";

**Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

Thejava.lang.Stringclass implements *Serializable*, *Comparable* and *CharSequence*  [interfaces](https://www.javatpoint.com/interface-in-java)

.istory of Java



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String, [StringBuffer](https://www.javatpoint.com/StringBuffer-class) and [StringBuilder](https://www.javatpoint.com/StringBuilder-class) classes implement it. It means, we can create strings in Java by using these three classes.



The Java String is immutable which means it cannot be changed. Whenever we change any string, a new instance is created. For mutable strings, you can use StringBuffer and StringBuilder classes.

We will discuss immutable string later. Let's first understand what String in Java is and how to create the String object.

What is String in Java?

Generally, String is a sequence of characters. But in Java, string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.

How to create a string object?

There are two ways to create String object:

1. By string literal
2. By new keyword

1) String Literal

Java String literal is created by using double quotes. For Example:

String s="welcome";

Each time you create a string literal, the JVM checks the "string constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned. If the string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

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



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool that is why 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.

Note: String objects are stored in a special memory area known as the "string constant pool".

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).

2) By new keyword

1. String s=**new** String("Welcome");//creates two objects and one reference

 variable

In such case, [JVM](https://www.javatpoint.com/jvm-java-virtual-machine) 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 a heap (non-pool).

Java String Example

**StringExample.java**

**public** **class** StringExample

{

**public** **static** **void** main(String args[])

{

String s1="java";//creating string by Java string literal

**char** ch[]={'s','t','r','i','n','g','s'};

String s2=**new** String(ch);//converting char array to string

String s3=**new** String("example");//creating Java string by new keyword

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}

}

**Output:**

java

strings

example

The above code, converts a ***char*** array into a **String** object. And displays the String objects ***s1, s2***, and ***s3*** on console using ***println()*** method.

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | [char charAt(int index)](https://www.javatpoint.com/java-string-charat) | It returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | It returns string length |
| 3 | [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns a formatted string. |
| 4 | [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | It returns formatted string with given locale. |
| 5 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index. |
| 6 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | It returns substring for given begin index and end index. |
| 7 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | It returns true or false after matching the sequence of char value. |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | It returns a joined string. |
| 10 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | It checks the equality of string with the given object. |
| 11 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | It checks if string is empty. |
| 12 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | It concatenates the specified string. |
| 13 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified char value. |
| 14 | [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | It replaces all occurrences of the specified CharSequence. |
| 15 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | It compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex. |
| 17 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | It returns a split string matching regex and limit. |
| 18 | [String intern()](https://www.javatpoint.com/java-string-intern) | It returns an interned string. |
| 19 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index. |
| 20 | [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified char value index starting with given index. |
| 21 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index. |
| 22 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | It returns the specified substring index starting with given index. |
| 23 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase. |
| 24 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | It returns a string in lowercase using specified locale. |
| 25 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase. |
| 26 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | It returns a string in uppercase using specified locale. |
| 27 | [String trim()](https://www.javatpoint.com/java-string-trim) | It removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | It converts given type into string. It is an overloaded method. |

What will we learn in String Handling?

* Concept of String
* Immutable String
* String Comparison
* String Concatenation
* Concept of Substring

# Immutable String in Java

A String is an unavoidable type of variable while writing any application program. String references are used to store various attributes like username, password, etc. In Java, **String objects are immutable**. Immutable simply means unmodifiable or unchangeable.

Once String object is created its data or state can't be changed but a new String object is created.

Let's try to understand the concept of immutability by the example given below:

**Testimmutablestring.java**

33.2M

rial

**class** Testimmutablestring

{

**public** **static** **void** main(String args[])

{

  String s="Sachin";

   s.concat(" Tendulkar");

System.out.println(s);

}

}

**Output:**

Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with Sachin Tendulkar. That is why String is known as immutable.



As you can see in the above figure that two objects are created but **s** reference variable still refers to "Sachin" not to "Sachin Tendulkar".

But if we explicitly assign it to the reference variable, it will refer to "Sachin Tendulkar" object.

For example:

**Testimmutablestring1.java**

**class** Testimmutablestring1

{

 **public** **static** **void** main(String args[])

{

String s="Sachin";

 s=s.concat(" Tendulkar");

  System.out.println(s);

  }

}

**Output:**

Sachin Tendulkar

In such a case, s points to the "Sachin Tendulkar". Please notice that still Sachin object is not modified.

### Why String objects are immutable in Java?

As 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.

Following are some features of String which makes String objects immutable.

**1. ClassLoader:**

A ClassLoader in Java uses a String object as an argument. Consider, if the String object is modifiable, the value might be changed and the class that is supposed to be loaded might be different.

To avoid this kind of misinterpretation, String is immutable.

**2. Thread Safe:**

As the String object is immutable we don't have to take care of the synchronization that is required while sharing an object across multiple threads.

**3. Security:**

As we have seen in class loading, immutable String objects avoid further errors by loading the correct class. This leads to making the application program more secure. Consider an example of banking software. The username and password cannot be modified by any intruder because String objects are immutable. This can make the application program more secure.

**4. Heap Space:**

The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space efficiently.

### Why String class is Final in Java?

The reason behind the String class being final is because no one can override the methods of the String class. So that it can provide the same features to the new String objects as well as to the old ones.

# Java String compare

We can compare String in Java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare String in Java:

1. By Using equals() Method
2. By Using == Operator
3. By compareTo() Method

## 1) By Using equals() Method

The String class equals() method compares the original content of the string. It compares values of string for equality. String class provides the following two methods:

40.6M

* **public boolean equals(Object another)** compares this string to the specified object.
* **public boolean equalsIgnoreCase(String another)** compares this string to another string, ignoring case.

**Teststringcomparison1.java**

**class** Teststringcomparison1

{

**public** **static** **void** main(String args[])

{

  String s1="Sachin";

 String s2="Sachin";

String s3=**new** String("Sachin");

String s4="Saurav";

System.out.println(s1.equals(s2));//true

System.out.println(s1.equals(s3));//true

  System.out.println(s1.equals(s4));//false

 }

}

**Output:**

true

true

false

In the above code, two strings are compared using **equals()** method of **String** class. And the result is printed as boolean values, **true** or **false**.

**Teststringcomparison2.java**

**class** Teststringcomparison2

{

**public** **static** **void** main(String args[])

{

  String s1="Sachin";

  String s2="SACHIN";

     System.out.println(s1.equals(s2));//false

System.out.println(s1.equalsIgnoreCase(s2));//true

 }

}

**Output:**

false

true

In the above program, the methods of **String** class are used. The **equals()** method returns true if String objects are matching and both strings are of same case. **equalsIgnoreCase()** returns true regardless of cases of strings.

## 2) By Using == operator

The == operator compares references not values.

**Teststringcomparison3.java**

**class** Teststringcomparison3

{

**public** **static** **void** main(String args[])

{

  String s1="Sachin";

String s2="Sachin";

    String s3=**new** String("Sachin");

System.out.println(s1==s2);

    System.out.println(s1==s3);

  }

}

**Output:**

true

false

Substring in Java

A part of String is called **substring**. In other words, substring is a subset of another String. Java String class provides the built-in substring() method that extract a substring from the given string by using the index values passed as an argument. In case of substring() method startIndex is inclusive and endIndex is exclusive.

Suppose the string is "**computer**", then the substring will be com, compu, ter, etc.

Note: Index starts from 0.

You can get substring from the given String object by one of the two methods:

1. **public String substring(int startIndex):**

This method returns new String object containing the substring of the given string from specified startIndex (inclusive). The method throws an IndexOutOfBoundException when the startIndex is larger than the length of String or less than zero.

1. **public String substring(int startIndex, int endIndex):**

This method returns new String object containing the substring of the given string from specified startIndex to endIndex. The method throws an IndexOutOfBoundException when the startIndex is less than zero or startIndex is greater than endIndex or endIndex is greater than length of String.

In case of String:SQL CREATE TABLE

* **startIndex:** inclusive
* **endIndex:** exclusive

Let's understand the startIndex and endIndex by the code given below.

1. String s="hello";
2. System.out.println(s.substring(0,2)); //returns he  as a substring

In the above substring, 0 points the first letter and 2 points the second letter i.e., e (because end index is exclusive).

Example of Java substring() method

**TestSubstring.java**

**public** **class** TestSubstring

{

 **public** **static** **void** main(String args[])

{

  String s="SachinTendulkar";

 System.out.println("Original String: " + s);

  System.out.println("Substring starting from index 6: " +s.substring(6));

 System.out.println("Substring starting from index 0 to 6: "+s.substring(0,6));

 }

}

**Output:**

Original String: SachinTendulkar

Substring starting from index 6: Tendulkar

Substring starting from index 0 to 6: Sachin

## 3) String compare by compareTo() method

The above code, demonstrates the use of **==** operator used for comparing two **String** objects.

## 3) By Using compareTo() method

The String class compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two String objects. If:

* **s1 == s2** : The method returns 0.
* **s1 > s2** : The method returns a positive value.
* **s1 < s2** : The method returns a negative value.

**Teststringcomparison4.java**

**class** Teststringcomparison4

{

 **public** **static** **void** main(String args[])

{

    String s1="Sachin";

    String s2="Sachin";

    String s3="Ratan";

    System.out.println(s1.compareTo(s2));//0

    System.out.println(s1.compareTo(s3));//1(because s1>s3)

    System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )

 }

}

**Output:**

0

1

-1

# String Concatenation in Java

In Java, String concatenation forms a new String that is the combination of multiple strings. There are two ways to concatenate strings in Java:

1. By + (String concatenation) operator
2. By concat() method

## 1) String Concatenation by + (String concatenation) operator

Java String concatenation operator (+) is used to add strings. For Example:

**TestStringConcatenation1.java**

**class** TestStringConcatenation1

{

  **public** **static** **void** main(String args[])

{

    String s="Sachin"+" Tendulkar";

    System.out.println(s);//Sachin Tendulkar

 }

}

**Output:**

39M

OPs Concepts in Java

Sachin Tendulkar

The String concatenation operator can concatenate not only String but primitive values also. For Example:

**TestStringConcatenation2.java**

1. **class** TestStringConcatenation2{
2. **public** **static** **void** main(String args[]){
3. String s=50+30+"Sachin"+40+40;
4. System.out.println(s);//80Sachin4040
5. }
6. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation2)

**Output:**

80Sachin4040

#### Note: After a string literal, all the + will be treated as string concatenation operator.

###

### String Concatenation by concat() method

The String concat() method concatenates the specified string to the end of current string. Syntax:

**public** String concat(String another)

Let's see the example of String concat() method.

**TestStringConcatenation3.java**

**class** TestStringConcatenation3

{

  **public** **static** **void** main(String args[])

{

     String s1="Sachin ";

     String s2="Tendulkar";

    String s3=s1.concat(s2);

System.out.println(s3);//Sachin Tendulkar

   }

}

**Output:**

Sachin Tendulkar

The above Java program, concatenates two String objects **s1** and **s2** using **concat()**

 method and stores the result into **s3** object.