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.

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

.



CharSequence Interface

The CharSequence interface is used to represent the sequence of characters. String,

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.

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:

1. 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 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. |

# 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**

1. **class** Testimmutablestring{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

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**

1. **class** Testimmutablestring1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin";
4. s=s.concat(" Tendulkar");
5. System.out.println(s);
6. }
7. }

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.

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.

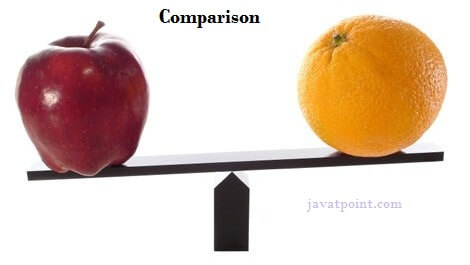
**4. Heap Space:**

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:

* **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.

## 2) By Using == operator

The == operator compares references not values.

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.

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.

1. **class** TestStringConcatenation1{
2. **public** **static** **void** main(String args[]){
3. String s="Sachin"+" Tendulkar";
4. System.out.println(s);//Sachin Tendulkar
5. }
6. }

The **Java compiler transforms** above code to this:

1. String s=(**new** StringBuilder()).append("Sachin").append(" Tendulkar).toString();

In Java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and it's append method. String concatenation operator produces a new String by appending the second operand onto the end of the first operand. The String concatenation operator can concatenate not only String but primitive values also.

### 2) String Concatenation by concat() method

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

1. **public** String concat(String another)

There are some other possible ways to concatenate Strings in Java,

### 1. String concatenation using StringBuilder class

StringBuilder is class provides append() method to perform concatenation operation. The append() method accepts arguments of different types like Objects, StringBuilder, int, char, CharSequence, boolean, float, double. StringBuilder is the most popular and fastet way to concatenate strings in Java. It is mutable class which means values stored in StringBuilder objects can be updated or changed.

### 2. String concatenation using format() method

String.format() method allows to concatenate multiple strings using format specifier like %s followed by the string values or objects.

### 3. String concatenation using String.join() method (Java Version 8+)

The String.join() method is available in Java version 8 and all the above versions. String.join() method accepts arguments first a separator and an array of String objects.

### 4. String concatenation using StringJoiner class (Java Version 8+)

StringJoiner class has all the functionalities of String.join() method. In advance its constructor can also accept optional arguments, prefix and suffix.

### 5. String concatenation using Collectors.joining() method (Java (Java Version 8+)

The Collectors class in Java 8 offers joining() method that concatenates the input elements in a similar order as they occur.

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.
2. **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:

* **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., (because end index is exclusive).

Java String Methods

# Java toString() Method

If you want to represent any object as a string, **toString() method** comes into existence.

The toString() method returns the String representation of the object.

If you print any object, Java compiler internally invokes the toString() method on the object. So overriding the toString() method, returns the desired output, it can be the state of an object etc. depending on your implementation.

### Advantage of Java toString() method

By overriding the toString() method of the Object class, we can return values of the object, so we don't need to write much code.

# Java String charAt()

The **Java String class charAt()** method returns a char value at the given index number.

The index number starts from 0 and goes to n-1, where n is the length of the string. It returns **StringIndexOutOfBoundsException,** if the given index number is greater than or equal to this string length or a negative number.

### Syntax

1. **public** **char** charAt(**int** index)

The method accepts **index** as a parameter. The starting index is 0. It returns a character at a specific index position in a string. It throws **StringIndexOutOfBoundsException** if the index is a negative value or greater than this string length.

Java String contains()

The **Java String class contains()** method searches the sequence of characters in this string. It returns *true* if the sequence of char values is found in this string otherwise returns *false*.

Signature

The signature of string contains() method is given below:

1. **public** **boolean** contains(CharSequence sequence)

Parameter

**sequence** : specifies the sequence of characters to be searched.

Returns

**true** if the sequence of char value exists, otherwise **false**.Features of Java - Javatpoint

Exception

**NullPointerException** : if the sequence is null.

Internal implementation

1. **public** **boolean** contains(CharSequence s) {
2. **return** indexOf(s.toString()) > -1;
3. }

Here, the conversion of CharSequence takes place into String. After that, the indexOf() method is invoked. The method indexOf() either returns 0 or a number greater than 0 in case the searched string is found.

However, when the searched string is not found, the indexOf() method returns -1. Therefore, after execution, the contains() method returns true when the indexOf() method returns a non-negative value (when the searched string is found); otherwise, the method returns false.

Limitations of the Contains() method

Following are some limitations of the contains() method:

* The contains() method should not be used to search for a character in a string. Doing so results in an error.
* The contains() method only checks for the presence or absence of a string in another string. It never reveals at which index the searched index is found. Because of these limitations, it is better to use the indexOf() method instead of the contains() method.

Java String isEmpty()

The **Java String class isEmpty()** method checks if the input string is empty or not. Note that here empty means the number of characters contained in a string is zero.

Signature

The signature or syntax of string isEmpty() method is given below:

1. **public** **boolean** isEmpty()

Returns

true if length is 0 otherwise false.

Empty Vs. Null Strings

Earlier in this tutorial, we have discussed that the empty strings contain zero characters. However, the same is true for a null string too. A null string is a string that has no value.

1. String str = ""; // empty string
2. String str1 = **null**; // null string. It is also not containing any characters.

The isEmpty() method is not fit for checking the null strings. The following example shows the same.

Java String join()

The **Java String class join()** method returns a string joined with a given delimiter. In the String join() method, the delimiter is copied for each element. The join() method is included in the Java string since JDK 1.8.

There are two types of join() methods in the Java String class.

Signature

The signature or syntax of the join() method is given below:

1. **public** **static** String join(CharSequence delimiter, CharSequence... elements)
2. and
3. **public** **static** String join(CharSequence delimiter, Iterable<? **extends** CharSequence> elements)

Parameters

**delimiter** : char value to be added with each element

**elements** : char value to be attached with delimiter

Returns

joined string with delimiter

Exception Throws

**NullPointerException** if element or delimiter is null.

Java String length()

The **Java String class length()** method finds the length of a string. The length of the Java string is the same as the Unicode code units of the string.

Signature

The signature of the string length() method is given below:

1. **public** **int** length()

Specified by

CharSequence interface

Returns

Length of characters. In other words, the total number of characters present in the string.839Java Try Catch

Internal implementation

1. **public** **int** length() {
2. **return** value.length;
3. }

The String class internally uses a char[] array to store the characters. The length variable of the array is used to find the total number of elements present in the array. Since the Java String class uses this char[] array internally; therefore, the length variable can not be exposed to the outside world. Hence, the Java developers created the length() method, the exposes the value of the length variable. One can also think of the length() method as the getter() method, that provides a value of the class field to the user. The internal implementation clearly depicts that the length() method returns the value of then the length variable.

Java String toCharArray()

The **java string toCharArray()** method converts this string into character array. It returns a newly created character array, its length is similar to this string and its contents are initialized with the characters of this string.

Internal implementation

1. **public** **char**[] toCharArray() {
2. // Cannot use Arrays.copyOf because of class initialization order issues
3. **char** result[] = **new** **char**[value.length];
4. System.arraycopy(value, 0, result, 0, value.length);
5. **return** result;
6. }

Signature

The signature or syntax of string toCharArray() method is given below:

1. **public** **char**[] toCharArray()

Returns

character array

# Java String toUpperCase()

The **java string toUpperCase()** method returns the string in uppercase letter. In other words, it converts all characters of the string into upper case letter.

The toUpperCase() method works same as toUpperCase(Locale.getDefault()) method. It internally uses the default locale.

Signature

There are two variant of toUpperCase() method. The signature or syntax of string toUpperCase() method is given below:

1. **public** String toUpperCase()
2. **public** String toUpperCase(Locale locale)

The second method variant of toUpperCase(), converts all the characters into uppercase using the rules of given Locale.

Returns

string in uppercase letter.

# Java String toLowerCase()

The **java string toLowerCase()** method returns the string in lowercase letter. In other words, it converts all characters of the string into lower case letter.

The toLowerCase() method works same as toLowerCase(Locale.getDefault()) method. It internally uses the default locale.

Signature

There are two variant of toLowerCase() method. The signature or syntax of string toLowerCase() method is given below:

1. **public** String toLowerCase()
2. **public** String toLowerCase(Locale locale)

The second method variant of toLowerCase(), converts all the characters into lowercase using the rules of given Locale. Tricky Program 16 - Autoboxing, Inheritance and ng

Returns

string in lowercase letter.

Java String trim()

The **Java String class trim()** method eliminates leading and trailing spaces. The Unicode value of space character is '\u0020'. The trim() method in Java string checks this Unicode value before and after the string, if it exists then the method removes the spaces and returns the omitted string.

The string trim() method doesn't omit middle spaces.

Signature

The signature or syntax of the String class trim() method is given below:

1. **public** String trim()

Returns

string with omitted leading and trailing spaces

Java String replace()

The **Java String class replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

Since JDK 1.5, a new replace() method is introduced that allows us to replace a sequence of char values.

Signature

There are two types of replace() methods in Java String class.

1. **public** String replace(**char** oldChar, **char** newChar)
2. **public** String replace(CharSequence target, CharSequence replacement)

The second replace() method is added since JDK 1.5.

58.3M

944Hello Java Program for Beginners

Parameters

**oldChar** : old character

**newChar** : new character

**target** : target sequence of characters

**replacement** : replacement sequence of characters

Returns

replaced string

Exception Throws

NullPointerException: if the replacement or target is equal to null.

Java String replaceAll()

The Java String class replaceAll() method returns a string replacing all the sequence of characters matching regex and replacement string.

Signature

1. **public** String replaceAll(String regex, String replacement)

Parameters

**regex** : regular expression

**replacement** : replacement sequence of characters

Returns

replaced string

7.5MChina's Gaming Industry Emerges From Devastating 9-Month Freeze

Exception Throws

PatternSyntaxException: if the syntax of the regular expression is not valid.

Internal implementation

1. **public** String replaceAll(String regex, String replacement) {
2. **return** Pattern.compile(regex).matcher(**this**).replaceAll(replacement);
3. }