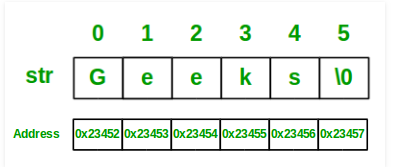
**String in JAVA**

Strings are defined as an array of characters.

The difference between a character array and a string is the string is terminated with a special character ‘\0’.

**Example:**

String str = "Geeks";



**Memory allotment of String**

Whenever a String Object is created,

two objects will be created-

one in the Heap Area and

one in the String constant pool

and the String object reference always points to heap area object.

**For example:**

String str = "Geeks";

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. For example:

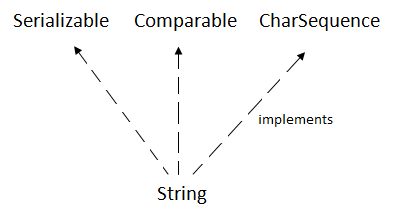
1. **char**[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=**new** String(ch);

is same as:

1. String s="javatpoint";

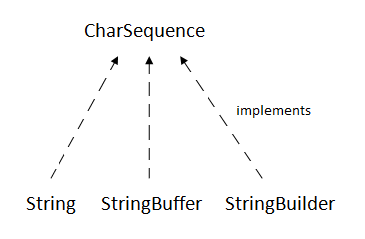
**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



## CharSequence Interface

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

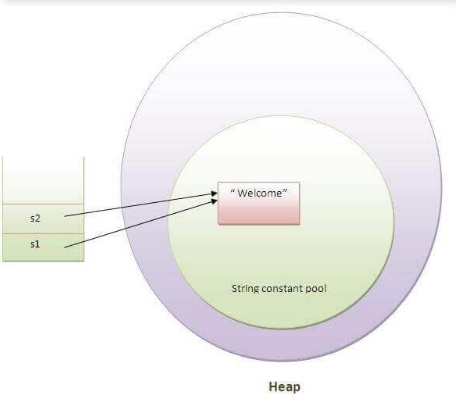
1) String Literal

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 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

1. **public** **class** StringExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java";//creating string by java string literal
4. **char** ch[]={'s','t','r','i','n','g','s'};
5. String s2=**new** String(ch);//converting char array to string
6. String s3=**new** String("example");//creating java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}

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

java

strings

example

# Immutable String in Java

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.

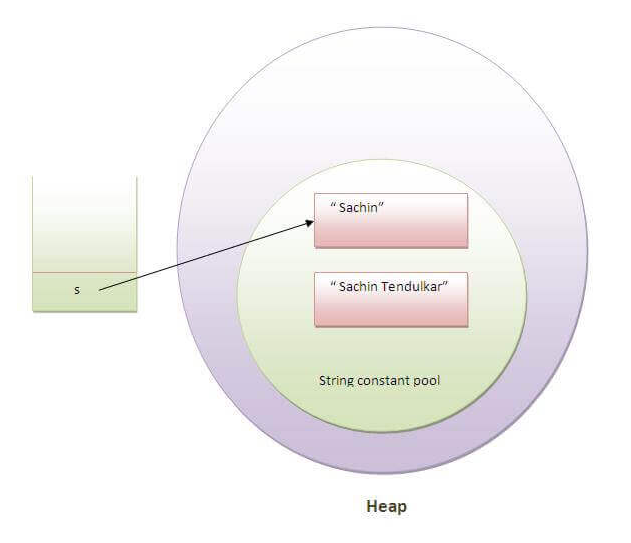
Let's try to understand the immutability concept by the example given below:

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. }

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

Output:Sachin

Now it can be understood by the diagram given below. Here Sachin is not changed but a new object is created with sachintendulkar. 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 explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

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. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testimmutablestring1)

Output:Sachin Tendulkar

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

Why string objects are immutable in java?

|  |
| --- |
| Because java uses the concept of string literal.Suppose there are 5 reference variables,all referes to one object "sachin".If one reference variable changes the value of the object, it will be affected to all the reference variables. That is why string objects are immutable in java. |

# 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 equals() method
2. By = = operator
3. By compareTo() method

## 1) String compare by equals() method

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

1. **class** Teststringcomparison1{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. String s4="Saurav";
7. System.out.println(s1.equals(s2));//true
8. System.out.println(s1.equals(s3));//true
9. System.out.println(s1.equals(s4));//false
10. }
11. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Teststringcomparison1)

Output:true

true

false

1. **class** Teststringcomparison2{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="SACHIN";
6. System.out.println(s1.equals(s2));//false
7. System.out.println(s1.equalsIgnoreCase(s2));//true
8. }
9. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Teststringcomparison2)

Output:

false

true

## 2) String compare by == operator

The = = operator compares references not values.

1. **class** Teststringcomparison3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. System.out.println(s1==s2);//true (because both refer to same instance)
7. System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
8. }
9. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Teststringcomparison3)

Output:true

false

## 3) String compare by compareTo() method

The String 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 variables. If:

* **s1 == s2** :0
* **s1 > s2**  :positive value
* **s1 < s2**  :negative value

1. **class** Teststringcomparison4{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3="Ratan";
6. System.out.println(s1.compareTo(s2));//0
7. System.out.println(s1.compareTo(s3));//1(because s1>s3)
8. System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9. }
10. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Teststringcomparison4)

Output:0

1

-1

# Java String equalsIgnoreCase()

The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

### Signature

1. **public** **boolean** equalsIgnoreCase(String str)

### Parameter

**str** : another string i.e. compared with this string.

### Returns

It returns **true** if characters of both strings are equal ignoring case otherwise **false**.

## Java String equalsIgnoreCase() method example

1. **public** **class** EqualsIgnoreCaseExample{
2. **public** **static** **void** main(String args[]){
3. String s1="javatpoint";
4. String s2="javatpoint";
5. String s3="JAVATPOINT";
6. String s4="python";
7. System.out.println(s1.equalsIgnoreCase(s2));//true because content and case both are same
8. System.out.println(s1.equalsIgnoreCase(s3));//true because case is ignored
9. System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same
10. }}

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

true

true

false

# String Concatenation in Java

n java, string concatenation forms a new string that is the combination of multiple strings. There are two ways to concat string 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:

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

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation1)

Output:Sachin Tendulkar

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 its 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 concat not only string but primitive values also. For Example:

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**](http://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation2)

80Sachin4040

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

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)

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

1. **class** TestStringConcatenation3{
2. **public** **static** **void** main(String args[]){
3. String s1="Sachin ";
4. String s2="Tendulkar";
5. String s3=s1.concat(s2);
6. System.out.println(s3);//Sachin Tendulkar
7. }
8. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestStringConcatenation3)

Sachin Tendulkar

# Substring in Java

A part of string is called **substring**. In other words, substring is a subset of another string. In case of substring startIndex is inclusive and endIndex is exclusive.

#### 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).
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.

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));//he

In the above substring, 0 points to h but 2 points to e (because end index is exclusive).

## Example of java substring

1. **public** **class** TestSubstring{
2. **public** **static** **void** main(String args[]){
3. String s="SachinTendulkar";
4. System.out.println(s.substring(6));//Tendulkar
5. System.out.println(s.substring(0,6));//Sachin
6. }
7. }

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

Tendulkar

Sachin

# Java String class methods

Java String toUpperCase() and toLowerCase() method

The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

1. String s="Sachin";
2. System.out.println(s.toUpperCase());//SACHIN
3. System.out.println(s.toLowerCase());//sachin
4. System.out.println(s);//Sachin(no change in original)

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

SACHIN

sachin

Sachin

Java String trim() method

The string trim() method eliminates white spaces before and after string.

1. String s="  Sachin  ";
2. System.out.println(s);//  Sachin
3. System.out.println(s.trim());//Sachin

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass1)

Sachin

Sachin

Java String startsWith() and endsWith() method

1. String s="Sachin";
2. System.out.println(s.startsWith("Sa"));//true
3. System.out.println(s.endsWith("n"));//true

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass2)

true

true

Java String charAt() method

The string charAt() method returns a character at specified index.

1. String s="Sachin";
2. System.out.println(s.charAt(0));//S
3. System.out.println(s.charAt(3));//h

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass3)

S

h

## Java String charAt() Example 4

Let's see an example where we are accessing all the elements present at odd index.

1. **public** **class** CharAtExample4 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Welcome to Javatpoint portal";
4. **for** (**int** i=0; i<=str.length()-1; i++) {
5. **if**(i%2!=0) {
6. System.out.println("Char at "+i+" place "+str.charAt(i));
7. }
8. }
9. }
10. }

Output:

Char at 1 place e

Char at 3 place c

Char at 5 place m

Char at 7 place

Char at 9 place o

Char at 11 place J

Char at 13 place v

Char at 15 place t

Char at 17 place o

Char at 19 place n

Char at 21 place

Char at 23 place o

Char at 25 place t

Char at 27 place l

## Java String charAt() Example 5

Let's see an example where we are counting frequency of a character in the string.

1. **public** **class** CharAtExample5 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Welcome to Javatpoint portal";
4. **int** count = 0;
5. **for** (**int** i=0; i<=str.length()-1; i++) {
6. **if**(str.charAt(i) == 't') {
7. count++;
8. }
9. }
10. System.out.println("Frequency of t is: "+count);
11. }
12. }

Output:

Frequency of t is: 4

Java String length() method

The string length() method returns length of the string.

1. String s="Sachin";
2. System.out.println(s.length());//6

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass4)

6

Java String intern() method

A pool of strings, initially empty, is maintained privately by the class String.

When the intern method is invoked, if the pool already contains a string equal to this String object as determined by the equals(Object) method, then the string from the pool is returned. Otherwise, this String object is added to the pool and a reference to this String object is returned.

1. String s=**new** String("Sachin");
2. String s2=s.intern();
3. System.out.println(s2);//Sachin

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Testmethodofstringclass5)

Sachin

Java String valueOf() method

The string valueOf() method coverts given type such as int, long, float, double, boolean, char and char array into string.

1. **int** a=10;
2. String s=String.valueOf(a);
3. System.out.println(s+10);

Output:

1010

Java String replace() method

The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

1. String s1="Java is a programming language. Java is a platform. Java is an Island.";
2. String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"
3. System.out.println(replaceString);

Output:

Kava is a programming language. Kava is a platform. Kava is an Island.

## ava String replace(char old, char new) method example

1. **public** **class** ReplaceExample1{
2. **public** **static** **void** main(String args[]){
3. String s1="javatpoint is a very good website";
4. String replaceString=s1.replace('a','e');//replaces all occurrences of 'a' to 'e'
5. System.out.println(replaceString);
6. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ReplaceExample1)

jevetpoint is e very good website

## Java String replace(CharSequence target, CharSequence replacement) method example

1. **public** **class** ReplaceExample2{
2. **public** **static** **void** main(String args[]){
3. String s1="my name is khan my name is java";
4. String replaceString=s1.replace("is","was");//replaces all occurrences of "is" to "was"
5. System.out.println(replaceString);
6. }}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=ReplaceExample2)

my name was khan my name was java

## Java String replace() Method Example 3

1. **public** **class** ReplaceExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str = "oooooo-hhhh-oooooo";
4. String rs = str.replace("h","s"); // Replace 'h' with 's'
5. System.out.println(rs);
6. rs = rs.replace("s","h"); // Replace 's' with 'h'
7. System.out.println(rs);
8. }
9. }

oooooo-ssss-oooooo

oooooo-hhhh-oooooo

# Java String compareTo()

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

It compares strings on the basis of Unicode value of each character in the strings.

If first string is lexicographically greater than second string, it returns positive number (difference of character value). If first string is less than second string lexicographically, it returns negative number and if first string is lexicographically equal to second string, it returns 0.

1. **if** s1 > s2, it returns positive number
2. **if** s1 < s2, it returns negative number
3. **if** s1 == s2, it returns 0

## Java String compareTo() method example

1. **public** **class** CompareToExample{
2. **public** **static** **void** main(String args[]){
3. String s1="hello";
4. String s2="hello";
5. String s3="meklo";
6. String s4="hemlo";
7. String s5="flag";
8. System.out.println(s1.compareTo(s2));//0 because both are equal
9. System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
10. System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
11. System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
12. }}

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

Output:

0

-5

-1

2

## Java String indexOf() method example

1. **public** **class** IndexOfExample{
2. **public** **static** **void** main(String args[]){
3. String s1="this is index of example";
4. //passing substring
5. **int** index1=s1.indexOf("is");//returns the index of is substring
6. **int** index2=s1.indexOf("index");//returns the index of index substring
7. System.out.println(index1+"  "+index2);//2 8
9. //passing substring with from index
10. **int** index3=s1.indexOf("is",4);//returns the index of is substring after 4th index
11. System.out.println(index3);//5 i.e. the index of another is
13. //passing char value
14. **int** index4=s1.indexOf('s');//returns the index of s char value
15. System.out.println(index4);//3
16. }}

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

2 8

5

3

## Java String indexOf(String substring) Method Example

This method takes substring as an argument and returns index of first character of the substring.

1. **public** **class** IndexOfExample2 {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "This is indexOf method";
4. // Passing Substring
5. **int** index = s1.indexOf("method"); //Returns the index of this substring
6. System.out.println("index of substring "+index);
7. }
9. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=IndexOfExample2)

index of substring 16

## Java String indexOf(String substring, int fromIndex) Method Example

This method takes substring and index as arguments and returns index of first character occured after the given fromIndex.

1. **public** **class** IndexOfExample3 {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "This is indexOf method";
4. // Passing substring and index
5. **int** index = s1.indexOf("method", 10); //Returns the index of this substring
6. System.out.println("index of substring "+index);
7. index = s1.indexOf("method", 20); // It returns -1 if substring does not found
8. System.out.println("index of substring "+index);
9. }
10. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=IndexOfExample3)

index of substring 16

index of substring -1

## Java String indexOf(int char, int fromIndex) Method Example

This method takes char and index as arguments and returns index of first character occured after the given fromIndex.

1. **public** **class** IndexOfExample4 {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "This is indexOf method";
4. // Passing char and index from
5. **int** index = s1.indexOf('e', 12); //Returns the index of this char
6. System.out.println("index of char "+index);
7. }
8. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=IndexOfExample4)

index of char 17

# Java String valueOf()

### Signature

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

1. **public** **static** String valueOf(**boolean** b)
2. **public** **static** String valueOf(**char** c)
3. **public** **static** String valueOf(**char**[] c)
4. **public** **static** String valueOf(**int** i)
5. **public** **static** String valueOf(**long** l)
6. **public** **static** String valueOf(**float** f)
7. **public** **static** String valueOf(**double** d)
8. **public** **static** String valueOf(Object o)

### Returns

string representation of given value

## Java String valueOf() method example

1. **public** **class** StringValueOfExample{
2. **public** **static** **void** main(String args[]){
3. **int** value=30;
4. String s1=String.valueOf(value);
5. System.out.println(s1+10);//concatenating string with 10
6. }}

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

Output:

3010

## Java String valueOf(boolean bol) Method Example

This is a boolean version of overloaded valueOf() method. It takes boolean value and returns a string. Let's see an example.

1. **public** **class** StringValueOfExample2 {
2. **public** **static** **void** main(String[] args) {
3. // Boolean to String
4. **boolean** bol = **true**;
5. **boolean** bol2 = **false**;
6. String s1 = String.valueOf(bol);
7. String s2 = String.valueOf(bol2);
8. System.out.println(s1);
9. System.out.println(s2);
10. }
11. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringValueOfExample2)

Output:

true

false

## Java String valueOf(char ch) Method Example

This is a char version of overloaded valueOf() method. It takes char value and returns a string. Let's see an example.

1. **public** **class** StringValueOfExample3 {
2. **public** **static** **void** main(String[] args) {
3. // char to String
4. **char** ch1 = 'A';
5. **char** ch2 = 'B';
6. String s1 = String.valueOf(ch1);
7. String s2 = String.valueOf(ch2);
8. System.out.println(s1);
9. System.out.println(s2);
10. }
11. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringValueOfExample3)

Output:

A

B

## Java String valueOf(float f) and valueOf(double d)

This is a float version of overloaded valueOf() method. It takes float value and returns a string. Let's see an example.

1. **public** **class** StringValueOfExample4 {
2. **public** **static** **void** main(String[] args) {
3. // Float and Double to String
4. **float** f  = 10.05f;
5. **double** d = 10.02;
6. String s1 = String.valueOf(f);
7. String s2 = String.valueOf(d);
8. System.out.println(s1);
9. System.out.println(s2);
10. }
11. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringValueOfExample4)

Output:

10.05

10.02

## Java String valueOf() Complete Examples

Let's see an example where we are converting all primitives and objects into strings.

1. **public** **class** StringValueOfExample5 {
2. **public** **static** **void** main(String[] args) {
3. **boolean** b1=**true**;
4. **byte** b2=11;
5. **short** sh = 12;
6. **int** i = 13;
7. **long** l = 14L;
8. **float** f = 15.5f;
9. **double** d = 16.5d;
10. **char** chr[]={'j','a','v','a'};
11. StringValueOfExample5 obj=**new** StringValueOfExample5();
12. String s1 = String.valueOf(b1);
13. String s2 = String.valueOf(b2);
14. String s3 = String.valueOf(sh);
15. String s4 = String.valueOf(i);
16. String s5 = String.valueOf(l);
17. String s6 = String.valueOf(f);
18. String s7 = String.valueOf(d);
19. String s8 = String.valueOf(chr);
20. String s9 = String.valueOf(obj);
21. System.out.println(s1);
22. System.out.println(s2);
23. System.out.println(s3);
24. System.out.println(s4);
25. System.out.println(s5);
26. System.out.println(s6);
27. System.out.println(s7);
28. System.out.println(s8);
29. System.out.println(s9);
30. }
31. }

[**Test it Now**](https://compiler.javatpoint.com/opr/test.jsp?filename=StringValueOfExample5)

Output:

true

11

12

13

14

15.5

16.5

java

StringValueOfExample5@2a139a55

# Java String substring()

The **java string substring()** method returns a part of the string.

We pass begin index and end index number position in the java substring method where start index is inclusive and end index is exclusive. In other words, start index starts from 0 whereas end index starts from 1.

There are two types of substring methods in java string.

### Internal implementation

1. **public** String substring(**int** beginIndex) {
2. **if** (beginIndex < 0) {
3. **throw** **new** StringIndexOutOfBoundsException(beginIndex);
4. }
5. **int** subLen = value.length - beginIndex;
6. **if** (subLen < 0) {
7. **throw** **new** StringIndexOutOfBoundsException(subLen);
8. }
9. **return** (beginIndex == 0) ? **this** : **new** String(value, beginIndex, subLen);
10. }

### Signature

1. **public** String substring(**int** startIndex)
2. and
3. **public** String substring(**int** startIndex, **int** endIndex)

If you don't specify endIndex, java substring() method will return all the characters from startIndex.

### Parameters

**startIndex** : starting index is inclusive

**endIndex** : ending index is exclusive

### Returns

specified string

### Throws

**StringIndexOutOfBoundsException** if start index is negative value or end index is lower than starting index.

## Java String substring() method example

1. **public** **class** SubstringExample{
2. **public** **static** **void** main(String args[]){
3. String s1="javatpoint";
4. System.out.println(s1.substring(2,4));//returns va
5. System.out.println(s1.substring(2));//returns vatpoint
6. }}

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

va

vatpoint

## Java String substring() Method Example 2

1. **public** **class** SubstringExample2 {
2. **public** **static** **void** main(String[] args) {
3. String s1="Javatpoint";
4. String substr = s1.substring(0); // Starts with 0 and goes to end
5. System.out.println(substr);
6. String substr2 = s1.substring(5,10); // Starts from 5 and goes to 10
7. System.out.println(substr2);
8. String substr3 = s1.substring(5,15); // Returns Exception
9. }
10. }

Javatpoint

point

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 5, end 15, length 10

StringIndexOutOfBoundsException if start index is negative value or end index is lower than starting index.

public class SubstringExample{

public static void main(String args[]){

String s1="java@uvpce";

System.out.println(s1.substring(2,4));//returns va

System.out.println(s1.substring(2));//returns va@uvpce

}}

Output:

va

va@uvpce

**palindrome**

importjava.util.\*;

class Palindrome

{

public static void main(String args[])

{

String original, reverse = "";

Scanner in = new Scanner(System.in);

System.out.println("Enter a string to check if it is a palindrome");

original = in.nextLine();

int length = original.length();

for ( inti = length - 1; i>= 0; i-- )

reverse = reverse + original.charAt(i);

if (original.equals(reverse))

System.out.println("Entered string is a palindrome.");

else

System.out.println("Entered string is not a palindrome.");

}

}

**String sorting**

importjava.util.Scanner;

public class Alphabetical\_Order

{

public static void main(String[] args)

{

int n;

String temp;

Scanner s = new Scanner(System.in);

System.out.print("Enter number of names you want to enter:");

n = s.nextInt();

String names[] = new String[n];

Scanner s1 = new Scanner(System.in);

System.out.println("Enter all the names:");

for(int i = 0; i< n; i++)

{

names[i] = s1.nextLine();

}

for (int i = 0; i< n; i++)

{

for (int j = i + 1; j < n; j++)

{

if (names[i].compareTo(names[j])>0)

{

temp = names[i];

names[i] = names[j];

names[j] = temp;

}

}

}

System.out.print("Names in Sorted Order:");

for (inti = 0; i< =n - 1; i++)

{

System.out.print(names[i] + ",");

}

}

}

**Arrays**

**importjava.util.Arrays;**

**...**

**...**

**Arrays.sort (int [])**

**Arrays.sort (String [])**

**Arrays.sort (float [])**

**Arrays.sort (double [])**

**Arrays.sort (long [])**

**Arrays.sort (Object [])**

String [] stringArray = {"ab", "aB", "c", "0", "2", "1Ad", "a10"};

System.out.println("\*\*\*\*\*\* Unsorted String Array \*\*\*\*\*\*\*");

for (String str : stringArray) {

System.out.println(str);

}

Output:

\*\*\*\*\*\* unsorted string \*\*\*\*\*\*\*

ab

aB

c

0

2

1Ad

a10

Sort in Ascending Order

//Sort array in ascending order

Arrays.sort(stringArray);

System.out.println("\*\*\*\*\*\* Sorted String Array \*\*\*\*\*\*\*");

for (String str : stringArray) {

System.out.println(str);

}

Output:

\*\*\*\*\*\* Sorted String Array \*\*\*\*\*\*\*

0

1Ad

2

a10

aB

ab

c

**String input = "hello";**

**char[] charArray = input.toCharArray();**

**Arrays.sort(charArray);**

**String sortedString = new String(charArray);**

**System.out.println(sortedString);**

**Q7.**

**WAP to check whether the given string is palindrome or not.**

**Programs.**

**public class prac4\_7 {**

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

**{**

**String s1;**

**System.out.println("enter string");**

**Scanner sc=new Scanner(System.in);**

**s1=sc.nextLine();**

**String s2="";**

**for(int j=s1.length()-1;j>=0;j--)**

**{**

**s2=s2+s1.charAt(j);**

**}**

**boolean b=s1.equals(s2);**

**if(b==true)**

**{**

**System.out.println("Strings are palindrom");**

**}**

**else**

**{**

**System.out.println("Strings are not palindrom");**

**}**

**}**

**}**

**Output**

**enter string**

**hello**

**Strings are not palindrom.**

**Q8.**

**WAP to sort all alphabets of given string and print it.**

**Programs.**

**public class prac4\_8 {**

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

**{**

**String s1;**

**Scanner sc=new Scanner(System.in);**

**System.out.println("enter string");**

**s1=sc.nextLine();**

**int l=s1.length(),i;**

**char ch[]=new char[l];**

**char temp;**

**for(i=0;i<s1.length();i++)**

**{**

**ch[i]=s1.charAt(i);**

**}**

**for(i=0;i<l;i++)**

**{**

**for(int j=i+1;j<l;j++)**

**{**

**if(ch[i]>ch[j])**

**{**

**temp=ch[i];**

**ch[i]=ch[j];**

**ch[j]=temp;**

**}**

**}**

**}**

**String s2="";**

**for(i=0;i<s1.length();i++)**

**{**

**s2=s2+ch[i];**

**}**

**System.out.println(s2);**

**}**

**}**

**Output**

**enter string**

**dcba**

**abcd**

# Difference between String and StringBuffer

There are many differences between String and StringBuffer. A list of differences between String and StringBuffer are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **String** | **StringBuffer** |
| 1) | String class is immutable. | StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

# Java StringBuffer class

Java StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

#### Note: Java StringBuffer class is thread-safe i.e. multiple threads cannot access it simultaneously. So it is safe and will result in an order.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringBuffer() | creates an empty string buffer with the initial capacity of 16. |
| StringBuffer(String str) | creates a string buffer with the specified string. |
| StringBuffer(int capacity) | creates an empty string buffer with the specified capacity as length. |

### Important Constructors of StringBuffer class

### Important methods of StringBuffer class

|  |  |  |
| --- | --- | --- |
| **Modifier and Type** | **Method** | **Description** |
| public synchronized StringBuffer | append(String s) | is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc. |
| public synchronized StringBuffer | insert(int offset, String s) | is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc. |
| public synchronized StringBuffer | replace(int startIndex, int endIndex, String str) | is used to replace the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | delete(int startIndex, int endIndex) | is used to delete the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | reverse() | is used to reverse the string. |
| public int | capacity() | is used to return the current capacity. |
| public void | ensureCapacity(int minimumCapacity) | is used to ensure the capacity at least equal to the given minimum. |
| public char | charAt(int index) | is used to return the character at the specified position. |
| public int | length() | is used to return the length of the string i.e. total number of characters. |
| public String | substring(int beginIndex) | is used to return the substring from the specified beginIndex. |
| public String | substring(int beginIndex, int endIndex) | is used to return the substring from the specified beginIndex and endIndex. |

What is mutable string

A string that can be modified or changed is known as mutable string. StringBuffer and StringBuilder classes are used for creating mutable string.

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

1. **class** StringBufferExample{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer("Hello ");
4. sb.append("Java");//now original string is changed
5. System.out.println(sb);//prints Hello Java
6. }
7. }

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

1. **class** StringBufferExample2{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer("Hello ");
4. sb.insert(1,"Java");//now original string is changed
5. System.out.println(sb);//prints HJavaello
6. }
7. }

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex.

1. **class** StringBufferExample3{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer("Hello");
4. sb.replace(1,3,"Java");
5. System.out.println(sb);//prints HJavalo
6. }
7. }

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

1. **class** StringBufferExample4{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer("Hello");
4. sb.delete(1,3);
5. System.out.println(sb);//prints Hlo
6. }
7. }

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

1. **class** StringBufferExample5{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer("Hello");
4. sb.reverse();
5. System.out.println(sb);//prints olleH
6. }
7. }

6) StringBuffer capacity() method

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

1. **class** StringBufferExample6{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
9. }
10. }

7) StringBuffer ensureCapacity() method

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity\*2)+2. For example if your current capacity is 16, it will be (16\*2)+2=34.

1. **class** StringBufferExample7{
2. **public** **static** **void** main(String args[]){
3. StringBuffer sb=**new** StringBuffer();
4. System.out.println(sb.capacity());//default 16
5. sb.append("Hello");
6. System.out.println(sb.capacity());//now 16
7. sb.append("java is my favourite language");
8. System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2
9. sb.ensureCapacity(10);//now no change
10. System.out.println(sb.capacity());//now 34
11. sb.ensureCapacity(50);//now (34\*2)+2
12. System.out.println(sb.capacity());//now 70
13. }
14. }

## Performance Test of String and StringBuffer

1. **public** **class** ConcatTest{
2. **public** **static** String concatWithString()    {
3. String t = "Java";
4. **for** (**int** i=0; i<10000; i++){
5. t = t + "Tpoint";
6. }
7. **return** t;
8. }
9. **public** **static** String concatWithStringBuffer(){
10. StringBuffer sb = **new** StringBuffer("Java");
11. **for** (**int** i=0; i<10000; i++){
12. sb.append("Tpoint");
13. }
14. **return** sb.toString();
15. }
16. **public** **static** **void** main(String[] args){
17. **long** startTime = System.currentTimeMillis();
18. concatWithString();
19. System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
20. startTime = System.currentTimeMillis();
21. concatWithStringBuffer();
22. System.out.println("Time taken by Concating with  StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");
23. }
24. }

Time taken by Concating with String: 578ms

Time taken by Concating with StringBuffer: 0ms