### 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

[**nex→**](https://www.javatpoint.com/immutable-string)

# 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, [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 is String in Java 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:

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".

### 2) By new keyword

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

### 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. }}

# Java String charAt()

The **java string 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 length of the string. It returns **StringIndexOutOfBoundsException** if given index number is greater than or equal to this string length or a negative number.

1. public class CharAtExample{
2. public static void main(String args[]){
3. String name="javatpoint";
4. char ch=name.charAt(4);//returns the char value at the 4th index
5. System.out.println(ch);
6. }}

**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. }

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

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. }

**next »« p**Java String compareTo()

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

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
4. **public** **class** CompareToExample{
5. **public** **static** **void** main(String args[]){
6. String s1="hello";
7. String s2="hello";
8. String s3="meklo";
9. String s4="hemlo";
10. String s5="flag";
11. System.out.println(s1.compareTo(s2));//0 because both are equal
12. System.out.println(s1.compareTo(s3));//-5 because "h" is 5 times lower than "m"
13. System.out.println(s1.compareTo(s4));//-1 because "l" is 1 times lower than "m"
14. System.out.println(s1.compareTo(s5));//2 because "h" is 2 times greater than "f"
15. }}
16. 0
17. -5
18. -1
19. 2

## Java String compareTo(): empty string

If you compare string with blank or empty string, it returns length of the string. If second string is empty, result would be positive. If first string is empty, result would be negative.

1. **public** **class** CompareToExample2{
2. **public** **static** **void** main(String args[]){
3. String s1="hello";
4. String s2="";
5. String s3="me";
6. System.out.println(s1.compareTo(s2));
7. System.out.println(s2.compareTo(s3));
8. }}

5

-2

The **java string concat()** method combines specified string at the end of this string. It returns combined string. It is like appending another string.

Example1:

**public** String concat(String anotherString)

1. **public** **class** ConcatExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java string";
4. s1.concat("is immutable");
5. System.out.println(s1);
6. s1=s1.concat(" is immutable so assign it explicitly");
7. System.out.println(s1);
8. }}

Example2:  
**public** **class** ConcatExample2 {

1. **public** **static** **void** main(String[] args) {
2. String str1 = "Hello";
3. String str2 = "Javatpoint";
4. String str3 = "Reader";
5. // Concatenating one string
6. String str4 = str1.concat(str2);
7. System.out.println(str4);
8. // Concatenating multiple strings
9. String str5 = str1.concat(str2).concat(str3);
10. System.out.println(str5);
11. }
12. }

Let's see an example where we are concatenating spaces and special chars to the string object.

1. **public** **class** ConcatExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str1 = "Hello";
4. String str2 = "Javatpoint";
5. String str3 = "Reader";
6. // Concatenating Space among strings
7. String str4 = str1.concat(" ").concat(str2).concat(" ").concat(str3);
8. System.out.println(str4);
9. // Concatenating Special Chars
10. String str5 = str1.concat("!!!");
11. System.out.println(str5);
12. String str6 = str1.concat("@").concat(str2);
13. System.out.println(str6);
14. }
15. }  [**« prev**](https://www.javatpoint.com/java-string-concat)

Java String contains()

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

1. **class** ContainsExample{
2. **public** **static** **void** main(String args[]){
3. String name="what do you know about me";
4. System.out.println(name.contains("do you know"));
5. System.out.println(name.contains("about"));
6. System.out.println(name.contains("hello"));
7. }}
8. true
9. true
10. false

Example2:

1. **public** **class** ContainsExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str = "To learn Java visit Javatpoint.com";
4. **if**(str.contains("Javatpoint.com")) {
5. System.out.println("This string contains javatpoint.com");
6. }**else**
7. System.out.println("Result not found");
8. }
9. }

This string contains javatpoint.com

[**next »**](https://www.javatpoint.com/java-string-equals)[**« prev**](https://www.javatpoint.com/java-string-contains)

Java String endsWith()

The **java string endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

**public** **boolean** endsWith(String suffix)

1. **public** **class** EndsWithExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java by javatpoint";
4. System.out.println(s1.endsWith("t"));
5. System.out.println(s1.endsWith("point"));
6. }}

true

true

EXAMPLE2:

1. **public** **class** EndsWithExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Welcome to Javatpoint.com";
4. System.out.println(str.endsWith("point"));
5. **if**(str.endsWith(".com")) {
6. System.out.println("String ends with .com");
7. }**else** System.out.println("It does not end with .com");
8. }
9. }

# Java String equals()

The **java string equals ()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

**public** **boolean** equals(Object anotherObject)

1. **public** **class** EqualsExample{
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.equals(s2));//true because content and case is same
8. System.out.println(s1.equals(s3));//false because case is not same
9. System.out.println(s1.equals(s4));//false because content is not same
10. }}

EXAMPLE2:

1. **public** **class** EqualsExample {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "javatpoint";
4. String s2 = "javatpoint";
5. String s3 = "Javatpoint";
6. System.out.println(s1.equals(s2)); // True because content is same
7. **if** (s1.equals(s3)) {
8. System.out.println("both strings are equal");
9. }**else** System.out.println("both strings are unequal");
10. }
11. }

true

both strings are unequal

# 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.

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

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. }}

true

true

false

[**next »**](https://www.javatpoint.com/java-string-getbytes)[**« prev**](https://www.javatpoint.com/java-string-equals)

Java String format()

The **java string format()** method returns the formatted string by given locale, format and arguments

here are two type of string format() method:

1. **public** **static** String format(String format, Object... args)
2. and,
3. **public** **static** String format(Locale locale, String format, Object... args)

### Parameters

**locale** : specifies the locale to be applied on the format() method.

**format** : format of the string.

**args** : arguments for the format string. It may be zero or more.

1. **public** **class** FormatExample{
2. **public** **static** **void** main(String args[]){
3. String name="sonoo";
4. String sf1=String.format("name is %s",name);
5. String sf2=String.format("value is %f",32.334345);
6. String sf3=String.format("value is %32.12f",32.33434);//returns 12 char fractional part filling with 0
8. System.out.println(sf1);
9. System.out.println(sf2);
10. System.out.println(sf3);
11. }}

name is sonoo

value is 32.334340

value is 32.334340000000

1. **public** **class** FormatExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str1 = String.format("%d", 101);          // Integer value
4. String str2 = String.format("%s", "Amar Singh"); // String value
5. String str3 = String.format("%f", 101.00);       // Float value
6. String str4 = String.format("%x", 101);          // Hexadecimal value
7. String str5 = String.format("%c", 'c');          // Char value
8. System.out.println(str1);
9. System.out.println(str2);
10. System.out.println(str3);
11. System.out.println(str4);
12. System.out.println(str5);
13. }
15. }

101

Amar Singh

101.000000

65

c

# Java String getBytes()

The **java string getBytes()** method returns the byte array of the string. In other words, it returns sequence of bytes.

### Signature

There are **3 variant of getBytes()** method. The signature or syntax of string getBytes() method is given below:

1. **public** **byte**[] getBytes()
2. **public** **byte**[] getBytes(Charset charset)
3. **public** **byte**[] getBytes(String charsetName)**throws** UnsupportedEncodingException
4. **public class StringGetBytesExample{**
6. **public static void main(String args[])**
7. **{ String s1="ABCDEFG";**
8. **byte[] barr=s1.getBytes();**
9. **for(int i=0;i<barr.length;i++){**
10. **System.out.println(barr[i]);**
11. **}**
12. **}}**
13. 65
14. 66
15. 67
16. 68
17. 69
18. 70
19. 71

This method returns a byte array that again can be passed to String constructor to get String.

1. **public** **class** StringGetBytesExample2 {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "ABCDEFG";
4. **byte**[] barr = s1.getBytes();
5. **for**(**int** i=0;i<barr.length;i++){
6. System.out.println(barr[i]);
7. }
8. // Getting string back
9. String s2 = **new** String(barr);
10. System.out.println(s2);
11. }
12. }
13. 65
14. 66
15. 67
16. 68
17. 69
18. 70
19. 71
20. ABCDEFG

# Java String getChars()

The **java string getChars()** method copies the content of this string into specified char array. There are 4 arguments passed in getChars() method. The signature of getChars() method is given below:

**public** **void** getChars(**int** srcBeginIndex, **int** srcEndIndex, **char**[] destination, **int** dstBeginIndex)

1. **public** **class** StringGetCharsExample{
2. **public** **static** **void** main(String args[]){
3. String str = **new** String("hello javatpoint how r u");
4. **char**[] ch = **new** **char**[10];
5. **try**{
6. str.getChars(6, 16, ch, 0);
7. System.out.println(ch);
8. }**catch**(Exception ex){System.out.println(ex);}
9. }}
10. javatpoint

# Java String indexOf()

The **java string indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

here are 4 types of indexOf method in java. The signature of indexOf methods are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1 | int indexOf(int ch) | returns index position for the given char value |
| 2 | int indexOf(int ch, int fromIndex) | returns index position for the given char value and from index |
| 3 | int indexOf(String substring) | returns index position for the given substring |
| 4 | int indexOf(String substring, int fromIndex) | returns index position for the given substring and from index |

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. }}

2 8

5

3

Example2:

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. }

index of substring 16

## Example3:

## 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. }

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. }

index of char 17

# Java String isEmpty()

The **java string isEmpty()** method checks if this string is empty or not. It returns *true*, if length of string is 0 otherwise false. In other words, true is returned if string is empty otherwise it returns false.

**public** **boolean** isEmpty()

## Java String isEmpty() method example

1. **public** **class** IsEmptyExample{
2. **public** **static** **void** main(String args[]){
3. String s1="";
4. String s2="javatpoint";
6. System.out.println(s1.isEmpty());
7. System.out.println(s2.isEmpty());
8. }}

true

false

## Java String isEmpty() Method Example 2

1. **public** **class** IsEmptyExample2 {
2. **public** **static** **void** main(String[] args) {
3. String s1="";
4. String s2="Javatpoint";
5. // Either length is zero or isEmpty is true
6. **if**(s1.length()==0 || s1.isEmpty())
7. System.out.println("String s1 is empty");
8. **else**
9. System.out.println("s1");
10. **if**(s2.length()==0 || s2.isEmpty())
11. System.out.println("String s2 is empty");
12. **else** System.out.println(s2);
13. }
14. }

String s1 is empty

Javatpoint

**java string join()**

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added.

**public** **static** String join(CharSequence delimiter, CharSequence... elements)

## Java String join() method example

1. **public** **class** StringJoinExample{
2. **public** **static** **void** main(String args[]){
3. String joinString1=String.join("-","welcome","to","javatpoint");
4. System.out.println(joinString1);
5. }}
6. welcome-to-javatpoint

Example2:

## java String join() Method Example 2

We can use delimeter to format the string as we did in the below example to show date and time.

1. **public** **class** StringJoinExample2 {
2. **public** **static** **void** main(String[] args) {
3. String date = String.join("/","25","06","2018");
4. System.out.print(date);
5. String time = String.join(":", "12","10","10");
6. System.out.println(" "+time);
7. }
8. }
9. 25/06/2018 12:10:10

# Java String lastIndexOf()

The **java string lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

int lastIndexOf(int ch)

## Java String lastIndexOf() method example

1. **public** **class** LastIndexOfExample{
2. **public** **static** **void** main(String args[]){
3. String s1="this is index of example";//there are 2 's' characters in this sentence
4. **int** index1=s1.lastIndexOf('s');//returns last index of 's' char value
5. System.out.println(index1);//6
6. }}

6

## Java String lastIndexOf(int ch, int fromIndex) Method Example

1. **public** **class** LastIndexOfExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "This is index of example";
4. **int** index = str.lastIndexOf('s',5);
5. System.out.println(index);
6. }
7. }

3

Java String lastIndexOf(String substring) Method Example

It returns the last index of the substring.

1. **public** **class** LastIndexOfExample3 {
2. **public** **static** **void** main(String[] args) {
3. String str = "This is last index of example";
4. **int** index = str.lastIndexOf("of");
5. System.out.println(index);
6. }
7. }

19

# Java String length()

The **java string length()** method length of the string. It returns count of total number of characters.

**public** **int** length()

1. **public** **class** LengthExample{
2. **public** **static** **void** main(String args[]){
3. String s1="javatpoint";
4. String s2="python";
5. System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string
6. System.out.println("string length is: "+s2.length());//6 is the length of python string
7. }}

string length is: 10

string length is: 6

Example2:

1. **public** **class** LengthExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Javatpoint";
4. **if**(str.length()>0) {
5. System.out.println("String is not empty and length is: "+str.length());
6. }
7. str = "";
8. **if**(str.length()==0) {
9. System.out.println("String is empty now: "+str.length());
10. }
11. }
12. }

String is not empty and length is: 10

String is empty now: 0

[**next »**](https://www.javatpoint.com/java-string-replaceall)[**« prev**](https://www.javatpoint.com/java-string-length)

Java String replace()

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

**public** String replace(**char** oldChar, **char** newChar)

## Java 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. }}

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. }}

my name was khan my name was java

# Java String replaceAll()

The **java string replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string.

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

1. **public** **class** ReplaceAllExample1{
2. **public** **static** **void** main(String args[]){
3. String s1="javatpoint is a very good website";
4. String replaceString=s1.replaceAll("a","e");//replaces all occurrences of "a" to "e"
5. System.out.println(replaceString);
6. }}

## Java String replaceAll() example: replace word

Let's see an example to replace all the occurrences of **single word or set of words**.

1. **public** **class** ReplaceAllExample2{
2. **public** **static** **void** main(String args[]){
3. String s1="My name is Khan. My name is Bob. My name is Sonoo.";
4. String replaceString=s1.replaceAll("is","was");//replaces all occurrences of "is" to "was"
5. System.out.println(replaceString);
6. }}

My name was Khan. My name was Bob. My name was Sonoo.

## Java String replaceAll() example: remove white spaces

Let's see an example to remove all the occurrences of **white spaces**.

1. **public** **class** ReplaceAllExample3{
2. **public** **static** **void** main(String args[]){
3. String s1="My name is Khan. My name is Bob. My name is Sonoo.";
4. String replaceString=s1.replaceAll("\\s","");
5. System.out.println(replaceString);
6. }}
7. MynameisKhan.MynameisBob.MynameisSonoo.

Java String split()

The **java string split()** method splits this string against given regular expression and returns a char array.

## Java String split() method example

The given example returns total number of words in a string excluding space only. It also includes special characters.

1. **public** **class** SplitExample{
2. **public** **static** **void** main(String args[]){
3. String s1="java string split method by javatpoint";
4. String[] words=s1.split("\\s");//splits the string based on whitespace
5. //using java foreach loop to print elements of string array
6. **for**(String w:words){
7. System.out.println(w);
8. }
9. }}
10. java
11. string
12. split
13. method
14. by
15. javatpoint
16. **public** **class** SplitExample2{
17. **public** **static** **void** main(String args[]){
18. String s1="welcome to split world";
19. System.out.println("returning words:");
20. **for**(String w:s1.split("\\s",0)){
21. System.out.println(w);
22. }
23. System.out.println("returning words:");
24. **for**(String w:s1.split("\\s",1)){
25. System.out.println(w);
26. }
27. System.out.println("returning words:");
28. **for**(String w:s1.split("\\s",2)){
29. System.out.println(w);
30. }
32. }}
33. returning words:
34. welcome
35. to
36. split
37. world
38. returning words:
39. welcome to split world
40. returning words:
41. welcome
42. to split world

# Java String startsWith()

The **java string startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

1. **public** **boolean** startsWith(String prefix)

1. **public** **boolean** startsWith(String prefix, **int** offset)
2. **public** **class** StartsWithExample{
3. **public** **static** **void** main(String args[]){
4. String s1="java string split method by javatpoint";
5. System.out.println(s1.startsWith("ja"));
6. System.out.println(s1.startsWith("java string"));
7. }}

true

true

1. **public** **class** StartsWithExample2 {
2. **public** **static** **void** main(String[] args) {
3. String str = "Javatpoint";
4. System.out.println(str.startsWith("J")); // True
5. System.out.println(str.startsWith("a")); // False
6. System.out.println(str.startsWith("a",1)); // True
7. }
8. }
9. true
10. false
11. true

# 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.

1. **public** String substring(**int** startIndex)
2. and
3. **public** String substring(**int** startIndex, **int** endIndex)
4. **public** **class** SubstringExample{
5. **public** **static** **void** main(String args[]){
6. String s1="javatpoint";
7. System.out.println(s1.substring(2,4));//returns va
8. System.out.println(s1.substring(2));//returns vatpoint
9. }}

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. }
11. Javatpoint
12. point
13. Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 5, end 15, length 10

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.

**public** **char**[] toCharArray()

1. **public** **class** StringToCharArrayExample{
2. **public** **static** **void** main(String args[]){
3. String s1="hello";
4. **char**[] ch=s1.toCharArray();
5. **for**(**int** i=0;i<ch.length;i++){
6. System.out.print(ch[i]);
7. }
8. }}

hello

1. **public** **class** StringToCharArrayExample2 {
2. **public** **static** **void** main(String[] args) {
3. String s1 = "Welcome to Javatpoint";
4. **char**[] ch = s1.toCharArray();
5. **int** len = ch.length;
6. System.out.println("Char Array length: " + len);
7. System.out.println("Char Array elements: ");
8. **for** (**int** i = 0; i < len; i++) {
9. System.out.println(ch[i]);
10. }
11. }
12. }

Char Array length: 21

Char Array elements:

W

e

l

c

o

m

e

t

o

J

a

v

a

t

p

o

i

n

t

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.

1. **public** **class** StringLowerExample{
2. **public** **static** **void** main(String args[]){
3. String s1="JAVATPOINT HELLO stRIng";
4. String s1lower=s1.toLowerCase();
5. System.out.println(s1lower);
6. }}

javatpoint hello string

1. **import** java.util.Locale;
2. **public** **class** StringLowerExample2 {
3. **public** **static** **void** main(String[] args) {
4. String s = "JAVATPOINT HELLO stRIng";
5. String eng = s.toLowerCase(Locale.ENGLISH);
6. System.out.println(eng);
7. String turkish = s.toLowerCase(Locale.forLanguageTag("tr")); // It shows i without dot
8. System.out.println(turkish);
9. }
10. }

javatpoint hello string

javatpo?nt hello str?ng

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.

1. **public** **class** StringUpperExample{
2. **public** **static** **void** main(String args[]){
3. String s1="hello string";
4. String s1upper=s1.toUpperCase();
5. System.out.println(s1upper);
6. }}

HELLO STRING

1. **import** java.util.Locale;
2. **public** **class** StringUpperExample2 {
3. **public** **static** **void** main(String[] args) {
4. String s = "hello string";
5. String turkish = s.toUpperCase(Locale.forLanguageTag("tr"));
6. String english = s.toUpperCase(Locale.forLanguageTag("en"));
7. System.out.println(turkish);//will print I with dot on upper side
8. System.out.println(english);
9. }
10. }

HELLO STR?NG

HELLO STRING

Java String trim()

The **java string 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 removes the spaces and returns the omitted string.

1. **public** **class** StringTrimExample{
2. **public** **static** **void** main(String args[]){
3. String s1="  hello string   ";
4. System.out.println(s1+"javatpoint");//without trim()
5. System.out.println(s1.trim()+"javatpoint");//with trim()
6. }}

hello string javatpoint

hello stringjavatpoint

1. **public** **class** StringTrimExample {
2. **public** **static** **void** main(String[] args) {
3. String s1 =" hello java string ";
4. System.out.println(s1.length());
5. System.out.println(s1); //Without trim()
6. String tr = s1.trim();
7. System.out.println(tr.length());
8. System.out.println(tr); //With trim()
9. }
10. }

22

hello java string

17

hello java string