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:

**char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=**new** String(ch);

is same as:

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(), ubstring() etc.

### **What is String in Java?**

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

### **How to create a string object?**

There are two ways to create String object:

1. By string literal
2. By new keyword

### **1) String Literal**

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

String s="welcome";

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

String s1="Welcome";

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.

### **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**

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

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

will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

### **Java String Example**

**StringExample.java**

**public** **class** StringExample{

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

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

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

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

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

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}

**Output:**

java

strings

example

### **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. |
|  |  |  |

## **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.

**Teststringcomparison1.java**

**class** Teststringcomparison1{

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

   String s1="Sachin";

   String s2="Sachin";

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

   String s4="Saurav";

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

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

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

 }

}

**Output:**

true

true

false

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

**Teststringcomparison2.java**

**class** Teststringcomparison2{

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

   String s1="Sachin";

   String s2="SACHIN";

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

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

 }

}

**Output:**

false

true

# String Concatenation in Java

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

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

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

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

**TestStringConcatenation1.java**

**class** TestStringConcatenation1{

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

   String s="Sachin"+" Tendulkar";

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

 }

}

**Output:**tagram Videos Now Have Auto-Generated Captions

Sachin Tendulkar

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

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. For Example:

**TestStringConcatenation2.java**

**class** TestStringConcatenation2{

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

   String s=50+30+"Sachin"+40+40;

   System.out.println(s);//80Sachin4040

 }

}

**Output:**

80Sachin4040

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

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

**public** String concat(String another)

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

**TestStringConcatenation3.java**

**class** TestStringConcatenation3{

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

   String s1="Sachin ";

   String s2="Tendulkar";

   String s3=s1.concat(s2);

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

  }

}

# 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.

**public** **class** TestSubstring{

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

 String s="SachinTendulkar";

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

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

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

 }

}

# 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**

**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.

**public** **class** CharAtExample{

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

String name="javatpoint";

**char** ch=name.charAt(4);//returns the char value at the 4th index

System.out.println(ch);

}}

**Output:**

t

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

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

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

**if** s1 > s2, it returns positive number

**if** s1 < s2, it returns negative number

**if** s1 == s2, it returns 0

### **Syntax**

**public** **int** compareTo(String anotherString)

The method accepts a parameter of type String that is to be compared with the current string.

# Java String concat

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

**public** String concat(String anotherString)

**public** **class** ConcatExample2 {

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

        String str1 = "Hello";

        String str2 = "Javatpoint";

        String str3 = "Reader";

        // Concatenating one string

        String str4 = str1.concat(str2);

        System.out.println(str4);

        // Concatenating multiple strings

        String str5 = str1.concat(str2).concat(str3);

        System.out.println(str5);

    }

}

**Output:**

HelloJavatpoint

HelloJavatpointReader

# 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:

**public** **int** length()

**public** **class** LengthExample{

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

String s1="javatpoint";

String s2="python";

System.out.println("string length is: "+s1.length());//10 is the length of javatpoint string

System.out.println("string length is: "+s2.length());//6 is the length of python string

}}

# 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.

**public** **class** StringUpperExample{

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

String s1="hello string";

String s1upper=s1.toUpperCase();

System.out.println(s1upper);

}}

Output:

HELLO STRING

# 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.

**public** **class** StringLowerExample{

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

String s1="JAVATPOINT HELLO stRIng";

String s1lower=s1.toLowerCase();

System.out.println(s1lower);

}}

Output:

javatpoint hello string

# Java StringBuffer Class

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

### **Important Constructors of StringBuffer Class**

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

### **What is a mutable String?**

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

### **1) StringBuffer Class append() Method**

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

**StringBufferExample.java**

**class** StringBufferExample{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

**Output:**

Hello Java

### **2) StringBuffer insert() Method**

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

**StringBufferExample2.java**

**class** StringBufferExample2{

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

StringBuffer sb=**new** StringBuffer("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}  }

**Output:**

HJavaello

### **3) StringBuffer replace() Method**

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

**StringBufferExample3.java**

**class** StringBufferExample3{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

**Output:**

HJavalo

### **4) StringBuffer delete() Method**

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

**StringBufferExample4.java**

**class** StringBufferExample4{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

**Output:**

Hlo

### **5) StringBuffer reverse() Method**

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

**StringBufferExample5.java**

**class** StringBufferExample5{

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

StringBuffer sb=**new** StringBuffer("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

**Output:**

olleH

### **6) StringBuffer capacity() Method**

The capacity() method of the 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.

**StringBufferExample6.java**

**class** StringBufferExample6{

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

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

**Output:**

16

16

34

### **7) StringBuffer ensureCapacity() method**

The ensureCapacity() method of the 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.

**StringBufferExample7.java**

**class** StringBufferExample7{

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

StringBuffer sb=**new** StringBuffer();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

**Output:**

16

16

34

34

70

# Java StringBuilder Class

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

## **Important Constructors of StringBuilder class**

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

## **Important methods of StringBuilder class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public StringBuilder append(String s) | It 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 StringBuilder insert(int offset, String s) | It 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 StringBuilder replace(int startIndex, int endIndex, String str) | It is used to replace the string from specified startIndex and endIndex. |
| public StringBuilder delete(int startIndex, int endIndex) | It is used to delete the string from specified startIndex and endIndex. |
| public StringBuilder reverse() | It is used to reverse the string. |
| public int capacity() | It is used to return the current capacity. |
| public void ensureCapacity(int minimumCapacity) | It is used to ensure the capacity at least equal to the given minimum. |
| public char charAt(int index) | It is used to return the character at the specified position. |
| public int length() | It is used to return the length of the string i.e. total number of characters. |
| public String substring(int beginIndex) | It is used to return the substring from the specified beginIndex. |
| public String substring(int beginIndex, int endIndex) | It is used to return the substring from the specified beginIndex and endIndex. |

## **Java StringBuilder Examples**

Let's see the examples of different methods of StringBuilder class.

### **1) StringBuilder append() method**

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

**StringBuilderExample.java**

16.3M

255

Triggers in SQL (Hindi)

**class** StringBuilderExample{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

**Output:**

Hello Java

### **2) StringBuilder insert() method**

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

**StringBuilderExample2.java**

**class** StringBuilderExample2{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

**Output:**

HJavaello

### **3) StringBuilder replace() method**

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

**StringBuilderExample3.java**

**class** StringBuilderExample3{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

**Output:**

HJavalo

### **4) StringBuilder delete() method**

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

**StringBuilderExample4.java**

**class** StringBuilderExample4{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

**Output:**

Hlo

### **5) StringBuilder reverse() method**

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

**StringBuilderExample5.java**

**class** StringBuilderExample5{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

**Output:**

olleH

### **6) StringBuilder capacity() method**

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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.

**StringBuilderExample6.java**

**class** StringBuilderExample6{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("Java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

**Output:**

16

16

34

### **7) StringBuilder ensureCapacity() method**

The ensureCapacity() method of StringBuilder 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.

**StringBuilderExample7.java**

**class** StringBuilderExample7{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("Java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

**Output:**

16

16

34

34

70

# Java StringBuilder Class

Java StringBuilder class is used to create mutable (modifiable) String. The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

## **Important Constructors of StringBuilder class**

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

## **Important methods of StringBuilder class**

|  |  |
| --- | --- |
| **Method** | **Description** |
| public StringBuilder append(String s) | It 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 StringBuilder insert(int offset, String s) | It 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 StringBuilder replace(int startIndex, int endIndex, String str) | It is used to replace the string from specified startIndex and endIndex. |
| public StringBuilder delete(int startIndex, int endIndex) | It is used to delete the string from specified startIndex and endIndex. |
| public StringBuilder reverse() | It is used to reverse the string. |
| public int capacity() | It is used to return the current capacity. |
| public void ensureCapacity(int minimumCapacity) | It is used to ensure the capacity at least equal to the given minimum. |
| public char charAt(int index) | It is used to return the character at the specified position. |
| public int length() | It is used to return the length of the string i.e. total number of characters. |
| public String substring(int beginIndex) | It is used to return the substring from the specified beginIndex. |
| public String substring(int beginIndex, int endIndex) | It is used to return the substring from the specified beginIndex and endIndex. |

## **Java StringBuilder Examples**

Let's see the examples of different methods of StringBuilder class.

### **1) StringBuilder append() method**

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

**StringBuilderExample.java**ML Tutorial

**class** StringBuilderExample{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.append("Java");//now original string is changed

System.out.println(sb);//prints Hello Java

}

}

**Output:**

Hello Java

### **2) StringBuilder insert() method**

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

**StringBuilderExample2.java**

**class** StringBuilderExample2{

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

StringBuilder sb=**new** StringBuilder("Hello ");

sb.insert(1,"Java");//now original string is changed

System.out.println(sb);//prints HJavaello

}

}

**Output:**

HJavaello

### **3) StringBuilder replace() method**

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

**StringBuilderExample3.java**

**class** StringBuilderExample3{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.replace(1,3,"Java");

System.out.println(sb);//prints HJavalo

}

}

**Output:**

HJavalo

### **4) StringBuilder delete() method**

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

**StringBuilderExample4.java**

**class** StringBuilderExample4{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.delete(1,3);

System.out.println(sb);//prints Hlo

}

}

**Output:**

Hlo

### **5) StringBuilder reverse() method**

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

**StringBuilderExample5.java**

**class** StringBuilderExample5{

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

StringBuilder sb=**new** StringBuilder("Hello");

sb.reverse();

System.out.println(sb);//prints olleH

}

}

**Output:**

olleH

### **6) StringBuilder capacity() method**

The capacity() method of StringBuilder class returns the current capacity of the Builder. The default capacity of the Builder 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.

**StringBuilderExample6.java**

**class** StringBuilderExample6{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("Java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

}

}

**Output:**

16

16

34

### **7) StringBuilder ensureCapacity() method**

The ensureCapacity() method of StringBuilder 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.

**StringBuilderExample7.java**

**class** StringBuilderExample7{

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

StringBuilder sb=**new** StringBuilder();

System.out.println(sb.capacity());//default 16

sb.append("Hello");

System.out.println(sb.capacity());//now 16

sb.append("Java is my favourite language");

System.out.println(sb.capacity());//now (16\*2)+2=34 i.e (oldcapacity\*2)+2

sb.ensureCapacity(10);//now no change

System.out.println(sb.capacity());//now 34

sb.ensureCapacity(50);//now (34\*2)+2

System.out.println(sb.capacity());//now 70

}

}

**Output:**

16

16

34

34

70

# 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**

**class** Testimmutablestring{

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

   String s="Sachin";

   s.concat(" Tendulkar");//concat() method appends the string at the end

   System.out.println(s);//will print Sachin because strings are immutable objects

 }

}

**Output:**

Sachin

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



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

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

For example:

**Testimmutablestring1.java**

**class** Testimmutablestring1{

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

   String s="Sachin";

   s=s.concat(" Tendulkar");

   System.out.println(s);

 }

}

**Output:**

Sachin Tendulkar

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

### **Why String objects are immutable in Java?**

As Java uses the concept of String literal. Suppose there are 5 reference variables, all refer to one object "Sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why String objects are immutable in Java.

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

**1. ClassLoader:**

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

To avoid this kind of misinterpretation, String is immutable.

**2. Thread Safe:**

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

**3. Security:**

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

**4. Heap Space:**

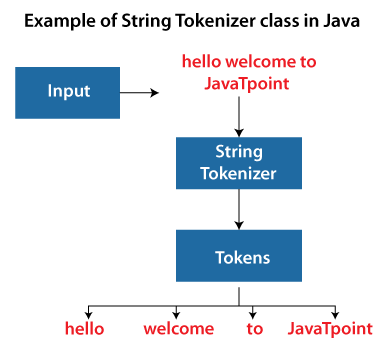
The immutability of String helps to minimize the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space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

# StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.



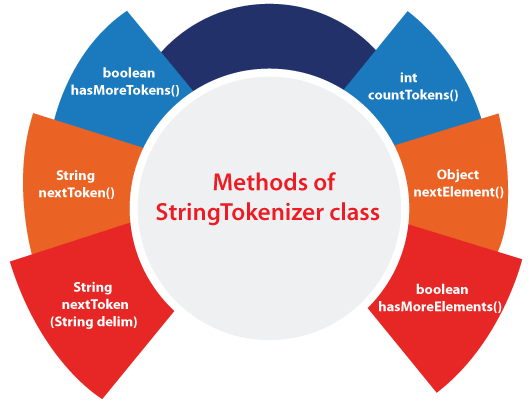
### **Constructors of the StringTokenizer Class**

There are 3 constructors defined in the StringTokenizer class.

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| StringTokenizer(String str) | It creates StringTokenizer with specified string. |
| StringTokenizer(String str, String delim) | It creates StringTokenizer with specified string and delimiter. |
| StringTokenizer(String str, String delim, boolean returnValue) | It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens. |

### **Methods of the StringTokenizer Class**

The six useful methods of the StringTokenizer class are as follows:



|  |  |
| --- | --- |
| **Methods** | **Description** |
| boolean hasMoreTokens() | It checks if there is more tokens available. |
| String nextToken() | It returns the next token from the StringTokenizer object. |
| String nextToken(String delim) | It returns the next token based on the delimiter. |
| boolean hasMoreElements() | It is the same as hasMoreTokens() method. |
| Object nextElement() | It is the same as nextToken() but its return type is Object. |
| int countTokens() | It returns the total number of tokens. |

### **Example of StringTokenizer Class**

Let's see an example of the StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

**Simple.java**

**import** java.util.StringTokenizer;

**public** **class** Simple{

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

   StringTokenizer st = **new** StringTokenizer("my name is khan"," ");

**while** (st.hasMoreTokens()) {

         System.out.println(st.nextToken());

     }

   }

}

**Output:**

my

name

is

khan

The above Java code, demonstrates the use of StringTokenizer class and its methods hasMoreTokens() and nextToken().

### **Example of nextToken(String delim) method of the StringTokenizer class**

**Test.java**

**import** java.util.\*;

**public** **class** Test {

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

       StringTokenizer st = **new** StringTokenizer("my,name,is,khan");

      // printing next token

      System.out.println("Next token is : " + st.nextToken(","));

   }

}

**Output:**

Next token is : my

### **Example of hasMoreTokens() method of the StringTokenizer class**

This method returns true if more tokens are available in the tokenizer String otherwise returns false.

**StringTokenizer1.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer1

{

 /\* Driver Code \*/

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

 {

   /\* StringTokenizer object \*/

   StringTokenizer st = **new** StringTokenizer("Demonstrating methods from StringTokenizer class"," ");

     /\* Checks if the String has any more tokens \*/

**while** (st.hasMoreTokens())

     {

         System.out.println(st.nextToken());

     }

 }

}

**Output:**

Demonstrating

methods

from

StringTokenizer

class

The above Java program shows the use of two methods hasMoreTokens() and nextToken() of StringTokenizer class.

### **Example of hasMoreElements() method of the StringTokenizer class**

This method returns the same value as hasMoreTokens() method of StringTokenizer class. The only difference is this class can implement the Enumeration interface.

**StringTokenizer2.java**

**import** java.util.StringTokenizer;

**public** **class** StringTokenizer2

{

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

 {

   StringTokenizer st = **new** StringTokenizer("Hello everyone I am a Java developer"," ");

**while** (st.hasMoreElements())

     {

         System.out.println(st.nextToken());

     }

 }

}

**Output:**

Hello

everyone

I

am

a

Java

developer