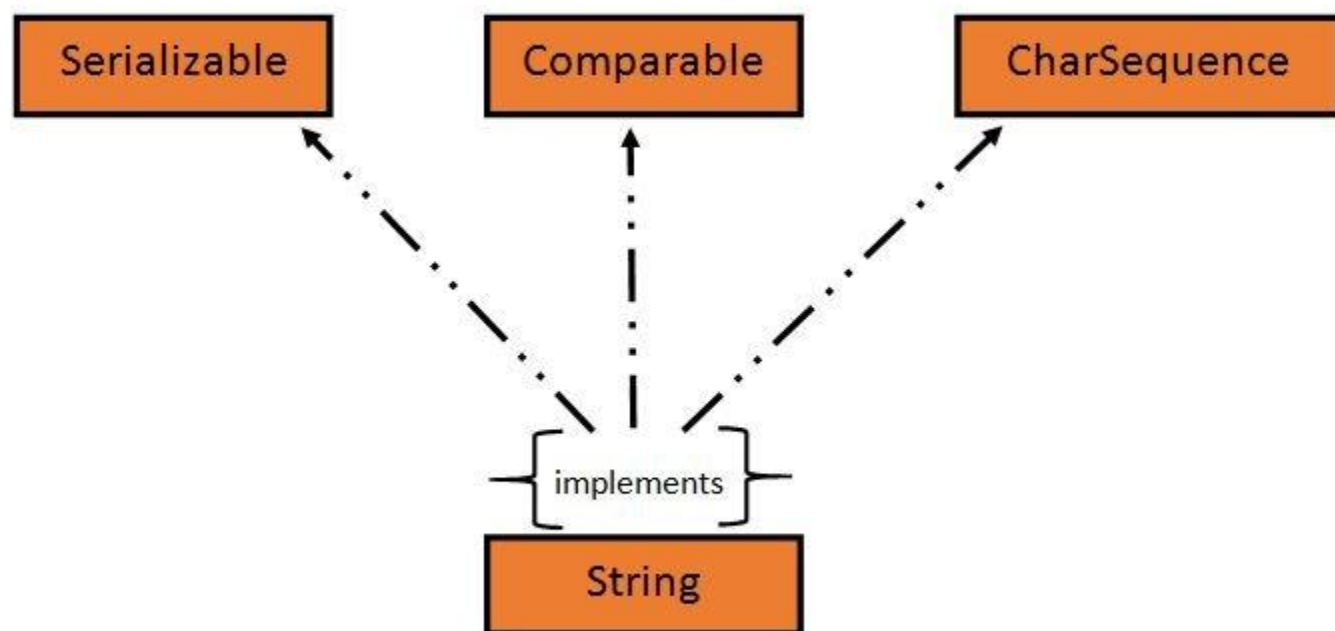


# Introduction to Java String Handling

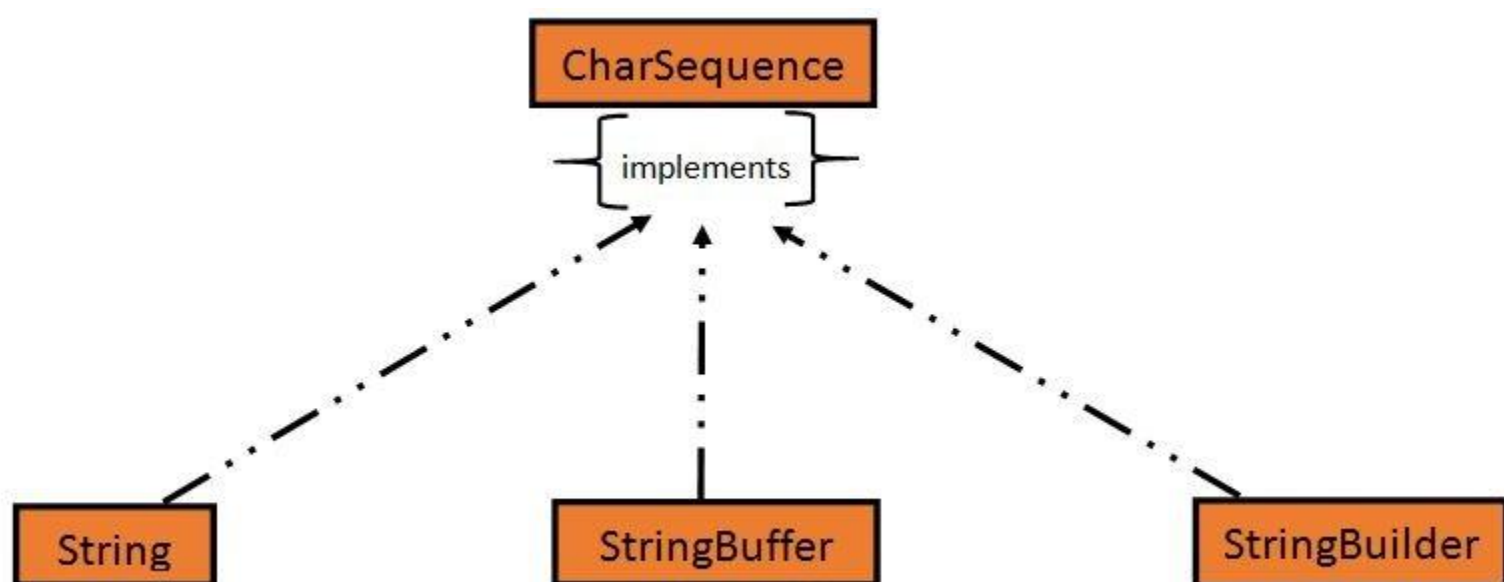
String is an **object** that represents sequence of characters. In Java, String is represented by String class which is located into `java.lang` package

It is probably the most commonly used class in java library. In java, every string that we create is actually an object of type **String**. One important thing to notice about string object is that string objects are **immutable** that means once a string object is created it cannot be changed.

The Java String class implements Serializable, Comparable and CharSequence interface that we have represented using the below image.



In Java, **CharSequence** Interface is used for representing a sequence of characters. CharSequence interface is implemented by String, StringBuffer and StringBuilder classes. This three classes can be used for creating strings in java.



What is an Immutable object?

An object whose state cannot be changed after it is created is known as an Immutable object. String, Integer, Byte, Short, Float, Double and all other wrapper classes objects are immutable.

Creating a String object

String can be created in number of ways, here are a few ways of creating string object.

1) Using a String literal

String literal is a simple string enclosed in double quotes `" "`. A string literal is treated as a String object.

```
public class Demo{

    public static void main(String[] args) {

        String s1 = "Hello Java";

        System.out.println(s1);

    }

}
```

Copy

Hello Java

## 2) Using new Keyword

We can create a new string object by using **new** operator that allocates memory for the object.

```
public class Demo{

    public static void main(String[] args) {

        String s1 = new String("Hello Java");

        System.out.println(s1);

    }

}
```

Copy

Hello Java

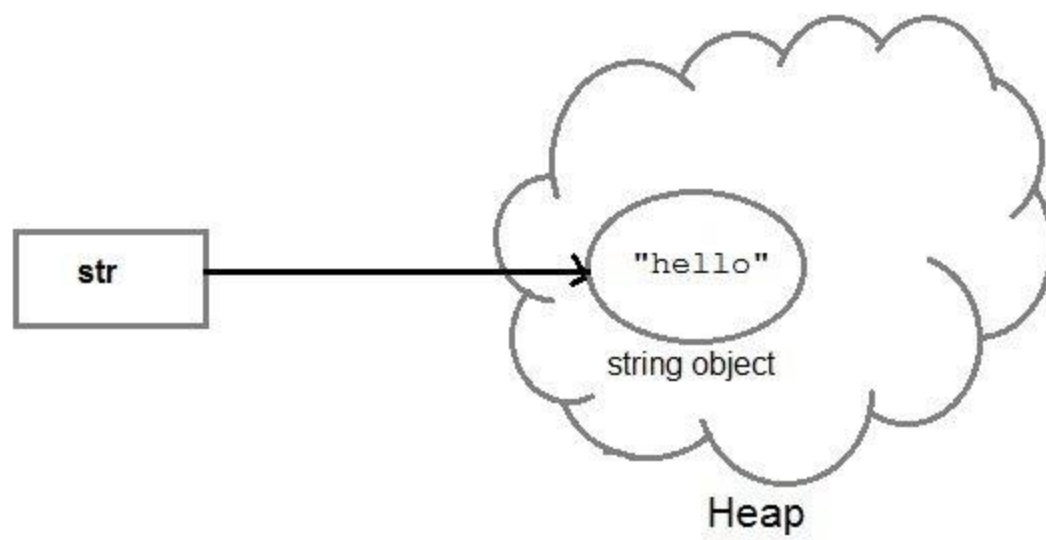
Each time we create a String literal, the JVM checks the string pool first. If the string literal already exists in the pool, a reference to the pool instance is returned. If string does not exist in the pool, a new string object is created, and is placed in the pool. String objects are stored in a special memory area known as **string constant pool** inside the heap memory.

String object and How they are stored

When we create a new string object using string literal, that string literal is added to the string pool, if it is not present there already.

```
String str= "Hello";
```

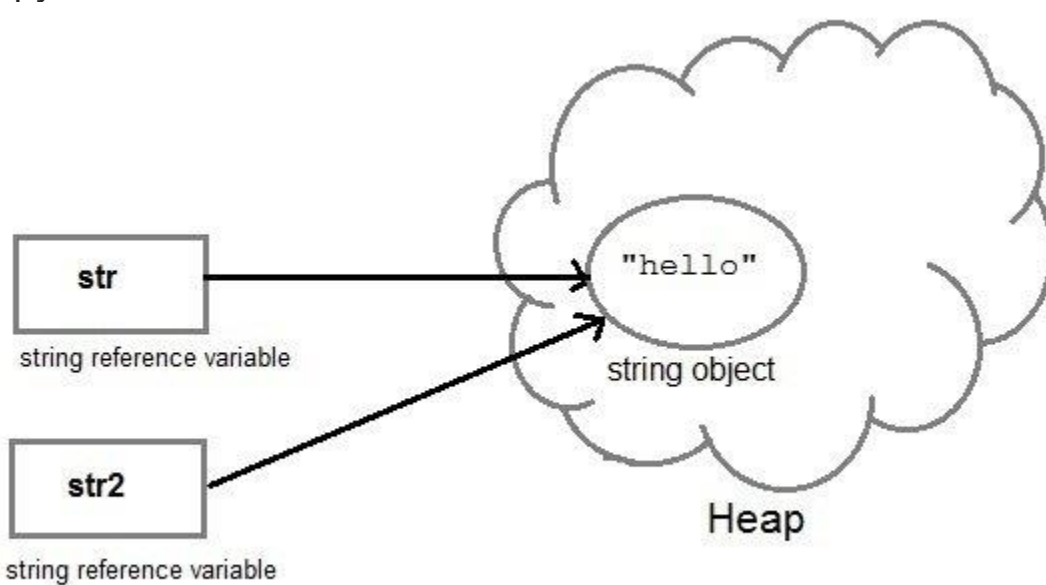
Copy



And, when we create another object with same string, then a reference of the string literal already present in string pool is returned.

```
String str2 = str;
```

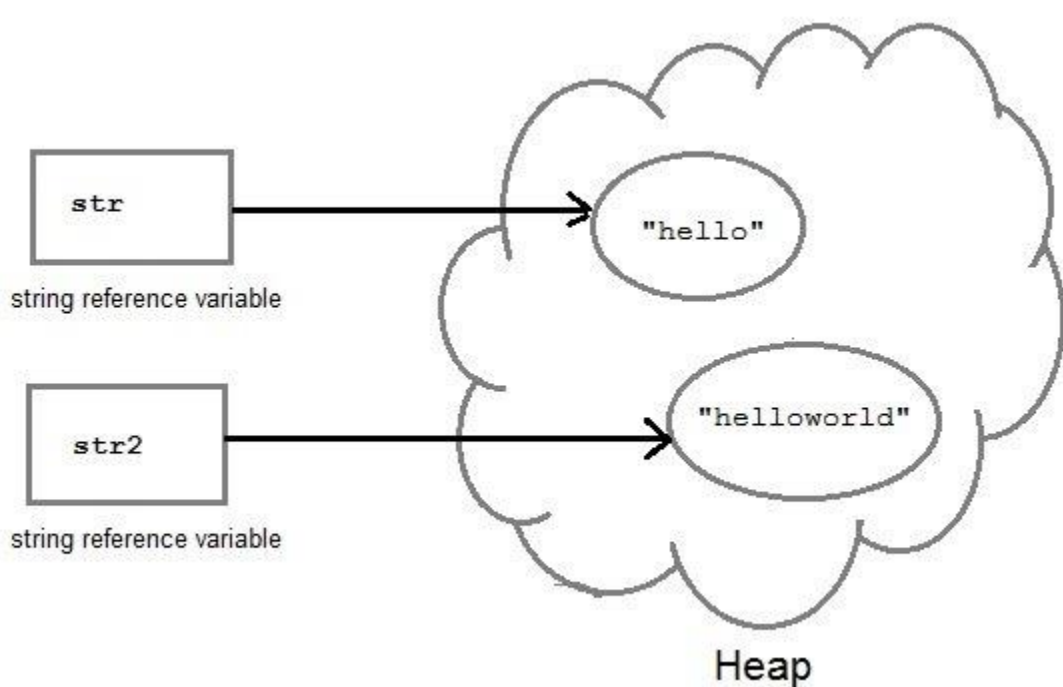
Copy



But if we change the new string, its reference gets modified.

```
str2=str2.concat("world");
```

Copy



Concatenating String

There are 2 methods to concatenate two or more string.

1. Using **concat()** method
2. Using **+** operator

## 1) Using concat() method

**Concat()** method is used to add two or more string into a single string object. It is string class method and returns a string object.

```
public class Demo{

    public static void main(String[] args) {

        String s = "Hello";

        String str = "Java";

        String str1 = s.concat(str);

        System.out.println(str1);

    }

}
```

Copy

HelloJava

## 2) Using + operator

Java uses "+" operator to concatenate two string objects into single one. It can also concatenate numeric value with string object. See the below example.

```
public class Demo{

    public static void main(String[] args) {

        String s = "Hello";

        String str = "Java";

        String str1 = s+str;

        String str2 = "Java"+11;

        System.out.println(str1);

        System.out.println(str2);

    }

}
```

Copy

HelloJava

Java11

## String Comparison

To compare string objects, Java provides methods and operators both. So we can compare string in following three ways.

1. Using `equals()` method
2. Using `==` operator
3. By `compareTo()` method

Using `equals()` method

`equals()` method compares two strings for equality. Its general syntax is,

```
boolean equals (Object str)
```

Copy

Example

It compares the content of the strings. It will return **true** if string matches, else returns **false**.

```
public class Demo{

    public static void main(String[] args) {

        String s = "Hell";

        String s1 = "Hello";

        String s2 = "Hello";

        boolean b = s1.equals(s2);    //true

        System.out.println(b);

        b = s.equals(s1);    //false

        System.out.println(b);

    }

}
```

Copy

```
true
false
```

Using `==` operator

The double `equal (==)` operator compares two object references to check whether they refer to same instance. This also, will return **true** on successful match else returns false.

```
public class Demo{

    public static void main(String[] args) {

        String s1 = "Java";
```

```

String s2 = "Java";

String s3 = new String ("Java");

boolean b = (s1 == s2);    //true

System.out.println(b);

b = (s1 == s3);    //false

System.out.println(b);

}

}

```

Copy

```

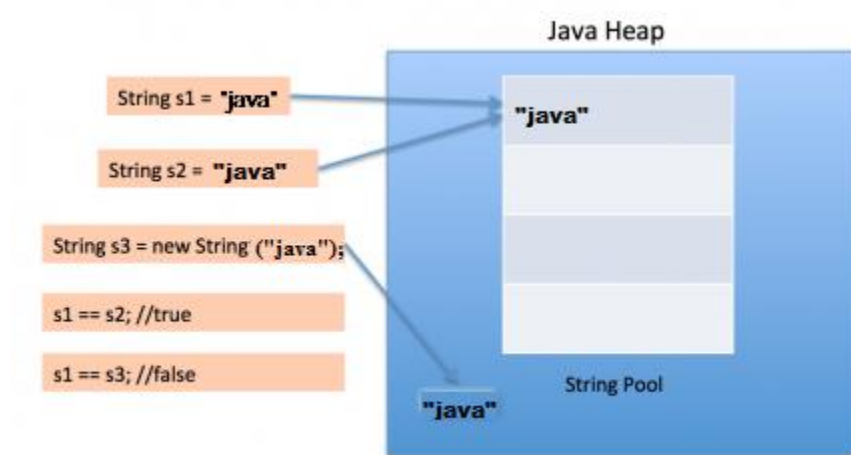
true
false

```

Explanation

We are creating a new object using new operator, and thus it gets created in a non-pool memory area of the heap. s1 is pointing to the String in string pool while s3 is pointing to the String in heap and hence, when we compare s1 and s3, the answer is false.

The following image will explain it more clearly.



By `compareTo()` method

String `compareTo()` method compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string. It compares the String based on natural ordering i.e alphabetically. Its general syntax is.

Syntax:

```

int compareTo(String str)

```

Copy

Example:

```

public class HelloWorld{

    public static void main(String[] args) {

```

```

String s1 = "Abhi";

String s2 = "ViraaJ";

String s3 = "Abhi";

int a = s1.compareTo(s2);    //return -21 because s1 < s2

System.out.println(a);

a = s1.compareTo(s3);    //return 0 because s1 == s3

System.out.println(a);

a = s2.compareTo(s1);    //return 21 because s2 > s1

System.out.println(a);

}

}

```

Copy

```

-21
0
21

```

## Java String class functions

The methods specified below are some of the most commonly used methods of the **String** class in Java. We will learn about each method with help of small code examples for better understanding.

**charAt()** method

String **charAt()** function returns the character located at the specified index.

```

public class Demo {

    public static void main(String[] args) {

        String str = "studytonight";

        System.out.println(str.charAt(2));

    }

}

```

Copy

**Output:** u

**NOTE:** Index of a String starts from 0, hence **str.charAt(2)** means third character of the String str.

**equalsIgnoreCase()** method

String **equalsIgnoreCase()** determines the equality of two Strings, ignoring their case (upper or lower case doesn't matter with this method).

```
public class Demo {

    public static void main(String[] args) {

        String str = "java";

        System.out.println(str.equalsIgnoreCase("JAVA"));

    }

}
```

Copy

```
true
```

**indexOf()** method

String **indexOf()** method returns the index of first occurrence of a substring or a character.  
indexOf() method has four override methods:

- **int indexOf(String str)**: It returns the index within this string of the first occurrence of the specified substring.
- **int indexOf(int ch, int fromIndex)**: It returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
- **int indexOf(int ch)**: It returns the index within this string of the first occurrence of the specified character.
- **int indexOf(String str, int fromIndex)**: It returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

**Example:**

```
public class StudyTonight {

    public static void main(String[] args) {

        String str="StudyTonight";

        System.out.println(str.indexOf('u'));    //3rd form

        System.out.println(str.indexOf('t', 3));    //2nd form

        String subString="Ton";

        System.out.println(str.indexOf(subString)); //1st form

        System.out.println(str.indexOf(subString,7));    //4th form

    }

}
```

Copy

```
2
```

```
11
```



5

-1

**NOTE:** -1 indicates that the substring/Character is not found in the given String.

**length()** method

String **length()** function returns the number of characters in a String.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "Count me";  
  
        System.out.println(str.length());  
  
    }  
  
}
```

Copy

8

**replace()** method

String **replace()** method replaces occurrences of character with a specified new character.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "Change me";  
  
        System.out.println(str.replace('m', 'M'));  
  
    }  
  
}  
  
;
```

Copy

Change Me

**substring()** method

String **substring()** method returns a part of the string. **substring()** method has two override methods.

1. public String substring(int begin);
2. public String substring(int begin, int end);

The first argument represents the starting point of the substring. If the **substring()** method is called with only one argument, the substring returns characters from specified starting point to the end of original string.

If method is called with two arguments, the second argument specify the end point of substring.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "0123456789";  
  
        System.out.println(str.substring(4));  
  
        System.out.println(str.substring(4,7));  
  
    }  
  
}
```

Copy

456789

456

**toLowerCase()** method

String **toLowerCase()** method returns string with all uppercase characters converted to lowercase.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "ABCDEF";  
  
        System.out.println(str.toLowerCase());  
  
    }  
  
}
```

Copy

abcdef

**toUpperCase()** method

This method returns string with all lowercase character changed to uppercase.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "abcdef";  
  
        System.out.println(str.toUpperCase());  
  
    }  
  
}
```

Copy

ABCDEF

`valueOf()` method

String class uses overloaded version of `valueOf()` method for all primitive data types and for type Object.

**NOTE:** `valueOf()` function is used to convert **primitive data types** into Strings.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        int num = 35;  
  
        String s1 = String.valueOf(num);    //converting int to String  
  
        System.out.println(s1);  
  
        System.out.println("type of num is:  
"+s1.getClass().getName());  
  
    }  
  
}
```

Copy

35

type of num is: java.lang.String

`toString()` method

String `toString()` method returns the string representation of an object. It is declared in the Object class, hence can be overridden by any java class. (Object class is super class of all java classes).

```
public class Car {  
  
    public static void main(String args[])  
  
    {  
  
        Car c = new Car();  
  
        System.out.println(c);  
  
    }  
  
    public String toString()  
  
    {  
  
        return "This is my car object";  
  
    }  
  
}
```

```
}
```

Copy

```
This is my car object
```

Whenever we will try to print any object of class Car, its `toString()` function will be called.

**NOTE:** If we don't override the `toString()` method and directly print the object, then it would print the object id that contains some hashcode.

`trim()` method

This method returns a string from which any leading and trailing whitespaces has been removed.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "    hello  ";  
  
        System.out.println(str.trim());  
  
    }  
  
}
```

Copy

```
hello
```

`contains()` Method

String `contains()` method is used to check the sequence of characters in the given string. It returns true if a sequence of string is found else it returns false.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String a = "Hello welcome to studytonight.com";  
  
        boolean b = a.contains("studytonight.com");  
  
        System.out.println(b);  
  
        System.out.println(a.contains("javatpoint"));  
  
    }  
  
}
```

Copy

true  
false

endsWith() Method

String **endsWith()** method is used to check whether the string ends with the given suffix or not. It returns true when suffix matches the string else it returns false.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String a="Hello welcome to studytonight.com";  
  
        System.out.println(a.endsWith("m"));  
  
        System.out.println(a.endsWith("com"));  
  
    }  
  
}
```

Copy

true  
true

format() Method

String **format()** is a string method. It is used to the format of the given string.

Following are the format specifier and their datatype:

| Format Specifier | Data Type      |
|------------------|----------------|
| %a               | floating point |
| %b               | Any type       |
| %c               | character      |
| %d               | integer        |
| %e               | floating point |

|    |                |
|----|----------------|
| %f | floating point |
| %g | floating point |
| %h | any type       |
| %n | none           |
| %o | integer        |
| %s | any type       |
| %t | Date/Time      |
| %x | integer        |

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String a1 = String.format("%d", 125);  
  
        String a2 = String.format("%s", "studytonight");  
  
        String a3 = String.format("%f", 125.00);  
  
        String a4 = String.format("%x", 125);  
  
        String a5 = String.format("%c", 'a');  
  
        System.out.println("Integer Value: "+a1);  
  
        System.out.println("String Value: "+a2);  
  
        System.out.println("Float Value: "+a3);  
  
        System.out.println("Hexadecimal Value: "+a4);  
  
        System.out.println("Char Value: "+a5);  
  
    }  
  
}
```

Copy

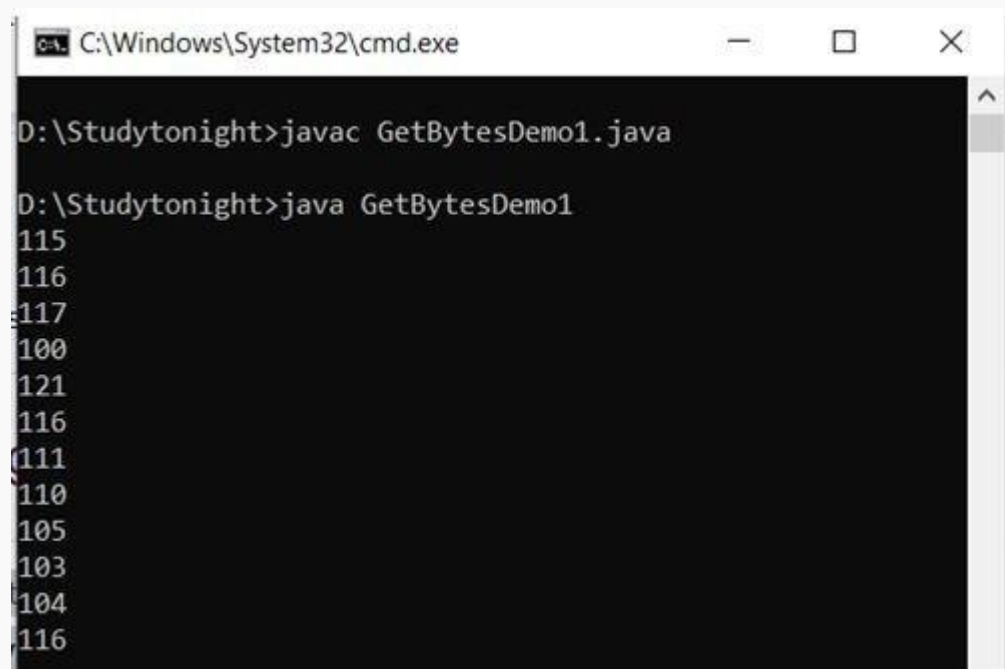
Integer Value: 125  
String Value: studytonight  
Float Value: 125.000000  
Hexadecimal Value: 7d  
Char Value: a

### `getBytes()` Method

String `getBytes()` method is used to get byte array of the specified string.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String a="studytonight";  
  
        byte[] b=a.getBytes();  
  
        for(int i=0;i<b.length;i++)  
  
        {  
  
            System.out.println(b[i]);  
  
        }  
  
    }  
  
}
```

Copy



```
C:\Windows\System32\cmd.exe  
D:\Studytonight>javac GetBytesDemo1.java  
D:\Studytonight>java GetBytesDemo1  
115  
116  
117  
100  
121  
116  
111  
110  
105  
103  
104  
116
```

### `getChars()` Method

String `getChars()` method is used to copy the content of the string into a char array.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String a= new String("Hello Welcome to studytonight.com");  
  
        char[] ch = new char[16];  
  
        try  
  
        {  
  
            a.getChars(6, 12, ch, 0);  
  
            System.out.println(ch);  
  
        }  
  
        catch (Exception ex)  
  
        {  
  
            System.out.println(ex);  
  
        }  
  
    }  
  
}
```

Copy

Welcom

`isEmpty()` Method

String `isEmpty()` method is used to check whether the string is empty or not. It returns true when length string is zero else it returns false.

```
public class IsEmptyDemo1  
  
{  
  
    public static void main(String args[])  
  
    {  
  
        String a="";  
  
        String b="studytonight";
```



```
        System.out.println(a.isEmpty());

        System.out.println(b.isEmpty());

    }

}
```

Copy

```
true
false
```

### join() Method

String **join()** method is used to join strings with the given delimiter. The given delimiter is copied with each element

```
public class JoinDemo1

{

    public static void main(String[] args)

    {

        String s = String.join("*","Welcome to studytonight.com");

        System.out.println(s);

        String date1 = String.join("/", "23", "01", "2020");

        System.out.println("Date: "+date1);

        String time1 = String.join(":", "2", "39", "10");

        System.out.println("Time: "+time1);

    }

}
```

Copy

```
Welcome to studytonight.com
Date: 23/01/2020
Time: 2:39:10
```

### startsWith() Method

String **startsWith()** is a string method in java. It is used to check whether the given string starts with given prefix or not. It returns true when prefix matches the string else it returns false.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        String str = "studytonight";  
  
        System.out.println(str.startsWith("s"));  
  
        System.out.println(str.startsWith("t"));  
  
        System.out.println(str.startsWith("study",1));  
  
    }  
}
```

Copy

true  
false  
false

String Methods List

| Method  | Description                                    |
|---|--|
| char charAt(int index)  | It returns char value for the particular index |
| int length()  | It returns string length                       |
| static String format(String format, Object... args)           | It returns a formatted string.                 |
| static String format(Locale l, String format, Object... args) | It returns formatted string with given locale. |
| String substring(int beginIndex)                              | It returns substring for given begin index.    |

|   |   |
|---|---|
| String substring(int beginIndex, int endIndex)  | It returns substring for given begin index and end index.           |
| boolean contains(CharSequence s)  | It returns true or false after matching the sequence of char value. |
| static String join(CharSequence delimiter, CharSequence... elements)                  | It returns a joined string.   |
| static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) | It returns a joined string.   |
| boolean equals(Object another)  | It checks the equality of string with the given object.             |
| boolean isEmpty()   | It checks if string is empty.                                       |
| String concat(String str)   | It concatenates the specified string.                               |
| String replace(char old, char new)  | It replaces all occurrences of the specified char value.            |
| String replace(CharSequence old, CharSequence new)                                    | It replaces all occurrences of the specified CharSequence.          |
| static String equalsIgnoreCase(String another)  | It compares another string. It doesn't check case.                  |
| String[] split(String regex)  | It returns a split string matching regex.                           |
| String[] split(String regex, int limit)   | It returns a split string matching regex and limit.                 |

|  |  |
|--|--|
| String intern()                              | It returns an interned string.                                       |
| int indexOf(int ch)                          | It returns the specified char value index.                           |
| int indexOf(int ch, int fromIndex)           | It returns the specified char value index starting with given index. |
| int indexOf(String substring)                | It returns the specified substring index.                            |
| int indexOf(String substring, int fromIndex) | It returns the specified substring index starting with given index.  |
| String toLowerCase()                         | It returns a string in lowercase.                                    |
| String toLowerCase(Locale l)                 | It returns a string in lowercase using specified locale.             |
| String toUpperCase()                         | It returns a string in uppercase.                                    |
| String toUpperCase(Locale l)                 | It returns a string in uppercase using specified locale.             |
| String trim()                                | It removes beginning and ending spaces of this string.               |
| static String valueOf(int value)             | It converts given type into string. It is an overloaded method.      |

## StringBuffer class in Java

StringBuffer class is used to create a **mutable** string object. It means, it can be changed after it is created. It represents growable and writable character sequence.

It is similar to String class in Java both are used to create string, but stringbuffer object can be changed.

So **StringBuffer** class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors.

1. **StringBuffer()**: It creates an empty string buffer and reserves space for 16 characters.
2. **StringBuffer(int size)**: It creates an empty string and takes an integer argument to set capacity of the buffer.
3. **StringBuffer(String str)**: It creates a stringbuffer object from the specified string.
4. **StringBuffer(charSequence []ch)**: It creates a stringbuffer object from the charsequence array.

Example: Creating a StringBuffer Object

In this example, we are creating string buffer object using StrigBuffer class and also testing its mutability.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        StringBuffer sb = new StringBuffer("study");  
  
        System.out.println(sb);  
  
        // modifying object  
  
        sb.append("tonight");  
  
        System.out.println(sb);    // Output: studytonight  
  
    }  
}
```

Copy

```
study  
studytonight
```

Example: difference between String and StringBuffer

In this example, we are creating objects of String and StringBuffer class and modifying them, but only stringbuffer object get modified. See the below example.

```
class Test {  
  
    public static void main(String args[])  
  
    {  
  
        String str = "study";  
  
        str.concat("tonight");  
  
        System.out.println(str);           // Output: study  
  
  
        StringBuffer strB = new StringBuffer("study");  
  
        strB.append("tonight");  
  
        System.out.println(strB);         // Output: studytonight  
  
    }  
  
}
```

Copy

```
study  
studytonight
```

Explanation:

Output is such because String objects are immutable objects. Hence, if we concatenate on the same String object, it won't be altered But StringBuffer creates mutable objects. Hence, it can be altered.

---

Important methods of StringBuffer class

The following methods are some most commonly used methods of StringBuffer class.

append()

This method will concatenate the string representation of any type of data to the end of the StringBuffer object. **append()** method has several overloaded forms.

```
StringBuffer append(String str)  
  
StringBuffer append(int n)  
  
StringBuffer append(Object obj)
```

Copy

The string representation of each parameter is appended to **StringBuffer** object.

```
public class Demo {  
  
    public static void main(String[] args) {
```

```
        StringBuffer str = new StringBuffer("test");

        str.append(123);

        System.out.println(str);

    }

}
```

Copy

test123

insert()

This method inserts one string into another. Here are few forms of `insert()` method.

```
StringBuffer insert(int index, String str)

StringBuffer insert(int index, int num)

StringBuffer insert(int index, Object obj)
```

Copy

Here the first parameter gives the index at which position the string will be inserted and string representation of second parameter is inserted into **StringBuffer** object.

```
public class Demo {

    public static void main(String[] args) {

        StringBuffer str = new StringBuffer("test");

        str.insert(2, 123);

        System.out.println(str);

    }

}
```

Copy

test123

reverse()

This method reverses the characters within a **StringBuffer** object.

```
public class Demo {

    public static void main(String[] args) {

        StringBuffer str = new StringBuffer("Hello");

        str.reverse();

    }

}
```

```
        System.out.println(str);  
    }  
}
```

Copy

olleH

replace()

This method replaces the string from specified start index to the end index.

```
public class Demo {  
    public static void main(String[] args) {  
        StringBuffer str = new StringBuffer("Hello World");  
        str.replace( 6, 11, "java");  
        System.out.println(str);  
    }  
}
```

Copy

Hello java

capacity()

This method returns the current capacity of **StringBuffer** object.

```
public class Demo {  
    public static void main(String[] args) {  
        StringBuffer str = new StringBuffer();  
        System.out.println( str.capacity() );  
    }  
}
```

Copy

16

**Note:** Empty constructor reserves space for 16 characters. Therefore the output is 16.

---

ensureCapacity()



This method is used to ensure minimum capacity of **StringBuffer** object.

If the argument of the `ensureCapacity()` method is less than the existing capacity, then there will be no change in existing capacity.

If the argument of the `ensureCapacity()` method is greater than the existing capacity, then there will be change in the current capacity using following rule: **newCapacity = (oldCapacity\*2) + 2**.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        StringBuffer str = new StringBuffer();  
  
        System.out.println( str.capacity()); //output: 16 (since  
empty constructor reserves space for 16 characters)  
  
        str.ensureCapacity(30); //greater than the existing capacity  
  
        System.out.println( str.capacity()); //output: 34 (by  
following the rule - (oldcapacity*2) + 2.) i.e (16*2)+2 = 34.  
  
    }  
  
}
```

Copy

16

34

## Java StringBuilder class

StringBuilder is identical to StringBuffer except for one important difference that it is not synchronized, which means it is not thread safe.

StringBuilder also used for creating string object that is mutable and non synchronized. The StringBuilder class provides no guarantee of synchronization. StringBuffer and StringBuilder both are mutable but if synchronization is not required then it is recommend to use StringBuilder class.

This class is located into `java.lang` package and signature of the class is as:

```
public final class StringBuilder  
  
extends Object  
  
implements Serializable, CharSequence
```

Copy

---

### **StringBuilder Constructors**

1. **StringBuilder ()**: creates an empty StringBuilder and reserves room for 16 characters.
2. **StringBuilder (int size)**: create an empty string and takes an integer argument to set capacity of the buffer.

- 3. **StringBuilder** (String str): create a StringBuilder object and initialize it with string str.
- 4. **StringBuilder** (CharSequence seq): It creates stringbuilder object by using CharSequence object.

Creating a StringBuilder Class

Lets use StringBuilder class to create string object and check its mutability also.

```
public class Demo {  
  
    public static void main(String[] args) {  
  
        StringBuilder sb = new StringBuilder("study");  
  
        System.out.println(sb);  
  
        // modifying object  
  
        sb.append("tonight.com");  
  
        System.out.println(sb);  
  
    }  
  
}
```

Copy

study

studytonight.com

Difference between StringBuffer and StringBuilder class

| StringBuffer class   | StringBuilder class  |
|--|--|
| StringBuffer is synchronized.  | StringBuilder is not synchronized.                           |
| Because of synchronisation, StringBuffer operation is slower than StringBuilder. | StringBuilder operates faster.                               |
| StringBuffer is thread-safe  | StringBuilder is not thread-safe                             |
| StringBuffer is less efficient as compare to StringBuilder                       | StringBuilder is more efficient as compared to StringBuffer. |

|                                   |   |
|-----------------------------------|---|
| Its storage area is in the heap   | Its storage area is the stack                       |
| It is mutable                     | It is mutable                                       |
| Methods are synchronized          | Methods are not synchronized                        |
| It is alternative of string class | It is more flexible as compared to the string class |
| Introduced in Java 1.0            | Introduced in Java 1.5                              |
| Its performance is moderate       | Its performance is very high                        |
| It consumes more memory           | It consumes less memory                             |

Example of StringBuffer class

```
public class BufferDemo{  
    public static void main(String[] args){  
StringBufferobj=new StringBuffer("Welcome to ");  
obj.append("studytonight.com");  
System.out.println(obj);  
    }  
}
```

Copy

Welcome to studytonight.com

Example of Stringbuilder Class

```
public class BuilderDemo{  
    public static void main(String[] args){
```

```
StringBuilderobj=new StringBuilder("Welcome to ");

obj.append("studytonight.com");

System.out.println(obj);

    }

}
```

Copy

```
Welcome to studytonight.com
```

### StringBuilder Methods

StringBuilder class has various methods to handle string object, such as append, insert, replace, reverse etc. Lets see usage of these with the help of examples.

Example of StringBuilder append string

In this example, we are appending a new string using appen() method to the existing string object.

```
public class Demo {

    public static void main(String[] args) {

        StringBuilder sb = new StringBuilder("study");

        System.out.println(sb);

        // appending object

        sb.append("tonight.com");

        System.out.println(sb);

    }

}
```

Copy

```
study
studytonight.com
```

### StringBuilder Replace Method

It is used to replace a substring from the string object. This method takes three arguments, first is start index, second is last index and third is substring to be replaced.

```
public class Demo {

    public static void main(String[] args) {

        StringBuilder sb = new StringBuilder("Java is a programming language");

        System.out.println(sb);

        // replacing object

        sb.replace( 10, 21, "computer");

        System.out.println(sb);

    }

}
```

Copy

Java is a programming language

Java is a computer language

### StringBuilder Reverse Method

It is used to reverse the string object. It completely reverses the string from start to end characters. See the below example.

```
public class Demo {

    public static void main(String[] args) {

        StringBuilder sb = new StringBuilder("Java stringbuilder");

        System.out.println(sb);

        // reverse object

        sb.reverse();

        System.out.println(sb);

    }

}
```

Copy

Java stringbuilder

redliubgnirts avaJ

# Java String tokenizer

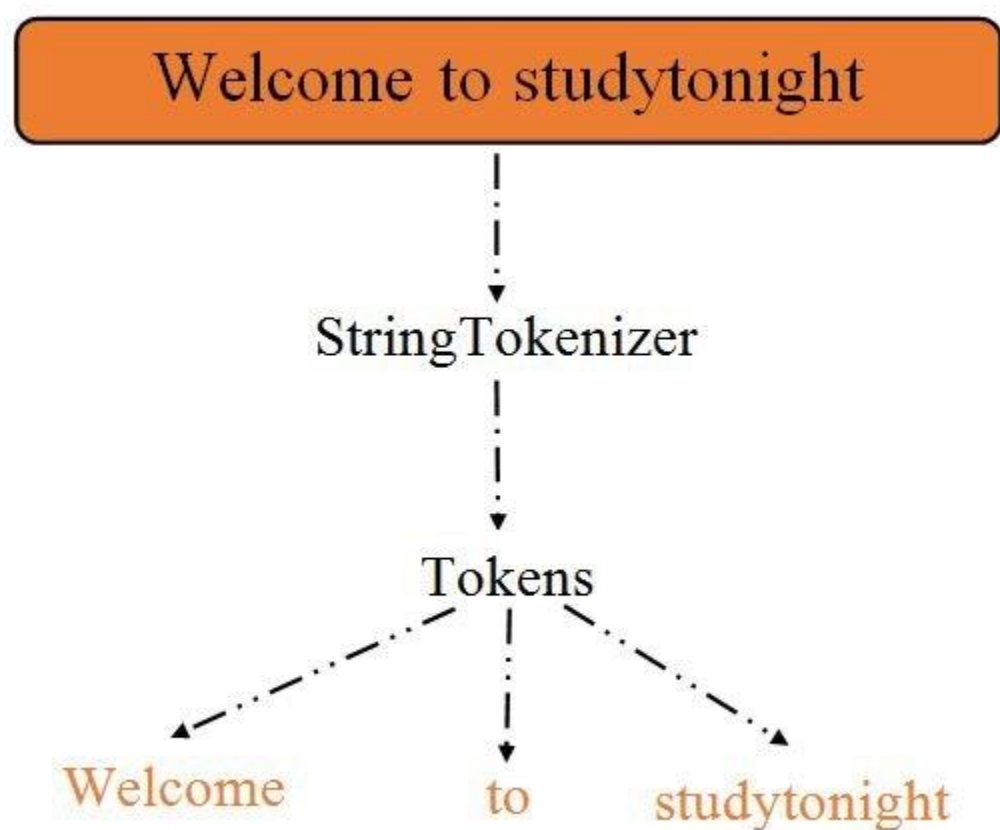
In Java, StringTokenizer is used to break a string into tokens based on provided delimiter. Delimiter can be specified either at the time of object creation or on a per-token basis.

Its object internally maintains a current position within the string to be tokenized. It is located into `java.util` package.

In string, tokenizer objects are maintained internally and returns a token of a substring from the given string.

Note: StringTokenizer is a deprecated class and available only for compatibility reasons.

We can get idea from the below image, how tokenizer breaks the string into tokens.



Following are the constructors in string tokenizer

1. `StringTokenizer(String str)`
2. `StringTokenizer(String str, String delim)`
3. `StringTokenizer(String str, String delim, booleanreturnValue)`

Following are the methods in string tokenizer

1. `booleanhasMoreTokens()`
2. `String nextToken()`
3. `String nextToken(String delim)`
4. `booleanhasMoreElements()`
5. `Object nextElement()`
6. `intcountTokens()`

Example:

In this example, we are using Stringtokenizer to break string into tokens based on space.

```

import java.util.StringTokenizer;

public class TokenDemo1

{

    public static void main(String args[])

    {

        StringTokenizerobj = new StringTokenizer("Welcome to
studytonight"," ");

        while (obj.hasMoreTokens())

        {

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

        }

    }

}

```

Copy

Welcome to studytonight

Example

Lets take another example to understand tokenizer, here we are breaking string into tokens based on the colon (:) delimiter.

```

import java.util.*;

public class TokenDemo2{

    public static void main(String args[])

    {

        String a= " : ";

        String b= "Welcome : to : studytonight : . : How : are : You
: ?";

        StringTokenizer c = new StringTokenizer(b, a);

        int count1 = c.countTokens();
    }
}

```

```
        for (inti = 0; i<count1; i++)

            System.out.println("token [" + i + "] : "

                                + c.nextToken());

        StringTokenizer d= null;

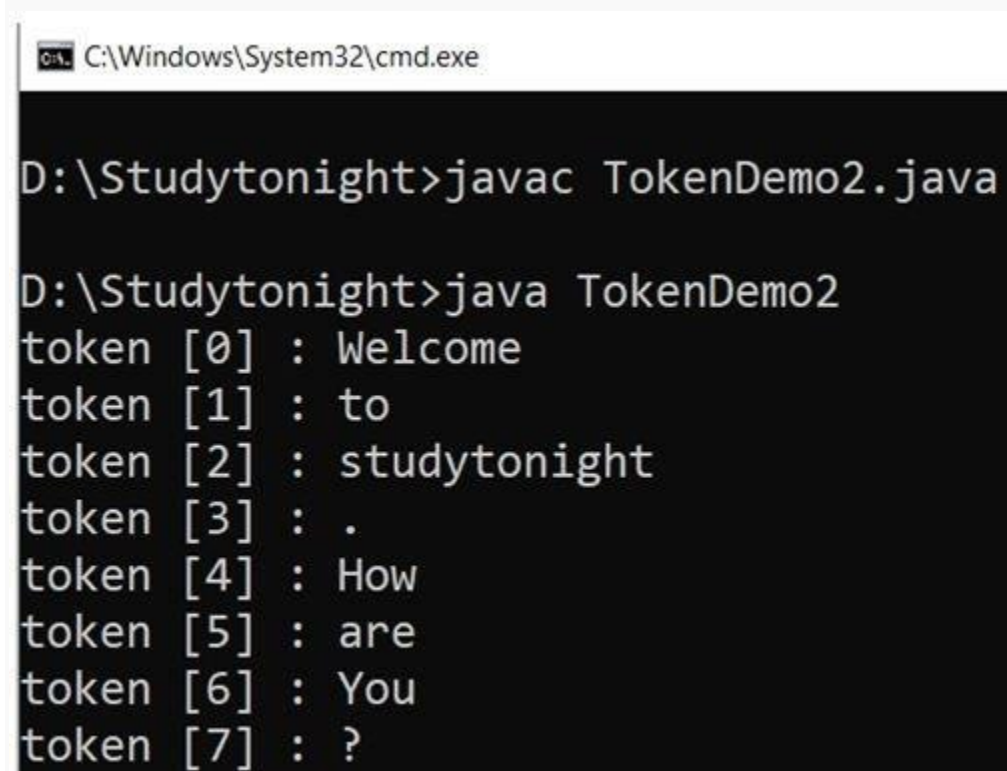
        while (c.hasMoreTokens())

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

    }

}
```

Copy



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe". The command prompt is at the directory "D:\Studytonight". The user has entered the command "javac TokenDemo2.java" to compile the Java file. Then, they entered "java TokenDemo2" to run the program. The output of the program is displayed as follows:

```
D:\Studytonight>javac TokenDemo2.java

D:\Studytonight>java TokenDemo2
token [0] : Welcome
token [1] : to
token [2] : studytonight
token [3] : .
token [4] : How
token [5] : are
token [6] : You
token [7] : ?
```