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);

String s=**new** String(“dfsf”);

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(), substring() etc.

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



### **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:

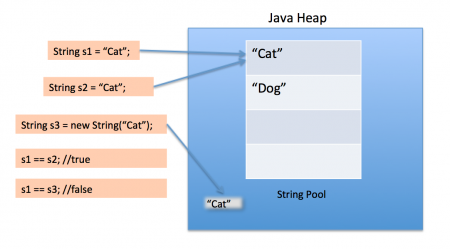
String s="welcome";

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.

#### **String objects are stored in a special memory area known as 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).

**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

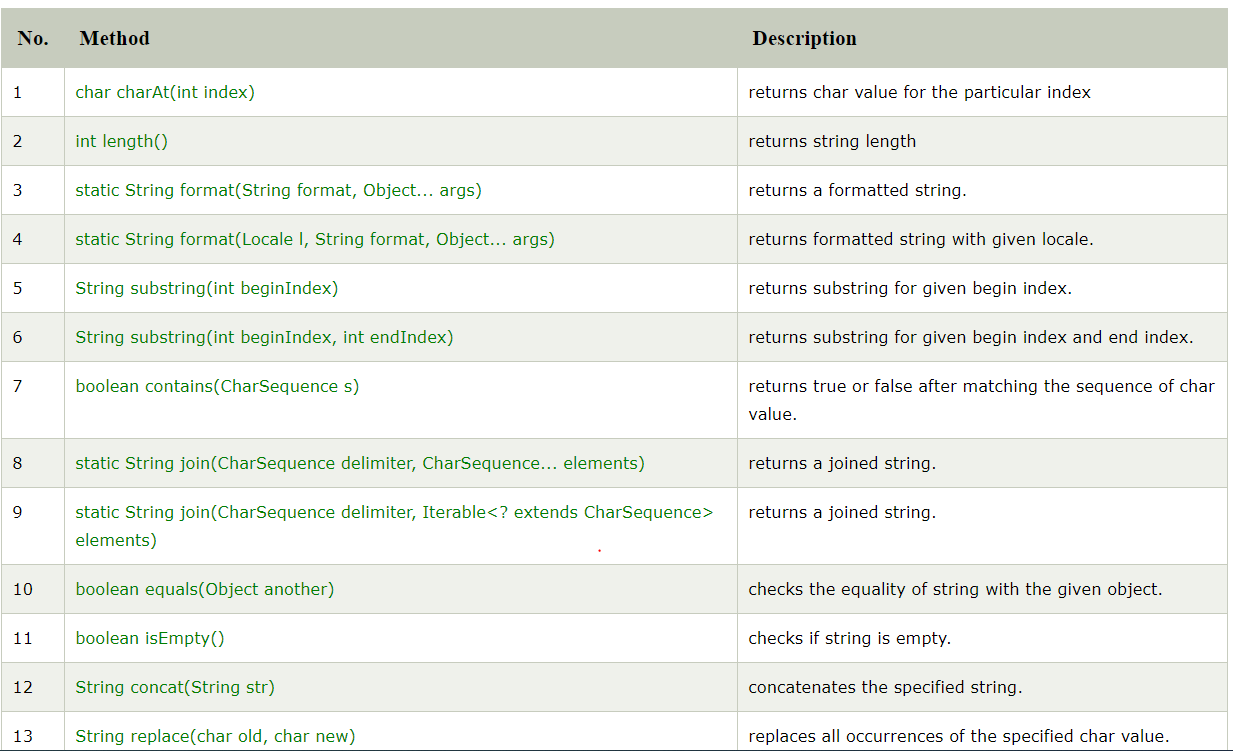
String s3=**new** String("example");//creating java string by new keyword

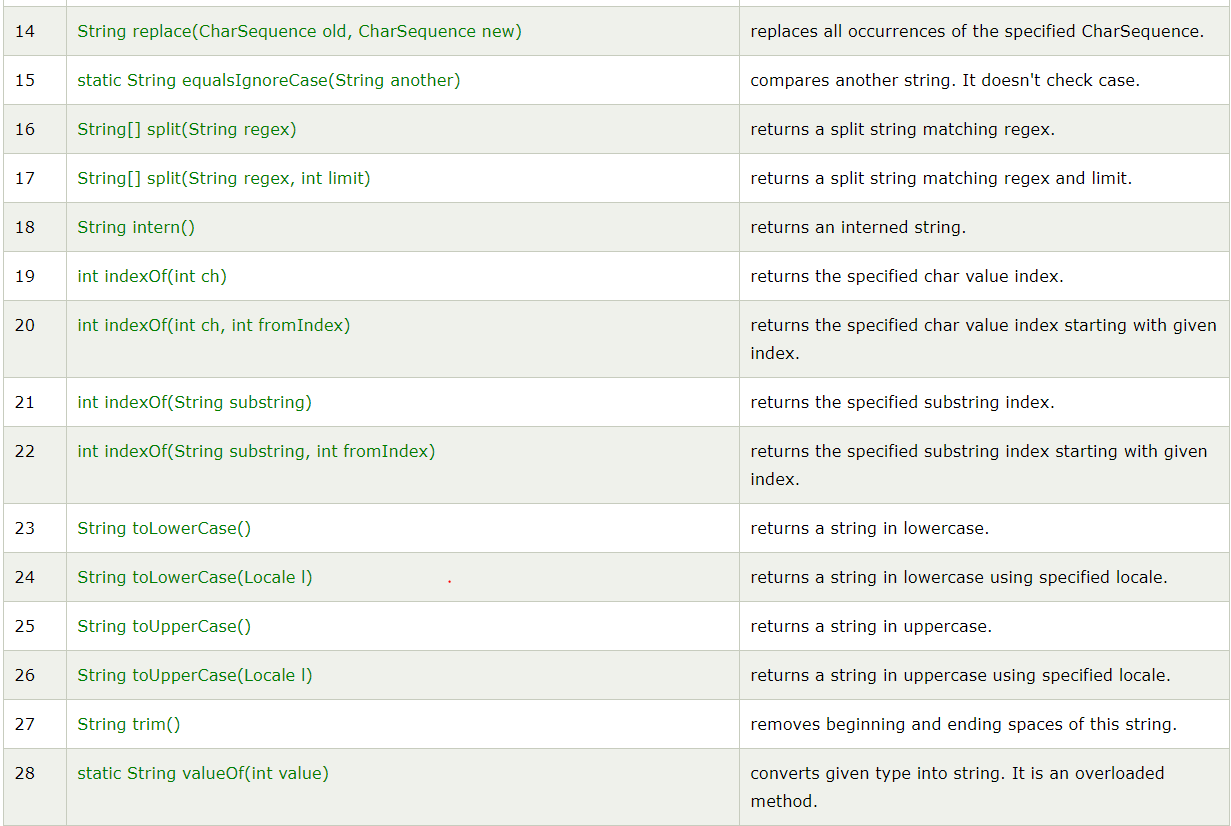
System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

}}





# Interning of String in Java

String Interning is a method of storing only one copy of each distinct String Value, which must be immutable.  
By applying String.intern() on a couple of strings will ensure that all strings having the same contents share the same memory. For example, if a name ‘Amy’ appears 100 times, by interning you ensure only one ‘Amy’ is actually allocated memory.

**public** **class** InternExample2 {

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

        String s1 = "Javatpoint";          String s2 = s1.intern();

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

        String s4 = s3.intern();

        System.out.println(s1==s2); // True

        System.out.println(s1==s3); // False

        System.out.println(s1==s4); // True

        System.out.println(s2==s3); // False

        System.out.println(s2==s4); // True

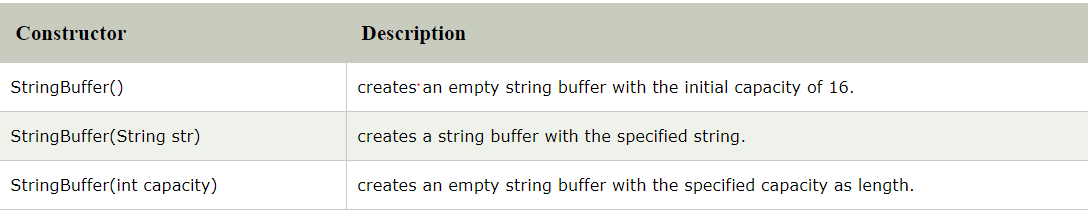
        System.out.println(s3==s4); // False

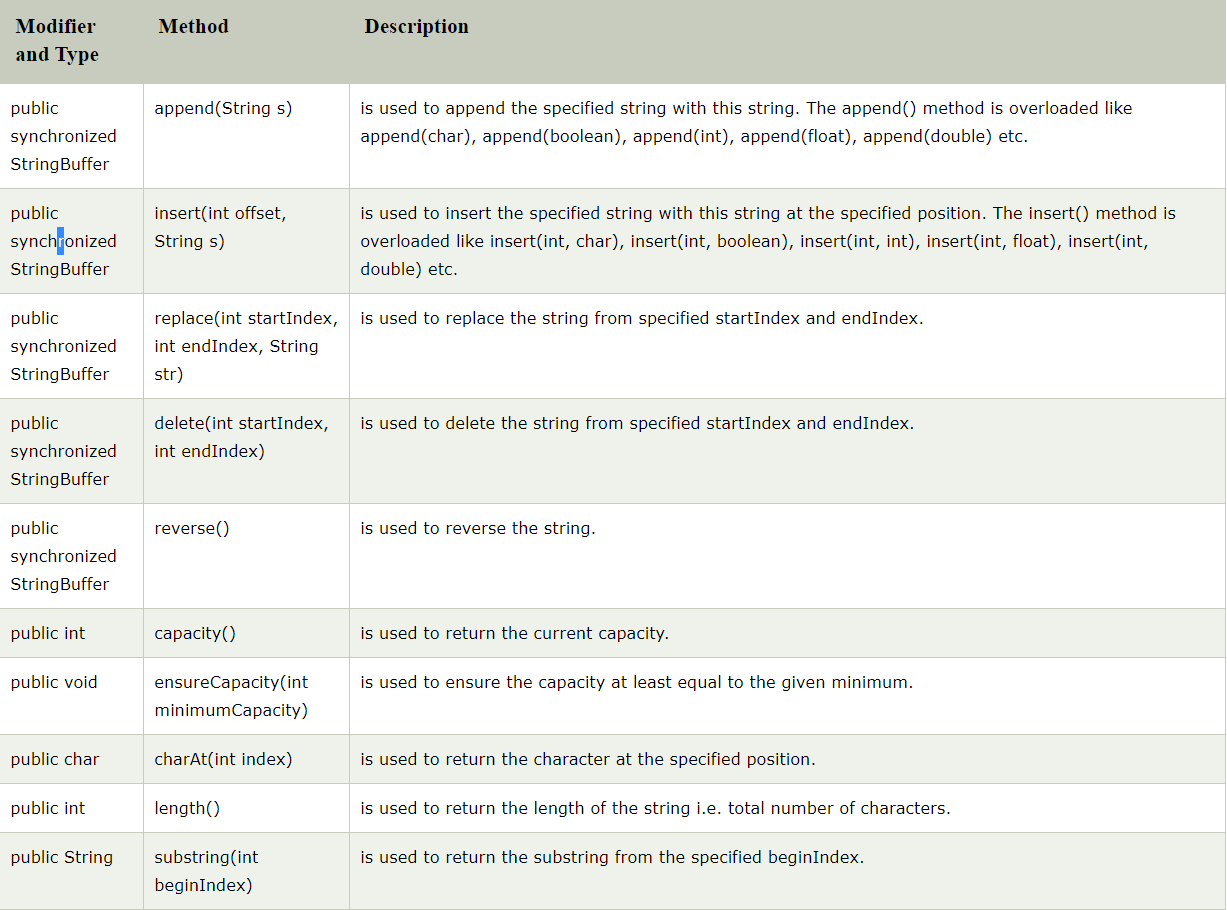
    }

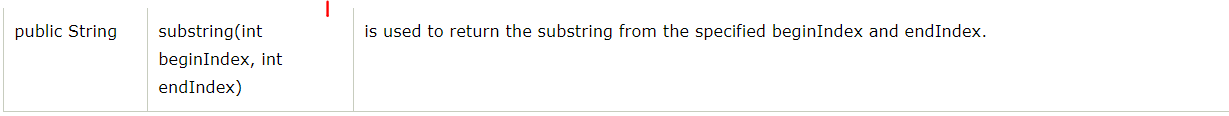
}

# Java StringBuffer class

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

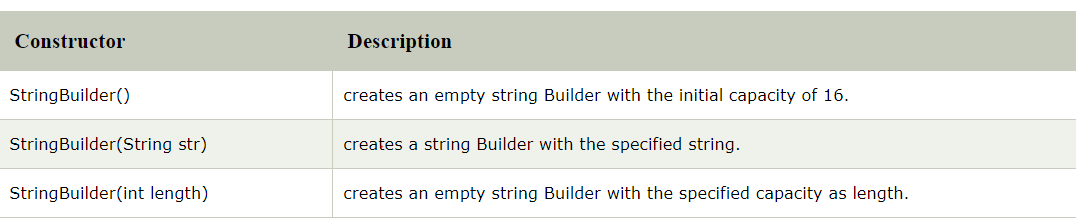
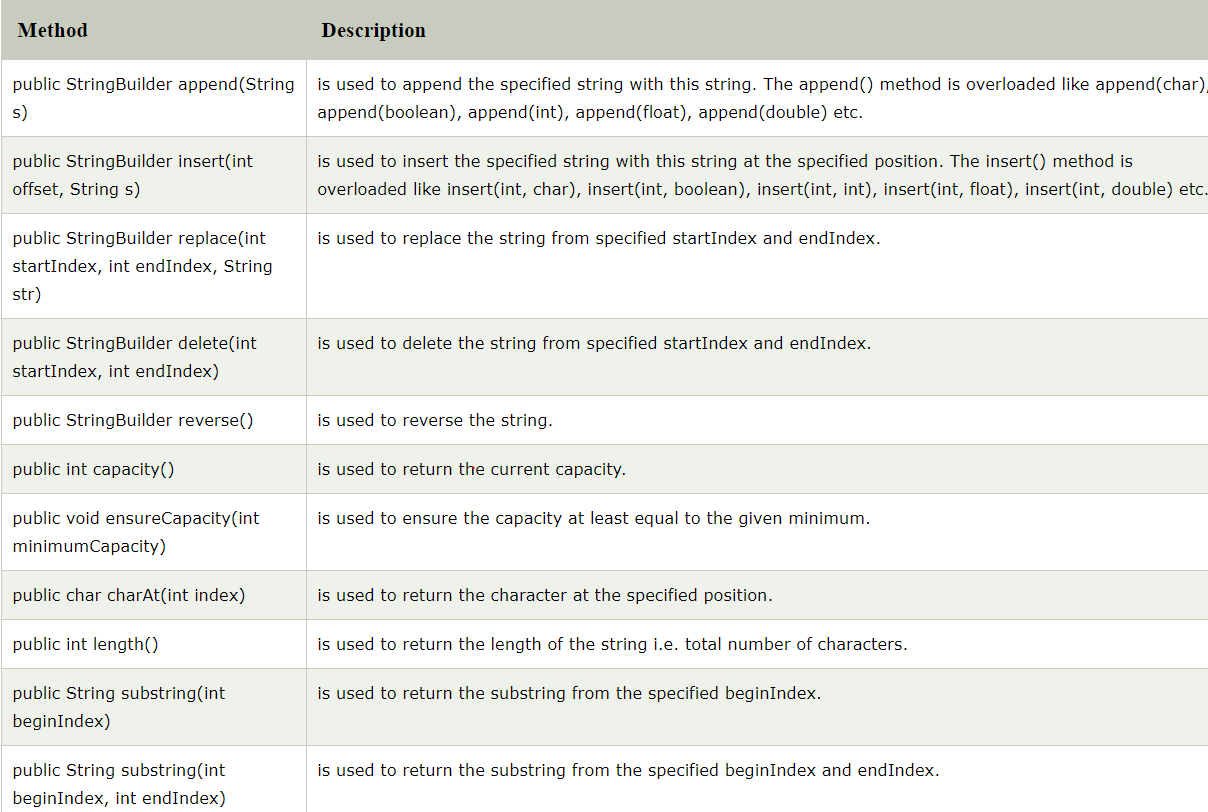


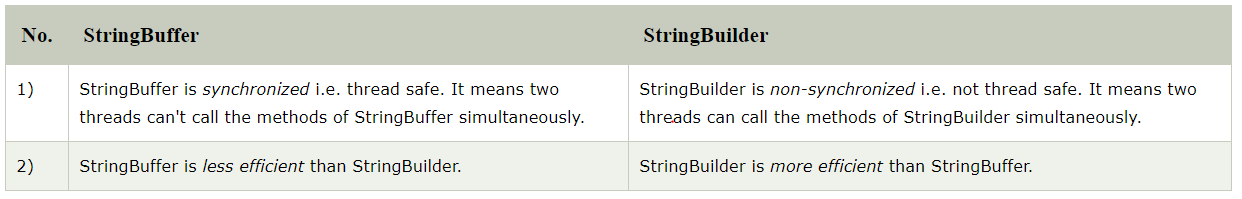




# Java StringBuilder class

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

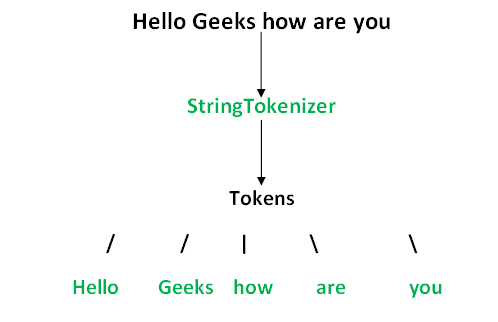
 



**String** is immutable ( once created can not be changed )object . The object created as a **String** is stored in the Constant **String** Pool. Every immutable object in **Java** is **thread safe** ,that implies **String** is also **thread safe** . **String** can not be used by two **threads** simultaneously.

# StringTokenizer class in Java with example

StringTokenizer class in Java is used to break a string into tokens.

**Example:**  


A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.  
A token is returned by taking a substring of the string that was used to create the StringTokenizer object.

**Constructors:**

**StringTokenizer(String str) :**

**str** is string to be tokenized.

Considers default delimiters like new line, space, tab,

carriage return and form feed.

**StringTokenizer(String str, String delim) :**

**delim** is set of delimiters that are used to tokenize

the given string.

**StringTokenizer(String str, String delim, boolean flag):**

The first two parameters have same meaning. The flag

serves following purpose.

If the **flag** is **false**, delimiter characters serve to

separate tokens. For example, if string is "hello geeks"

and delimiter is " ", then tokens are "hello" and "geeks".

If the **flag** is **true**, delimiter characters are

considered to be tokens. For example, if string is "hello

geeks" and delimiter is " ", then tokens are "hello", " "

and "geeks".

**Immutable:**

**Immutable objects** are instances whose state doesn’t change after it has been initialized. For example, [String](https://www.journaldev.com/16928/java-string) is an immutable class and once instantiated its value never changes

An immutable class is good for caching purposes because you don’t have to worry about the value changes.

Another benefit of immutable class is that it is inherently [**thread-safe**](https://www.journaldev.com/1061/thread-safety-in-java), so you don’t have to worry about thread safety in case of multi-threaded environment.

**Create custom immutable class**

To create an immutable class in Java, you have to do the following steps.

1. Declare the class as final so it can’t be extended.
2. Make all fields private so that direct access is not allowed.
3. Don’t provide setter methods for variables.
4. Make all **mutable fields final** so that its value can be assigned only once.
5. Initialize all the fields via a [constructor](https://www.journaldev.com/18899/constructor-in-java) performing deep copy.
6. Perform [cloning](https://www.journaldev.com/60/java-clone-object-cloning-java) of objects in the getter methods to return a copy rather than returning the actual object reference.

To understand points 4 and 5, let’s run the sample Final class that works well and values don’t get altered after instantiation.