# **Java String**

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

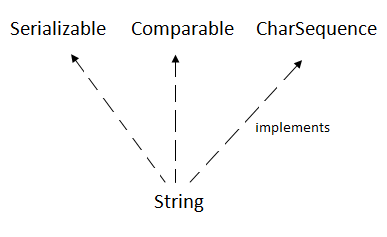
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);

is same as:

1. String s="javatpoint";

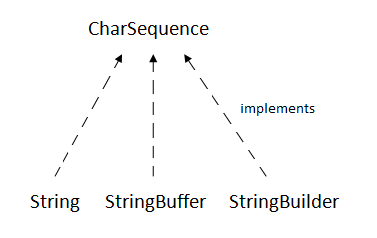
**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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



## **CharSequence Interface**

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

We will discuss about 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 string object.

### **How to create String object?**

|  |
| --- |
| There are two ways to create String object:   1. By string literal 2. By new keyword |

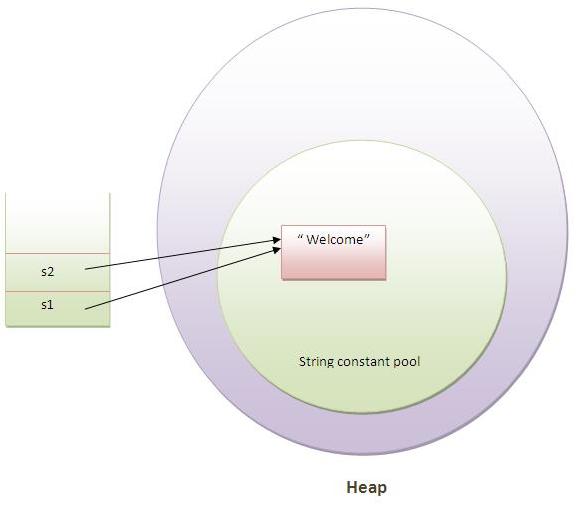
### **1) String Literal**

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 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";//will not create 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, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

#### **Note: String objects are stored in a special memory area known as string constant pool.**

### **Why java uses concept of string literal?**

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

### **2) By new keyword**

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

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

### **Java String Example**

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

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) | returns char value for the particular index |
| 2 | [int length()](https://www.javatpoint.com/java-string-length) | returns string length |
| 3 | [static String format(String format, Object... args)](https://www.javatpoint.com/java-string-format) | returns formatted string |
| 4 | [static String format(Locale l, String format, Object... args)](https://www.javatpoint.com/java-string-format) | returns formatted string with given locale |
| 5 | [String substring(int beginIndex)](https://www.javatpoint.com/java-string-substring) | returns substring for given begin index |
| 6 | [String substring(int beginIndex, int endIndex)](https://www.javatpoint.com/java-string-substring) | returns substring for given begin index and end index |
| 7 | [boolean contains(CharSequence s)](https://www.javatpoint.com/java-string-contains) | 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) | returns a joined string |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](https://www.javatpoint.com/java-string-join) | returns a joined string |
| 10 | [boolean equals(Object another)](https://www.javatpoint.com/java-string-equals) | checks the equality of string with object |
| 11 | [boolean isEmpty()](https://www.javatpoint.com/java-string-isempty) | checks if string is empty |
| 12 | [String concat(String str)](https://www.javatpoint.com/java-string-concat) | concatinates specified string |
| 13 | [String replace(char old, char new)](https://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified char value |
| 14 | [String replace(CharSequence old, CharSequence new)](https://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified CharSequence |
| 15 | [static String equalsIgnoreCase(String another)](https://www.javatpoint.com/java-string-equalsignorecase) | compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](https://www.javatpoint.com/java-string-split) | returns splitted string matching regex |
| 17 | [String[] split(String regex, int limit)](https://www.javatpoint.com/java-string-split) | returns splitted string matching regex and limit |
| 18 | [String intern()](https://www.javatpoint.com/java-string-intern) | returns interned string |
| 19 | [int indexOf(int ch)](https://www.javatpoint.com/java-string-indexof) | returns specified char value index |
| 20 | [int indexOf(int ch, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | returns specified char value index starting with given index |
| 21 | [int indexOf(String substring)](https://www.javatpoint.com/java-string-indexof) | returns specified substring index |
| 22 | [int indexOf(String substring, int fromIndex)](https://www.javatpoint.com/java-string-indexof) | returns specified substring index starting with given index |
| 23 | [String toLowerCase()](https://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase. |
| 24 | [String toLowerCase(Locale l)](https://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase using specified locale. |
| 25 | [String toUpperCase()](https://www.javatpoint.com/java-string-touppercase) | returns string in uppercase. |
| 26 | [String toUpperCase(Locale l)](https://www.javatpoint.com/java-string-touppercase) | returns string in uppercase using specified locale. |
| 27 | [String trim()](https://www.javatpoint.com/java-string-trim) | removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](https://www.javatpoint.com/java-string-valueof) | converts given type into string. It is overloaded. |

# **Immutable String in Java**

In java, **string objects are immutable**. Immutable simply means unmodifiable or unchangeable.

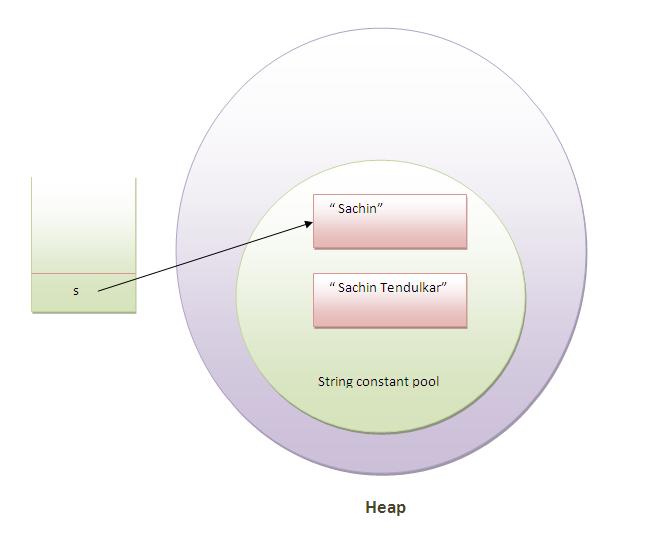
Once string object is created its data or state can't be changed but a new string object is created.

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

1. class Testimmutablestring{
2. public static void main(String args[]){
3. String s="Sachin";
4. s.concat(" Tendulkar");//concat() method appends the string at the end
5. System.out.println(s);//will print Sachin because strings are immutable objects
6. }
7. }

Output:Sachin

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



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

But if we explicitely assign it to the reference variable, it will refer to "Sachin Tendulkar" object.For example:

1. class Testimmutablestring1{
2. public static void main(String args[]){
3. String s="Sachin";
4. s=s.concat(" Tendulkar");
5. System.out.println(s);
6. }
7. }

Output:Sachin Tendulkar

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

### **Why string objects are immutable in java?**

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

# **Java StringBuffer class**

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

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

### **Important Constructors of StringBuffer class**

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

### **Important methods of StringBuffer class**

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

### **What is mutable string**

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

### **1) StringBuffer append() method**

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

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

### **2) StringBuffer insert() method**

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

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

### **3) StringBuffer replace() method**

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

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

### **4) StringBuffer delete() method**

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

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

### **5) StringBuffer reverse() method**

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

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

### **6) StringBuffer capacity() method**

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

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

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

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

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

# **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.