

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:

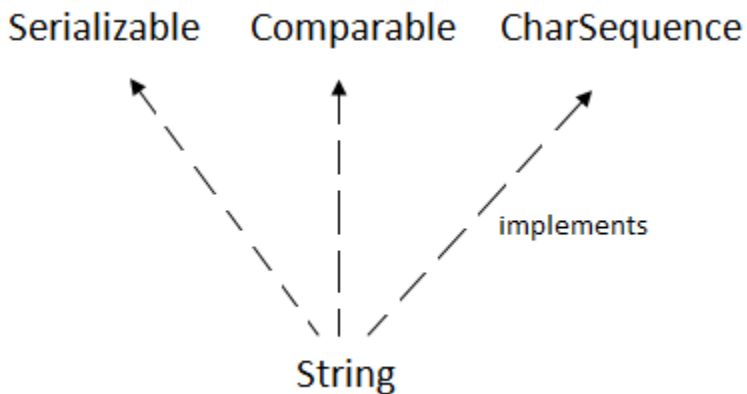
1. `char[] ch={'j','a','v','a','t','p','o','i','n','t'};`
2. `String s=new String(ch);`

is same as:

1. `String s="javatpoint";`

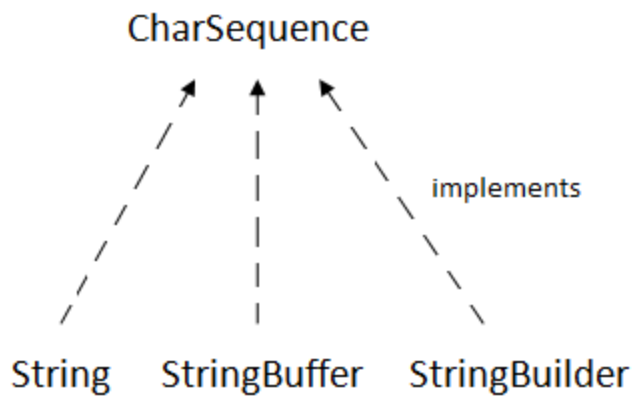
Java String class provides a lot of methods to perform operations on string such as `compare()`, `concat()`, `equals()`, `split()`, `length()`, `replace()`, `compareTo()`, `intern()`, `substring()` etc.

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

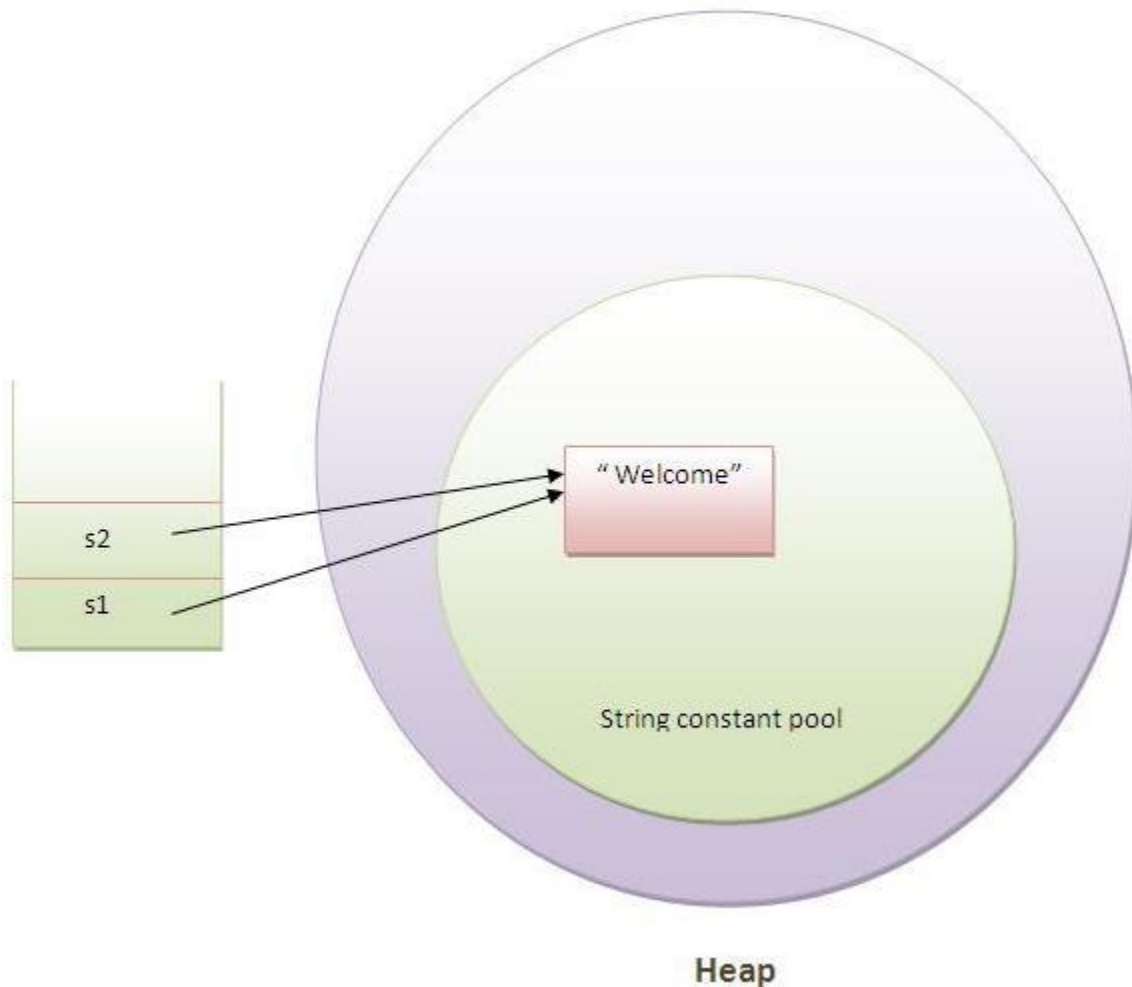
Java String literal is created by using double quotes. For Example:

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. }}

Test it Now

```
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	<u>char charAt(int index)</u>	returns char value for the particular

		index
2	<u>int length()</u>	returns string length
3	<u>static String format(String format, Object... args)</u>	returns formatted string
4	<u>static String format(Locale l, String format, Object... args)</u>	returns formatted string with given locale
5	<u>String substring(int beginIndex)</u>	returns substring for given begin index
6	<u>String substring(int beginIndex, int endIndex)</u>	returns substring for given begin index and end index
7	<u>boolean contains(CharSequence s)</u>	returns true or false after matching the sequence of char value
8	<u>static String join(CharSequence delimiter, CharSequence... elements)</u>	returns a joined string
9	<u>static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)</u>	returns a joined string
10	<u>boolean equals(Object another)</u>	checks the equality of string with object
11	<u>boolean isEmpty()</u>	checks if string is empty
12	<u>String concat(String str)</u>	concatinates specified string
13	<u>String replace(char old, char new)</u>	replaces all occurrences of specified char value
14	<u>String replace(CharSequence old, CharSequence new)</u>	replaces all occurrences of specified CharSequence
15	<u>static String equalsIgnoreCase(String another)</u>	compares another string. It doesn't check case.
16	<u>String[] split(String regex)</u>	returns splitted string matching regex
17	<u>String[] split(String regex, int limit)</u>	returns splitted string matching regex

		and limit
18	<u>String intern()</u>	returns interned string
19	<u>int indexOf(int ch)</u>	returns specified char value index
20	<u>int indexOf(int ch, int fromIndex)</u>	returns specified char value index starting with given index
21	<u>int indexOf(String substring)</u>	returns specified substring index
22	<u>int indexOf(String substring, int fromIndex)</u>	returns specified substring index starting with given index
23	<u>String toLowerCase()</u>	returns string in lowercase.
24	<u>String toLowerCase(Locale l)</u>	returns string in lowercase using specified locale.
25	<u>String toUpperCase()</u>	returns string in uppercase.
26	<u>String toUpperCase(Locale l)</u>	returns string in uppercase using specified locale.
27	<u>String trim()</u>	removes beginning and ending spaces of this string.
28	<u>static String valueOf(int value)</u>	converts given type into string. It is overloaded.

Do You Know ?

- Why String objects are immutable?
- How to create an immutable class?
- What is string constant pool?
- What code is written by the compiler if you concat any string by + (string concatenation operator)?
- What is the difference between StringBuffer and StringBuilder class?

What we will learn in String Handling ?

- Concept of String

- Immutable String
- String Comparison
- String Concatenation
- Concept of Substring
- String class methods and its usage
- StringBuffer class
- StringBuilder class
- Creating Immutable class
- toString() method
- StringTokenizer class

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.

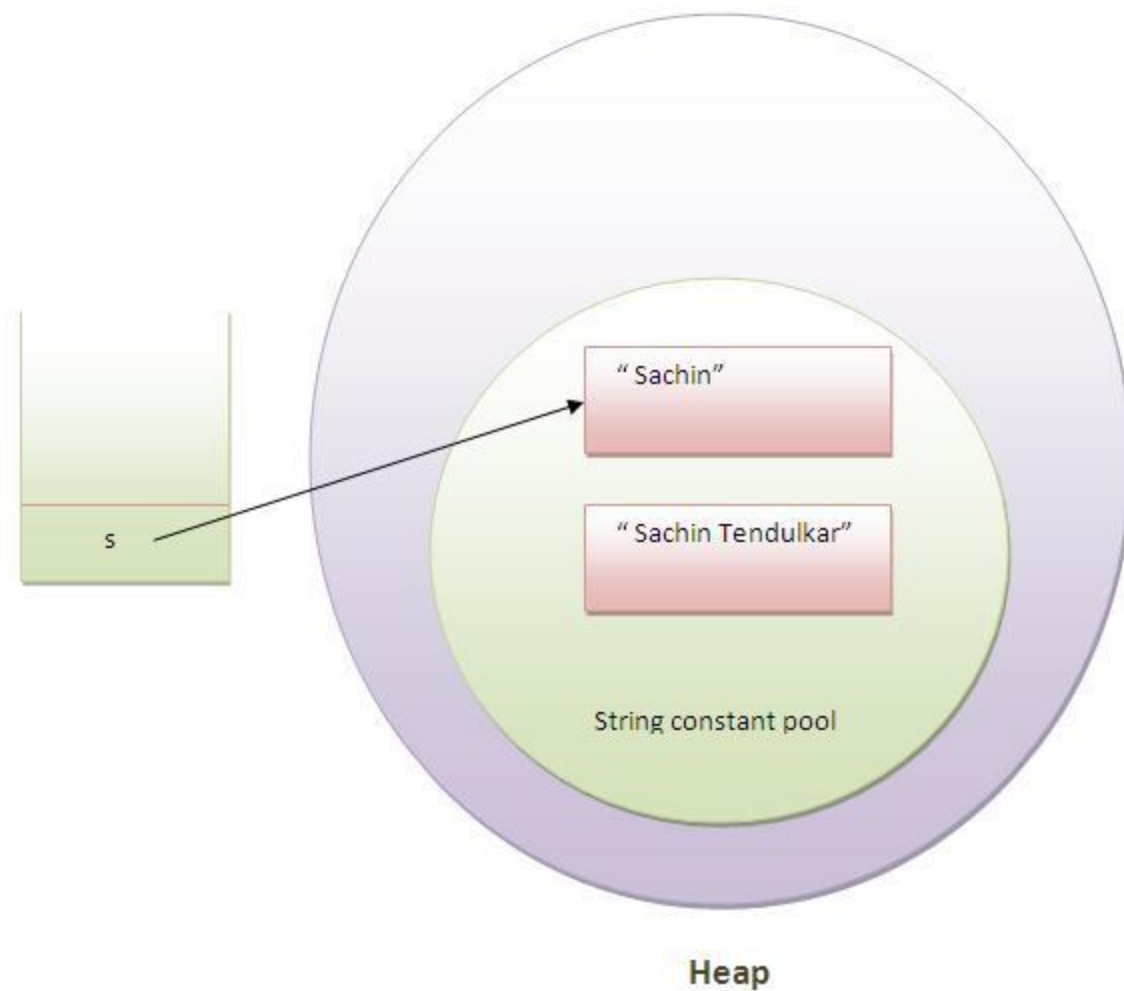
Let's try to understand the immutability concept by the example given below:

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. }

Test it Now

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 explicitly 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. }
```

Test it Now

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 refers 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 String compare

We can compare string in java on the basis of content and reference.

It is used in **authentication** (by equals() method), **sorting** (by compareTo() method), **reference matching** (by == operator) etc.

There are three ways to compare string in java:

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

1) String compare by equals() method

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides 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.

1. **class** Teststringcomparison1{
2. **public static void** main(String args[]){
3. String s1="Sachin";
4. String s2="Sachin";
5. String s3=**new** String("Sachin");
6. String s4="Saurav";

```

7. System.out.println(s1.equals(s2));//true
8. System.out.println(s1.equals(s3));//true
9. System.out.println(s1.equals(s4));//false
10. }
11. }

```

Test it Now

```

Output: true
      true
      false

```

```

1. class Teststringcomparison2{
2.   public static void main(String args[]){
3.     String s1="Sachin";
4.     String s2="SACHIN";
5.
6.     System.out.println(s1.equals(s2));//false
7.     System.out.println(s1.equalsIgnoreCase(s3));//true
8.   }
9. }

```

Test it Now

```

Output: false
      true

```

[Click me for more about equals\(\) method](#)

2) String compare by == operator

The = operator compares references not values.

```

1. class Teststringcomparison3{
2.   public static void main(String args[]){
3.     String s1="Sachin";
4.     String s2="Sachin";
5.     String s3=new String("Sachin");
6.     System.out.println(s1==s2);//true (because both refer to same instance)
7.     System.out.println(s1==s3);//false(because s3 refers to instance created in nonpool)
8.   }
9. }

```

Test it Now

```

Output: true

```

false

3) String compare by compareTo() method

The String compareTo() method compares values lexicographically and returns an integer value that describes if first string is less than, equal to or greater than second string.

Suppose s1 and s2 are two string variables. If:

- **s1 == s2** :0
- **s1 > s2** :positive value
- **s1 < s2** :negative value

```
1. class Teststringcomparison4{
2.     public static void main(String args[]){
3.         String s1="Sachin";
4.         String s2="Sachin";
5.         String s3="Ratan";
6.         System.out.println(s1.compareTo(s2));//0
7.         System.out.println(s1.compareTo(s3));//1(because s1>s3)
8.         System.out.println(s3.compareTo(s1));//-1(because s3 < s1 )
9.     }
10. }
```

Test it Now

```
Output:0
        1
        -1
```

String Concatenation in Java

In java, string concatenation forms a new string *that is* the combination of multiple strings. There are two ways to concat string 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:

1. **class** TestStringConcatenation1{
2. **public static void** main(String args[]){
3. String s="Sachin"+" Tendulkar";
4. System.out.println(s);*//Sachin Tendulkar*
5. }
6. }

Test it Now

Output:Sachin Tendulkar

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

1. String s=(**new** StringBuilder()).append("Sachin").append(" Tendulkar").toString();

In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its 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 concat not only string but primitive values also. For Example:

1. **class** TestStringConcatenation2{
2. **public static void** main(String args[]){
3. String s=**50**+**30**+"Sachin"+**40**+**40**;
4. System.out.println(s);*//80Sachin4040*
5. }
6. }

Test it Now

80Sachin4040

Note: After a string literal, all the + will be treated as string concatenation operator.

2) String Concatenation by concat() method

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

1. **public** String concat(String another)

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

1. **class** TestStringConcatenation3{

```
2. public static void main(String args[]){
3.     String s1="Sachin ";
4.     String s2="Tendulkar";
5.     String s3=s1.concat(s2);
6.     System.out.println(s3);//Sachin Tendulkar
7. }
8. }
```

Test it Now

Sachin Tendulkar