Strings in Java

What is a String?

In computer programming, a **string** is traditionally a sequence of characters, either as a literal constant or as some kind of variable. The latter may allow its elements to be mutated and the length changed, or it may be fixed (after creation). A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. *String* may also denote more general arrays or other sequence (or list) data types and structures.

Depending on the programming language and precise data type used, a variable declared to be a string may either cause storage in memory to be statically allocated for a predetermined maximum length or employ dynamic allocation to allow it to hold a variable number of elements.

When a string appears literally in source code, it is known as a string literal or an anonymous string.

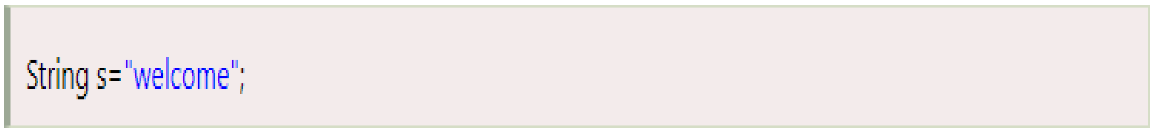
A string is generally considered as a data type and is often implemented as an array data structure of bytes (or words) that stores a sequence of elements, typically characters, using some character encoding. String may also denote more general arrays or other sequence (or list) data types and structures. 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.

There are two ways to create String object:

1. **By string literal**
2. **By new keyword**

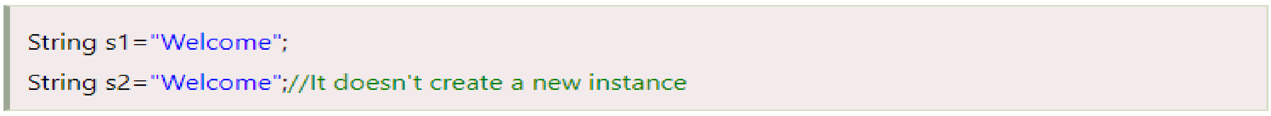
1. String Literal

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

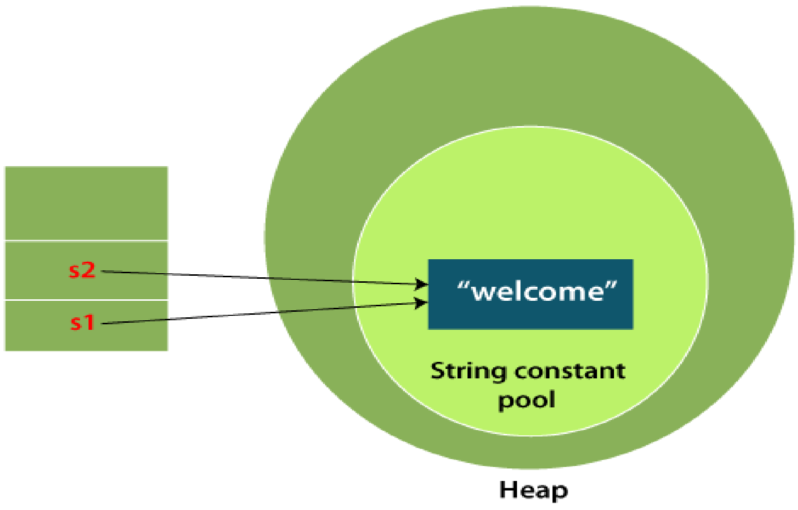


```java
String s="welcome";
```

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:



```java
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.

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

- **The Concept of String Literal is used in Java 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 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
```

The above code converts a *char* array into a String object. And displays the String objects *s1*, *s2*, and *s3* on the console using *println()* method.

Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

	STRING METHODS	**DESCRIPTION**
1.	char charAt(int index)	It returns char value for the particular index
2.	int length()	It returns string length
3.	static String format(String format, Object... args)	It returns a formatted string.
4.	static String format(Locale l, String format, Object... args)	It returns a formatted string with a given locale.
5.	String substring(int beginIndex)	It returns substring for given begin index.
6.	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index.
7.	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value.
8.	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string.
9.	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string.
10.	boolean equals(Object another)	It checks the equality of string with the given object.

11.	boolean isEmpty()	It checks if string is empty.
12.	String concat(String str)	It concatenates the specified string.
13.	String replace(char old, char new)	It replaces all occurrences of the specified char value.
14.	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence.
15	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case.
16.	String[] split(String regex)	It returns a split string matching regex.
17.	String[] split(String regex, int limit)	It returns a split string matching regex and limit.
18.	String intern()	It returns an interned string.
19.	int indexOf(int ch)	It returns the specified char value index.
20.	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index.
21.	int indexOf(String substring)	It returns the specified substring index.
22.	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index.
23.	String toLowerCase()	It returns a string in lowercase.

24	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale.
25.	String toUpperCase()	It returns a string in uppercase.
26.	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale.
27	String trim()	It removes beginning and ending spaces of this string.
28.	static String valueOf(int value)	It converts given type into string. It is an overloaded method.

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:

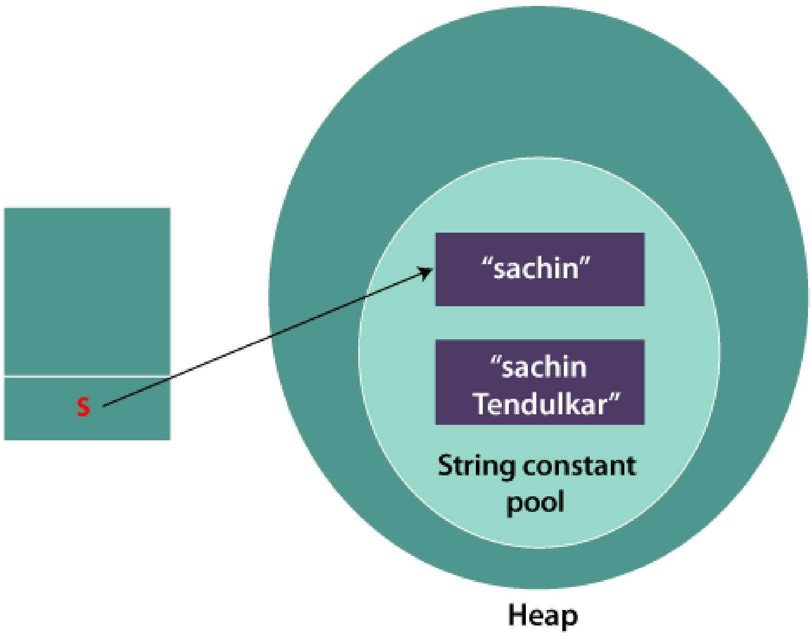
Testimmutablestring.java

```
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 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
```

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?

Java uses the concept of String literal. Suppose there are 5 reference variables, all referring 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 synchronisation 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:

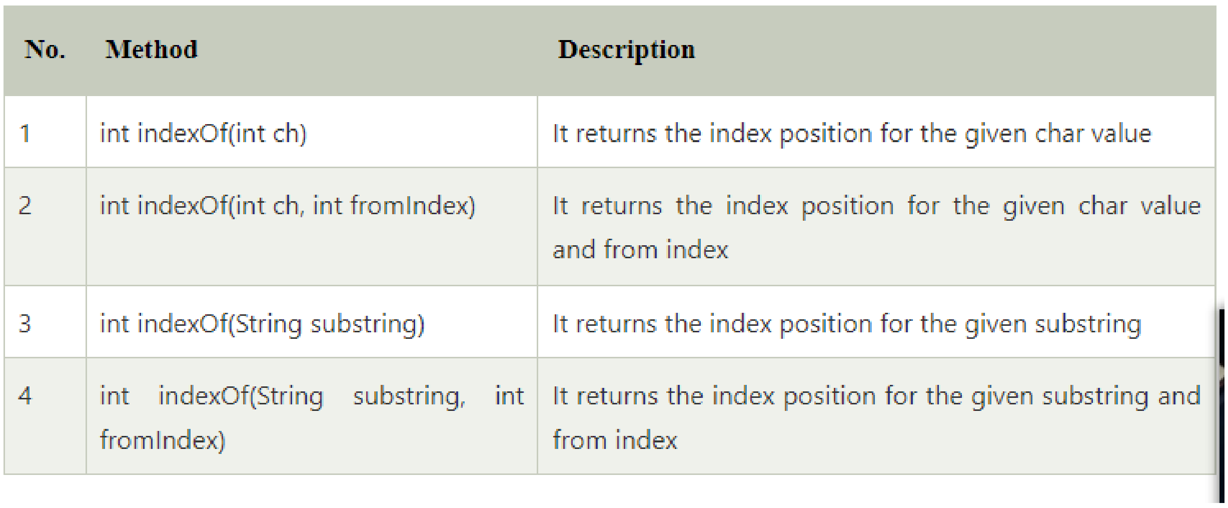
The immutability of String helps to minimise the usage in the heap memory. When we try to declare a new String object, the JVM checks whether the value already exists in the String pool or not. If it exists, the same value is assigned to the new object. This feature allows Java to use the heap space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

Java String indexOf()

The **Java String class indexOf()** method returns the position of the first occurrence of the specified character or string in a specified string.

Signature

There are four overloaded indexOf() method in Java. The signature of indexOf() methods are given below:



No.	Method	Description
1	int indexOf(int ch)	It returns the index position for the given char value
2	int indexOf(int ch, int fromIndex)	It returns the index position for the given char value and from index
3	int indexOf(String substring)	It returns the index position for the given substring
4	int indexOf(String substring, int fromIndex)	It returns the index position for the given substring and from index

Note: Strings in Java are immutable. You can read a char from a specific index with charAt(int index) but you can not modify it. For that you would need to convert to a char array as you suggested and then build a new string from the array.