String (part - 1 - Immutable String)

Index

String Introduction

In Java, strings are treated as objects; they represent a sequence of characters or char values. *java.lang.String* class is used to create strings and are always enclosed within double quote i.e., " "

Strings are of two types **Mutable** (Changeable) and **Immutable** (Unchangeable) string.

Immutable string means reference of string can be mutable but instance is immutable..

For **Mutable** String we use the **StringBuffer** and **StringBuilder class**.

Immutable String

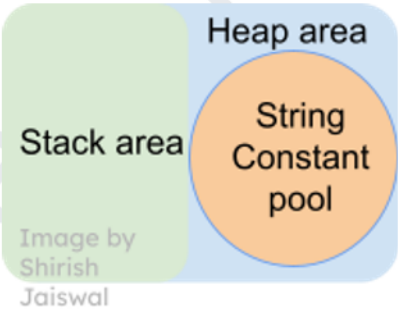
Syntax to declare the **Immutable string:**

1. Without using **new** Keyword

   String str = "Java";

   Datatype reference operator value

   one object is created in `string pool`



2. Using **new** Keyword

   String str = new String("Java");

   two object created one in `string pool` and another in `heap area`

3. From Character array to string

   char [] ch = {'j', 'a', 'v', 'a'};

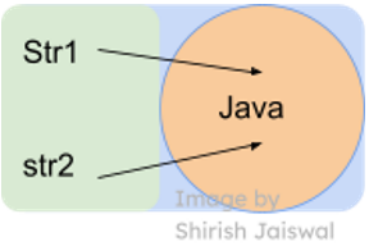
   String str = new String (ch);

Memory of every object in java will be allocated in the **Heap Area** and Strings are treated as objects.

1. Inside **Heap area** duplicates are allowed.
2. In the Heap area there is a String Constant Pool / String Pool where duplicates are not permitted.

1. Whenever we create string **without using new** keyword only **one object** is created in string constant pool
2. Whenever we create a string **using a new** keyword **two objects** are created, one inside the String constant pool and another in the Heap area.

Without using new Keyword




```
String str1 = "Java";
String str2 = "Java";
System.out.println (str1 == str2);
```

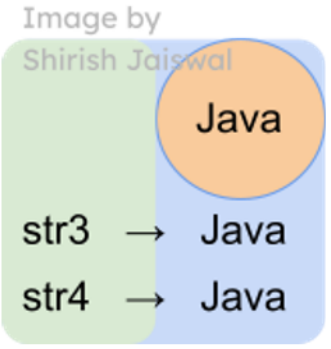
Output : **true**

Before allocating the memory JVM will scan the entire string pool to find out whether the same object already exists or not, if it exists the object will not be created and reference will refer to the already created same object.

But if it does not exist then an object will be created in the string pool.

As duplicates are not allowed in the string constant pool. In the above example both the strings are referring to the same object as they contain the same value.

Using new Keyword




```
String str3 = new String ("Java");
2 object : one in string pool and another in heap area
String str4 = new String ("Java");
1 object in heap coz string pool does not support duplicate
System.out.println (str3 == str4);
```

Output : false

Whenever we create a string using a **new** keyword two objects will be created inside the memory.

One inside **string pool** and another inside **heap area**. This happens when string **str3** is created and it will start referring to a heap area object.

But when **str4** is created, a new object is created inside the **heap area** but **not in the string pool** because string pool does not support duplicates.

str3 and **str4** will be referring to objects created inside the heap area.

As duplicates are allowed so both the references are referring to the different object inside the heap.

So the output is false because the **==** operator will not compare the value but the references where it's pointing.

Different Methods to Compare String - (examples pg. 8)

1. == Reference value will be compared

2. .equals() Actual value will be compared
   S.o.p (str1.equals (str2)); **output :** true

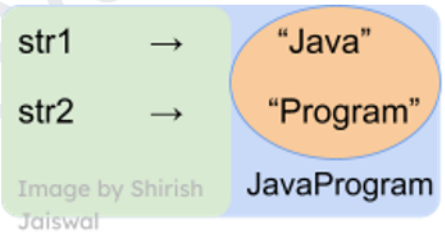
3. .equalIgnoreCase() Actual value will be compared ignoring case

4. .compareTo() Compare value lexicographically i.e., character by character
   This will compare ASCII value return **int** value.
   S.o.p (str3.compareTo (str4)); **output :** 0
   If any one of the letters is different it will stop and give the difference of ASCII value between them.

Concatenation of the strings - Joining two string

```
String str1 = "Java";
String str2 = "Program";
System.out.println (str1.concat (str2) );
```

Output : JavaProgram

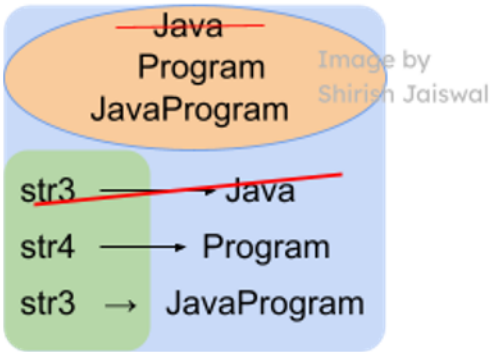


`JavaProgram` object will be created but no one will refer to it because we are not storing it.

We have studied strings are immutable but what if e.g,

```
String str3 = new String ("Java");
String str4 = new String ("Program");
str3        = str3.concat (str4);
OR  str3    = str3 + str4;
System.out.println (str3);
```

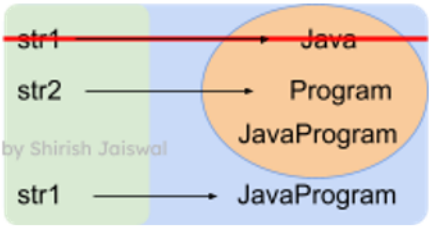
Output : JavaProgram



Oops, str3's value has changed...!!!! But strings are **immutable**...!!!

Nope, an object of value "`JavaProgram`" has been created and **`str3`** has started referring to the "`JavaProgram`" object.

```
String      str1      =      "Java";
String      str2      =      "Program";
```



What if we concat **`str1`** and **`str2`** using *.concat()* method and these strings are created by **not using new** keyword,

```
str1    =      str1.concat (str2);
```

Question : Where will the new string be stored inside the heap area or inside the String pool?

Answer : Inside the Heap area. Because whenever we are using any of the references or any of the methods the object will be stored inside the Heap area.

Note : Whenever we are using references with the methods or without methods, the memory allocation will be inside the heap area.

These will be **resolved in Run time** not in the Compile time.

But if we concate the string value :

```
str1    =      "Java"      +      "Program";
```

It will not create objects in heap area although it's performing concating operations, because it's not concating any reference value, it's concating the actual value and no references are involved while concating the actual value.

Note : The Concat method is a little slower than + operator.

```
System.out.println (str1  + str2 + 9 + 10);
```
Output : JavaProgram910

Here the addition of 9 and 10 is not performed and concatenated to string as it is.

```
System.out.println (9 + 10 + str1 + str2);
```
Output : 19JavaProgram

Here the addition of 9 and 10 is performed and then concatenated to string.

Note : If string is first then all after values will be treated as string.

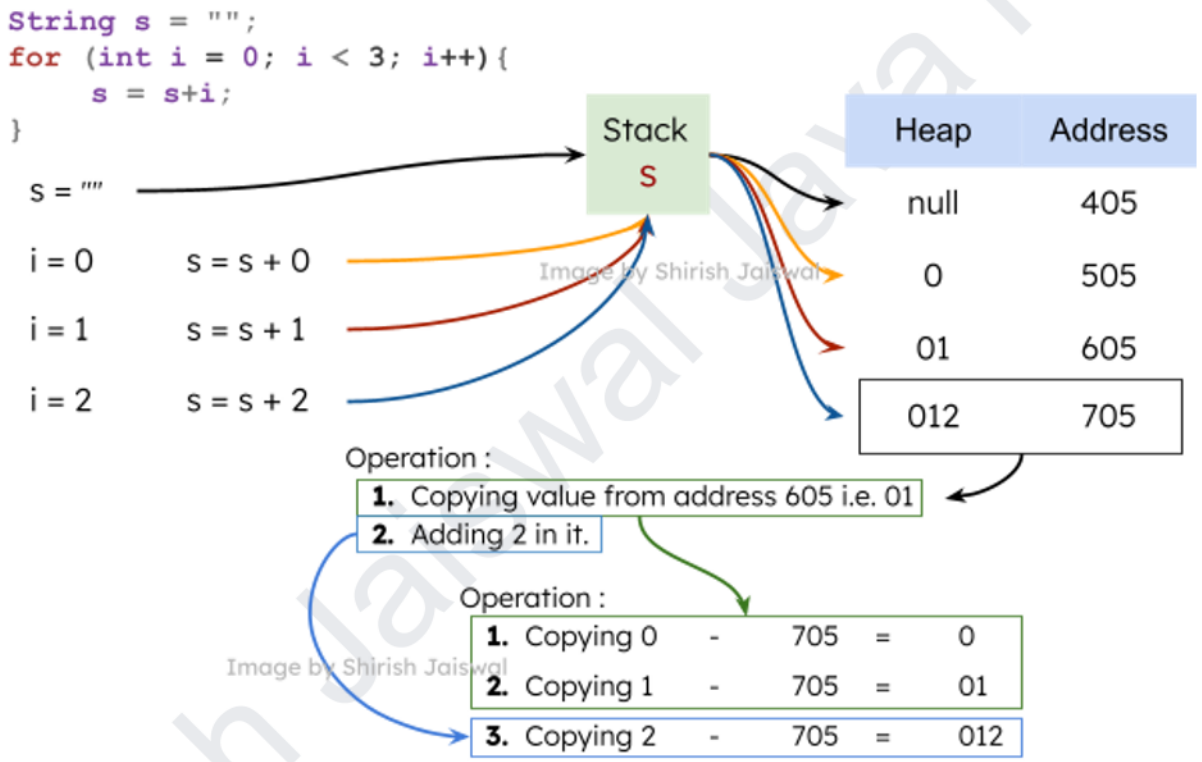
Impact of Immutability

- Space optimization
- Performance is slow

```
String s = "";
for (int i = 0; i < 3; i++){
      s = s+i;
}
```

Question : What is the Time Complexity of the above snippet? **Why?**

Answer : $O(n^2)$



So as operations of copying and concating another value to it makes TC to $O(n^2)$ as loop will run $\frac{n(n+1)}{2}$

Example if we have a String size 2GB and we want to concat a string “Apple” to it.
It will first copy that 2GB of String to the new address and then add “Apple” to it.
So it will take more time.

For this reason we have StringBuilder

String Methods

String methods	Output
`String str = "hello "; //declaration by string literal`	
length of the string `S.o.p (str.length());`	6

String to uppercase / lowercase
`S.o.p (str.toUpperCase());`
`S.o.p (str.toLowerCase());`
	HELLO
	hello

Check if string is empty `S.o.p (str.isEmpty());`	false
Check specific char whether string contains it `S.o.p (str.contains("h"));`	true
Get index of the char `S.o.p (str.indexOf("h"));`	1
To replace any character inside string `S.o.p (str.replace("LL", "YY"));`	heYYo

To check whether a string starts/ends with char. generally used for username
`S.o.p (str.startsWith("#@"));`
`S.o.p (str.endsWith("Lo "));`
	false
	true

To trim white spaces inside the string
`S.o.p (str.trim());`
`S.o.p (str.strip());`
	heLLo

To find out character at specific index `S.o.p (str.charAt(1));`	h
To find out the character present between the specific range `S.o.p (str.substring(1, 4));`	heL
To separate every character from the string `S.o.p (Arrays.toString(str.toCharArray()));`	[, h, e, L, L, o,]
To find out int value of every character `S.o.p (Arrays.toString(str.getBytes()));`	[32, 104, 101, 76, 76, 111, 32]
To split the string by any specific string `S.o.p (Arrays.toString(str.split("e")));`	[h, LLo]

Comparing two string - example

String methods and operators	Output

```
String str1 = "heLLo ";
String str2 = "heLLo ";
```

Compare str1 and str2

```
S.o.p (str1 == str2);
```

true

This happens because of String Pool. Java sees whether the same string is already declared or not. If it's already declared it will refer to the same object, else it will create a new one.

While we create using `new` keyword ew object is created in heap which support duplicates

```
String str3 = new String("hello");
String str4 = new String("hello");
```

Compare reference `str3` and `str4`

```
S.o.p (str3 == str4);
```

false

Oops this gives false but the values of `str3` and `str4` are the same!

We need to use another method i.e.,

`Obj1.equals(obj2)`

```
S.o.p (str3.equals(str4));
```

true

Now let's create both types of String

```
String str5 = "hello";
String str6 = new String("hello");
```

String Pool
Heap area

Now we will compare both the strings: In this **str6** reference is compared in the string pool because such a value is already in it.

```
S.o.p (str5 == str6.intern());
```

true

Now we will declare the same name to two strings but with different cases.

```
String fname = "tony";
String lname = "Tony";
```

Compare fname to lname

```
S.o.p (fname == lname);
```

false

false it's right. But we humans, for us it's the same so what to do? we can use `equalsIgnorcase()` method

```
S.o.p (fname.equalsIgnoreCase(lname));
```

true