String in Java is immutable

Lakshini Kuganandamurthy · Follow

Published in Nerd For Tech · 7 min read · Apr 10, 2022

Java String

Ok, I know most of you all know this!!! I knew it too!!! But did we correctly understand the term *"immutable"* and do we know ***why Strings are immutable in Java???***

Ok, let's dig in!!!

What is immutability???

In simple terms, *Once an object is created, it cannot be modified or changed.* A value and an object are two different terms in Java. *Value* is what *an instance variable or a reference variable holds. An instance variable holds the actual value* (based on the type) that e.g. a programmer provides. For e.g.

```
int a = 123;
float s = 65.7;
String name = "Lakshi";
```

As you already know *objects in Java* are stored in the *Heap area of the memory area in JVM*. Refer to my article on JVM architecture, for a better understanding. Therefore, a *reference variable* holds the *memory address of the object it is pointing to, as its value.* The figure below illustrates a reference variable "s".

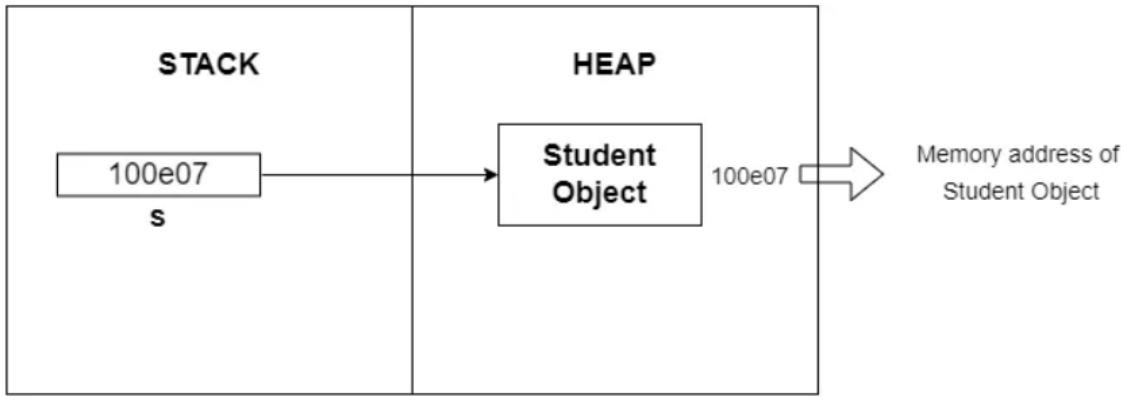


Illustration of a reference variable

Whereas an object is an instance of a class with many instance variables

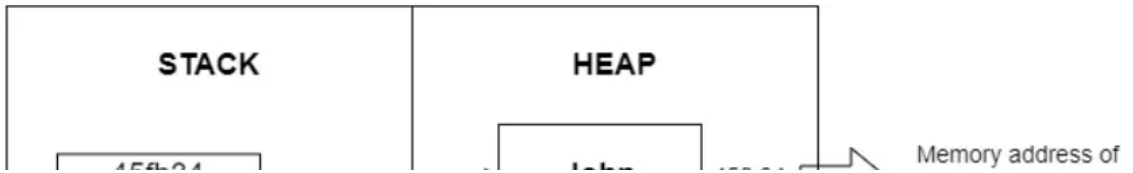
Whereas an object is an *instance of a class* with many *instance variables* generally referred to as *"properties"*.

String is a class in Java. Therefore, any variable declared with type "String" becomes a *reference variable pointing to an instance of String class (i.e. String object)*. As shown below, a *variable "name"* is *assigned the value "John"*. Later we change the *value of the variable to "Jane"*. So here, only the values of the reference variable, "name" is changed. The String object which is given the value "John" is never changed, when the "name" variable got modified to "Jane".

```
String name = "John";
name = "Jane";
```

Confusing right??? Let's dig in to see what happens in the stack and heap areas of the memory, for the code above!!!

As we said earlier, any object created in Java is stored in the Heap. When the *"name" variable is assigned "John", JVM creates an object in the Heap* and *sets its value to "John"*(because, a String value is assigned, String is a class in Java, therefore a String object gets created). The *"name" variable points to this object.* This is illustrated below.



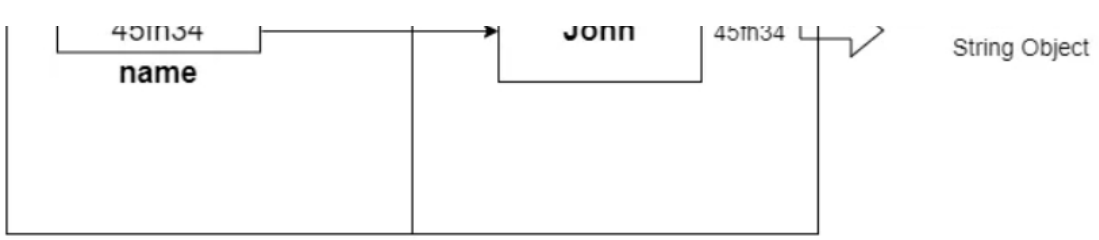
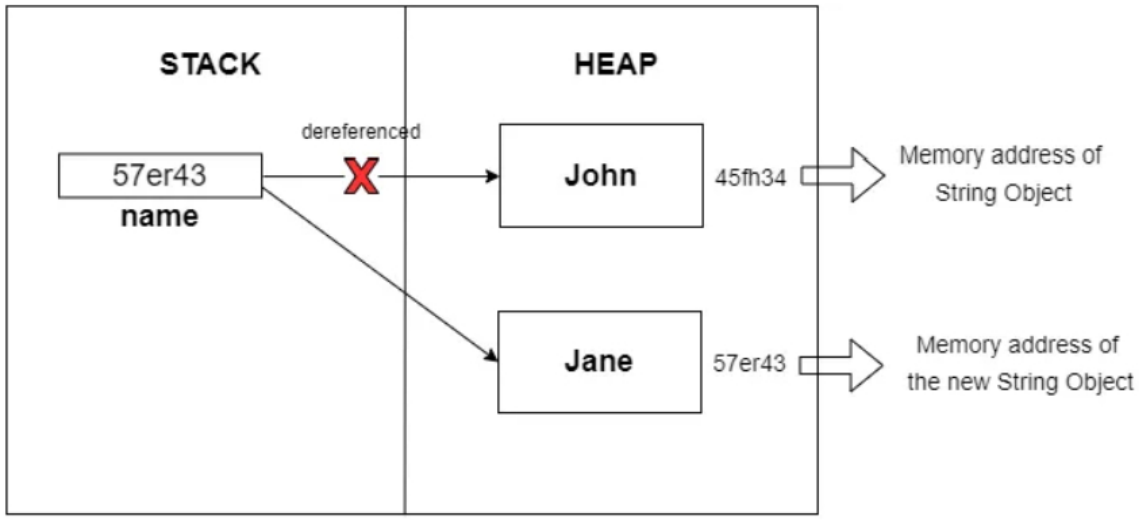


Illustration of object reference

Later, ***when "name" is assigned "Jane"***. This ***creates another String object and sets its value to "Jane"***. Now, where does the "name" variable point to??? Yes, the ***"name" variable dereferences from "John" object and points to "Jane" object***, as shown below.

Note that these are String objects which are given values "John" and "Jane".



Referencing with Objects

> *So, did we change or modify the object??? No, but a new String object was created (Jane object) and the "name" variable starts pointing to the newly*

> *created object, i.e. the "name" variable now points to a different location in the heap (a different memory address). So, this implies that whenever you modify the value of the "name" variable, it creates a new Object with that value and "name" will point to that object thereafter!!!* **This is how String in Java is immutable!!!**
>
> *Just a reminder, note that both objects have different memory addresses, in the heap.*

Let's see the reason why Java String was designed to be immutable!!!

Why Java String is immutable???

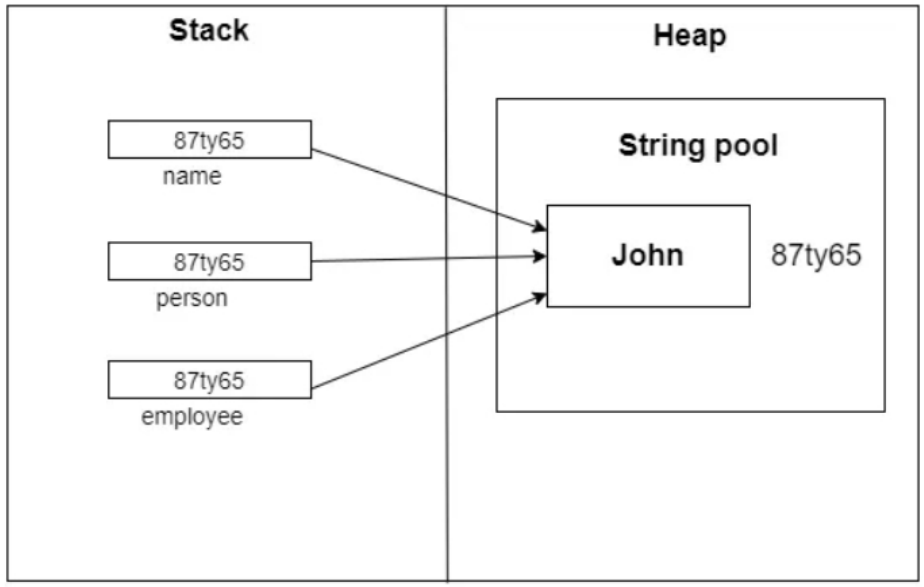
Observe the below code. These are three reference variables of type "String" having the same value "John". We compare all three variables using the "==" and .equals() function.

```
1  public class Application {
2
3      public static void main(String args[]){
4
5          String name = "John";
6          String person = "John";
7          String employee = "John";
8
9          //using the == sign
10         System.out.println(name==person); //true
11         System.out.println(person==employee); //true
12         System.out.println(name==employee); //true
13
14         System.out.println();
15         //using the .equals
```

```
16          System.out.println(name.equals(person)); //true
17          System.out.println(person.equals(employee)); //true
18          System.out.println(name.equals(employee)); //true
19      }
20  }
```

StringApplication.java hosted with ❤ by GitHub view raw

This is represented in the Heap memory as follows.



Heap representation for he above code

> **The "==" compares the reference (memory address of the object pointing to) of the variables**, *since all the variables are pointing to the same object, the* ***first 3 print statements will return "true"***
>
> **The .equals() function compares the values of the objects, these variables point to**, *since all the variables point to the same object having the*

> *value "John", **the last 3 print statements too will return "true".***

Let's say the String "John" occupies 4 bytes of heap memory. What if there were 1000 variables, assigned the value, "John". There will be 1000 objects created in the heap with the value "John". Therefore, it will occupy 4000 (4 x 1000) bytes of memory. Isn't this a waste of heap memory, objects with the same value occupying this much space!!!

Yes, that is why Java uses a ***String pool to store String literals.***

Java's String pool

A String pool is a storage area in the heap. Whenever a variable is declared with the type String (reference variable to a String object), a String object (to which the variable points) is created occupying space in the heap. Creating the same objects many times, in the heap, will reduce memory space and performance.

The string pool is maintained by the String class in Java. Therefore, the ***String class creates a pool of String literals,*** to decrease the number of String objects created in the JVM heap. Now, when a String literal is created (a String value assigned to the reference variable), JVM first checks the String pool for that literal. If the literal is already present in the pool, it returns a reference to the pooled instance (***i.e. it will return the value stored in the reference variable that is already pointing to that literal, this will make the new reference variable start pointing to the object in the pool, this way the same object can be referenced by 1000s of variables saving heap space***). If that literal is not present in the pool, JVM creates a new String Object (the literal as the

value) in the String pool. *It is also good to note that only distinct String objects are stored in the String pool!!!*

Also, note that there are 2 ways to create String Objects in the heap, in Java.

1. Using String literal, e.g.

```
String str1 = "Python";
String str2 = "Data Science";
```

2. Using the “new” keyword

```
String str4 = new String("Java");
String str5 = new String("C++");
```

. . .

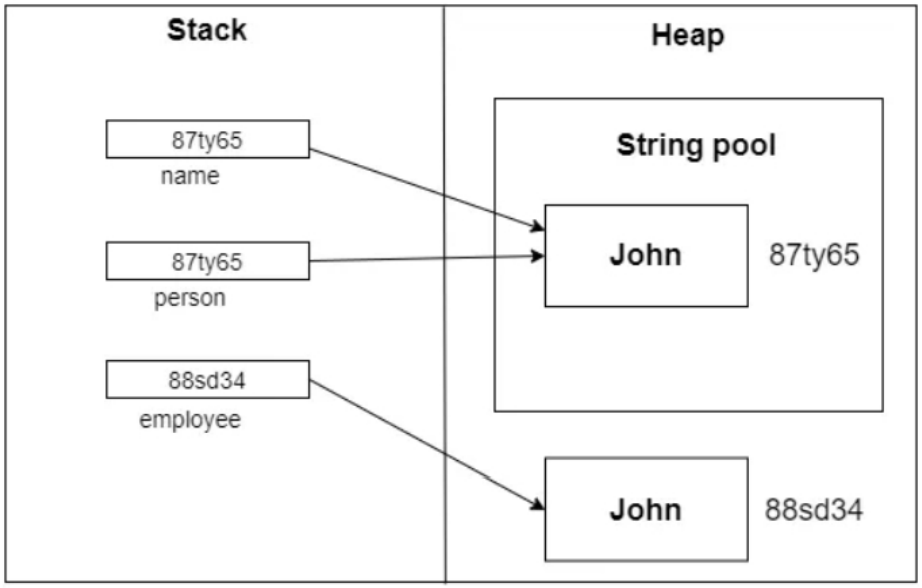
Let’s consider the below code.

```
1  public class Application {
2
3      public static void main(String args[]){
4
5          String name = "John";
6          String person = "John";
7          String employee = new String("John");
8
```

```
9           //using the == sign
10          System.out.println(name==person); //true
11          System.out.println(person==employee); //false
12          System.out.println(name==employee); //false
13
14          System.out.println();
15
16          //using the .equals
17          System.out.println(name.equals(person)); //true
18          System.out.println(person.equals(employee)); //true
19          System.out.println(name.equals(employee)); //true
20
21          System.out.println();
22
23          String str1 = "Python";
24          String str2 = "Data Science";
25          String str3 = "Python";
26
27          String str4 = new String("Java");
28          String str5 = new String("Python");
29
30          //using the == sign
31          System.out.println(str1==str3); //true
32          System.out.println(str3==str5); //false
33
34
35          System.out.println();
36
37          //using the .equals
38          System.out.println(str1.equals(str3)); //true
39          System.out.println(str1.equals(str5)); //true
40
41      }
42  }
```

Application.java hosted with ❤ by GitHub view raw

The heap and stack representations are as follows.

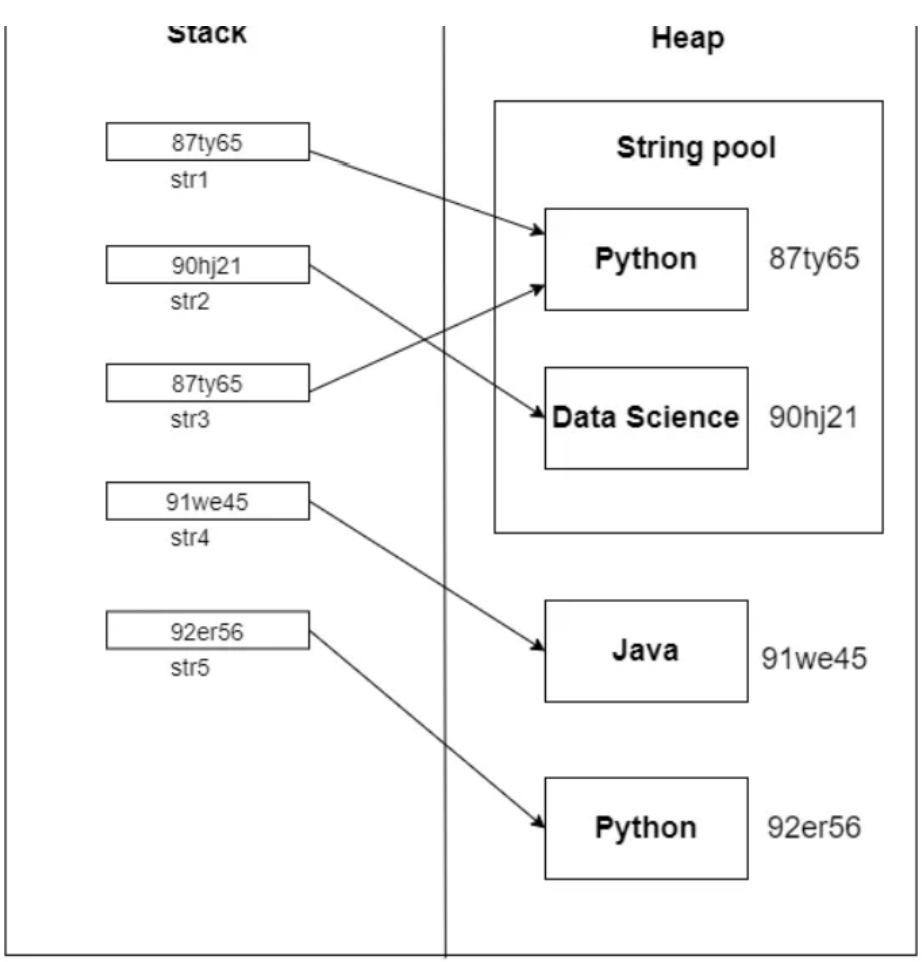


Representation of heap and Stack when the "new" keyword is used

Here, whenever the "new" keyword is used to create String objects, we are asking JVM explicitly, to create a String object. Therefore ***JVM will create a String object with the value "John", outside the String pool*** and "employee" will point to that object hereafter, as shown above.

So, when comparing "person" or "name" variables with "employee" using the "==", ***will return false because the "employee" points to a different location (reference) in the heap now.***

The .equals() function will anyway ***return true because all three variables point to objects with the same value, "John".***



Representation of Heap and Stack when the "new" keyword is used

Likewise when comparing *"str1" and "str3" using "==", will return true,* because both refer to the same object. But *"str1" or "str3" compared with "str5", will return false*, as "str5" points to the object explicitly created in the heap (using "new").

> *Note that all the String objects created with the "new" keyword take place in the heap and not in the String pool.*

> *Also as said earlier, when "str1" is assigned "Python", JVM checks the String pool, since it is not present, it will create the respective String object in the String pool. The same applies to "str2" assigned with "Data Science". Later when "str3" is assigned "Python" again, JVM checks for the "Python" object in the String pool, since it is already present, it returns the reference of the pooled instance, i.e. str1, without creating a new String object in the String pool.*

The String class is able to maintain a String pool, basically because String in Java is immutable.

Emphasize again!!!

Immutability does not mean you cannot change the value of the reference variable but means you cannot change the object created!!!

. . .

Java String is Thread-safe

Because ***String is immutable***, it is ***thread safe*** too. Let's say there is a method that takes the "name" variable as a parameter as shown below. The "name" variable points to the "John" object in the heap.

```
public class Application {

   public static void main(String args[]){
       givePromotion("John");
   }

   public void givePromotion(String name){
       //promotion code
   }
```

```
}
```

A Java program can create many threads (non-daemon) when running. Assume that two threads are accessing the same method concurrently. If Java String was not immutable, when one thread is executing the givePromotion() method, another thread could modify the value of the String object from “John” to “Jane”. The executing thread takes the “name” (pointing to the same String Object but with a different value) as “Jane” and a different employee may get the promotion. This is illustrated below.

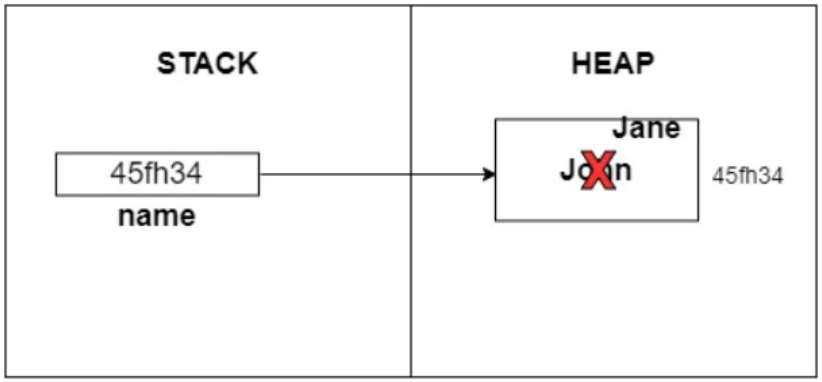


Illustration if String object is not immutable

> *Because String objects are immutable, one thread cannot modify the object created, when the other is executing the respective method. Therefore, same String object can be accessed by multiple threads concurrently.*

Phew!!! A lot to understand right!!! Take it slow and you’ll grab it!!!

Happy Learning!!!

. . .

References

1. (57) Why is immutability matters ? — YouTube
2. String Pool in Java — Javatpoint
3. What is Java String Pool? — JournalDev

Java String | Immutable | Object Reference

5

Written by Lakshini Kuganandamurthy

37 Followers · Writer for Nerd For Tech

A passionate individual eager to learn and improve. Associate Software Engineer, Virtusa.

Follow

More from Lakshini Kuganandamurthy and Nerd For Tech

Lakshini Kuganandamurthy

Daemon and Non-daemon threads in

2 min read · Apr 4, 2022

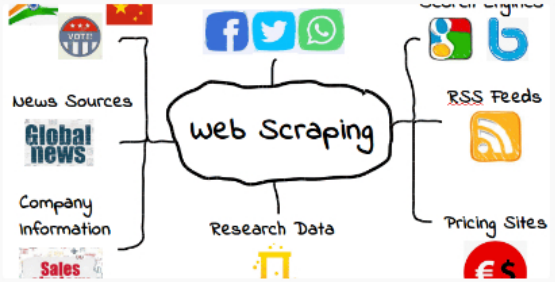
Gunjan Sahu in Nerd For Tech

Flipkart Data Engineer Interview

Flipkart Data Engineer Interview

6 min read · Jul 12

192 1



Yennhi95zz in Nerd For Tech

$4000 Freelance Income: Exploring More Than Just Web...

I've just been using Upwork for 3 months, but in that time I've completed various projects i...

· 7 min read · Jul 17

Lakshini Kuganand... in SLIIT Women In FOSS Co...

Java??? and a Deep dive into JVM

What is Java?

9 min read · Jan 31, 2022

375 2 55

See all from Lakshini Kuganandamurthy | See all from Nerd For Tech

Recommended from Medium

Aparna Rathore

Serialization in Java

Serialization in Java is the process of converting an object into a stream of bytes s...

3 min read · Apr 29

mehmoodGhaffar

List.of() Vs Arrays.asList

Understanding the Distinction between Java's List and Arrays Factory Methods

3 min read · Jun 7

9 1

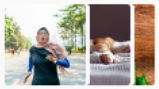
Lists

Staff Picks

467 stories · 330 saves

Stories to Help You Level-Up at Work

19 stories · 237 saves

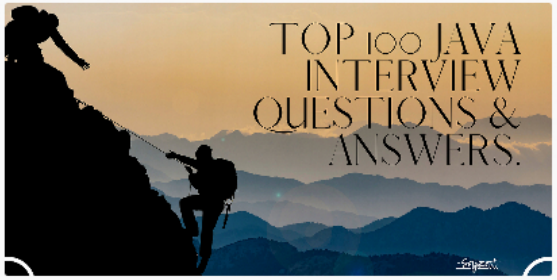


Self-Improvement 101

20 stories · 660 saves

Productivity 101

20 stories · 612 saves



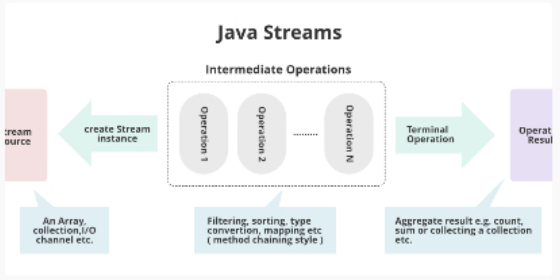
VoiceOfSilence

Top 100 Java Interview Questions & Answers.

1. What is a class in Java? Answer: Java encapsulates the codes in various classes...

17 min read · Aug 4

41



Bhaskar Sharan

Practice Java Streams Questions

If you are reading this then you should know how to work with Java Stream API. Please...

4 min read · Jun 8

29 2

Shivani kaur

Data Structures in Java: Understanding and Implementin...

Introduction

4 min read · Jun 9

Abhijeet Verma

Java Generics

The Java Generics programming is used to deal with type-safe objects. It makes the co...

5 min read · Apr 19

2

See more recommendations