Strings and StringBuilder in Java

* **What is String?**

String is basically a collection/sequence of characters.
and it is stored in String data type.

Example

$$\underbrace{\text{String}}_{\substack{\text{datatype} \\ \text{String declaration}}} \ \underbrace{\text{name}}_{\substack{\text{reference} \\ \text{variable}}} = \underbrace{\text{"Kunal Kushwaha"}}_{\substack{\text{value} \\ \text{(collection of character)}}} \overset{\text{object}}{}$$

⇒ String is the most commonly used class in the Java's class library.

$$\underbrace{\text{S}}_{\substack{\text{Everything} \\ \text{that start} \\ \text{with capital letter} \\ \text{is a class.}}}\text{tring name = "Kunal *Kushwaha"}$$

⇒ Every String that we create, it's actually an object of type String.

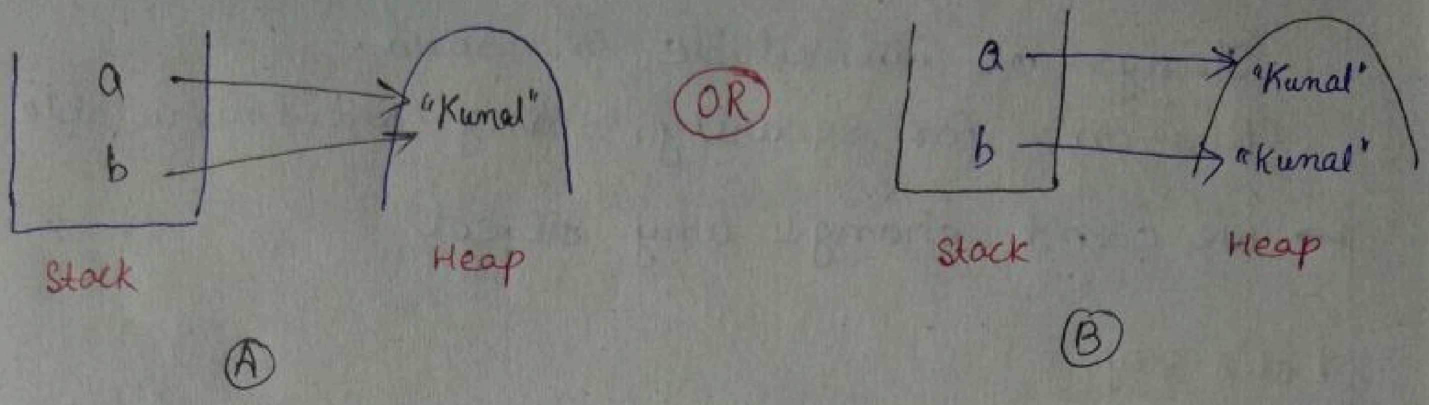
* **Internal Working of String :—**

Let say,

```
String a = "Kunal"
String b = "Kunal"
```

Q. Is this creating two different objects or is it pointing to same object?

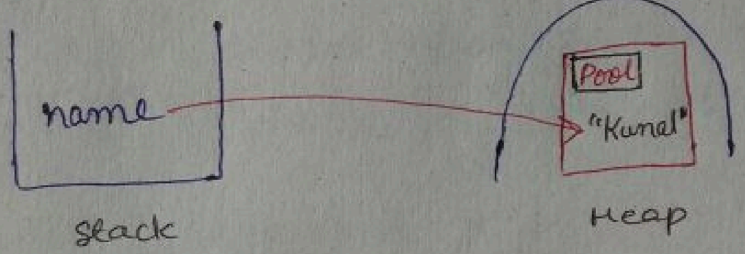
How it is stored Ⓐ or Ⓑ ?



Regarding this let's understand some concepts :-

1. String Pool :- It is a separate memory structure inside the heap.

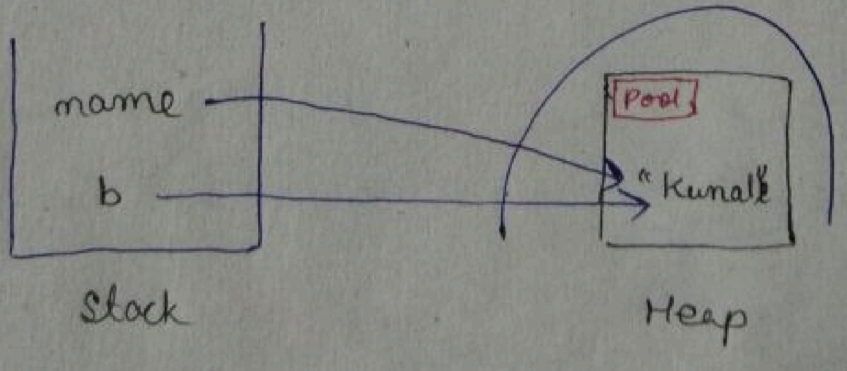
Ex - String name = "Kunal"



• Use of Pool :-

→ All the simillar values of strings are not recreated in the pool. That makes our programs more optimized.

Ex→ String name = "Kunal" ; String b = "Kunal"



Here, it says that "Kunal" already exists in the pool. So, no need to create it again. Hence, point b to "Kunal".

2. Immutability :-

→ strings are immutable in Java.

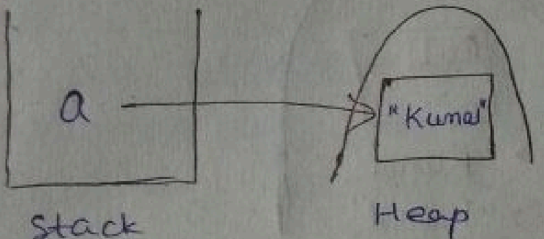
Reason :- For Security

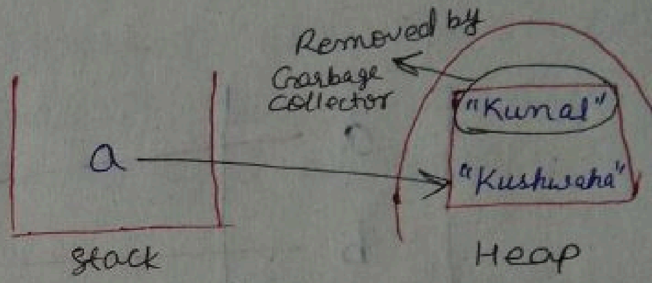
→ we can't change any object.

* Let's say :—

Initially ;- String a = "Kunal"

Then, a = "Kushwaha"





Here, we haven't change the object i.e "Kunal".
we have created a new object i.e. "Kushwaha"

* String Comparison Methods :-

① == method :-

== ⇒ a comparator

It checks ~~the~~ if the reference variables are pointing to same object

case - A

$$\begin{tikzcd} a \arrow[r] & \text{"Kunal"} \\ b \arrow[r] & \text{"Kunal"} \end{tikzcd}$$

⇒ a==b will give False

case - B

$$\begin{tikzcd} a \arrow[dr] & \\ & \text{"Kunal"} \\ b \arrow[ur] & \end{tikzcd}$$

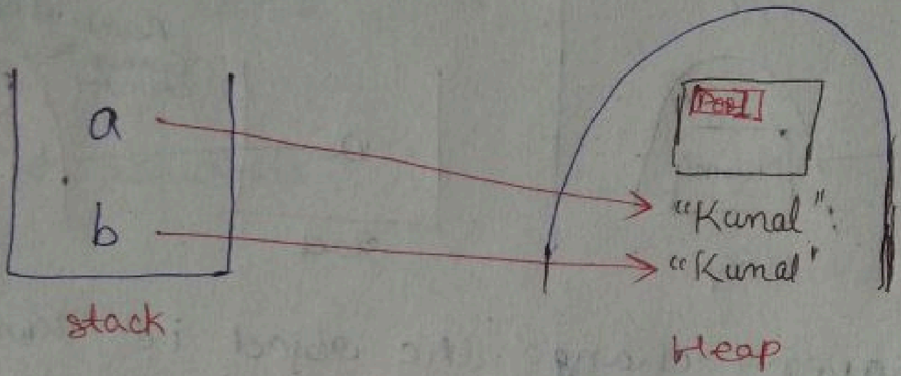
⇒ a==b will give True

* How to create different objects of same value :-

⇒ For this, we use "new" keyword.

```
String a = new String ("Kunal")
String b = new String ("Kunal")
```

// creating these values outside the pool but in heap.



NOTE :- This time a==b will give False.

② .equals method :- when we only need to check value, we use .equals method.

Ex

```
String a = new String ("Kunal");
String b = new String ("Kunal");
System.out.println (a.equals (b));
```

O/P ⇒ True
Because, it just checks the values are same or not.

* PrintStream — a class in java.
* out — a variable of type PrintStream.
* println — a method of PrintStream class.

NOTE:
Internal working of println → println calling the valueOf ~~function~~ method and that is calling toString() and then it's returning the string.

* Pretty Printing :— It prints/present the source code in an attractive way, so that it can be easily analyzed by the interpreter as well as easily read by humans.

Ex → Print the value of π till 3 digit after decimal.

$$\text{System.out.}\underbrace{\text{printf}}_{\text{Print formatted string}}(\text{"Pie: }\underbrace{\%}_{\text{placeholder}}\text{.3f", Math.PI);}$$

* System.out.println('a' + 'b');
  O/P ⇒ 195 [ASCII value of a = 97, ASCII value of b = 98]

* System.out.println("a" + "b");
  O/P ⇒ ab // String concatenation

* System.out.println('a' + 3);
  O/P ⇒ 100 [ASCII value of a = 97]

* System.out.println((char)('a' + 3));
  O/P ⇒ d

* System.out.println("a" + 1); // String "a" is not converting into its ASCII value .....

O/P ⇒ a1

<u>NOTE</u> : when an integer is added with a string it is converted to its rapper class integer. i.e, it is going to use toString().

** String Performance ** $\boxed{\text{V.V.I}}$

<u>Ex</u>

```
public static void main(String[] args) {
            String series = " ";
            for(int i = 0; i < 26; i++) {
                    char ch = (char)('a' + i);
                    series = series + ch;
            }
            System.out.println(series);
}
```

O/P ⇒ abcdefghijklmnopqrstuvwxyz

### Let's see the working of above code, And what is the problem? why it is not a very good solution?

⇒ Initially, series = " " // empty string

⇒ After 1st iteration ⇒ series = " " + 'a' = "a"

⇒ After 2nd iteration ⇒ series = "a" + 'b' = "ab"

⇒ After 3rd iter. ⇒ Series = "ab" + 'c' = "abc"

** Explaination **

⇒ So, we noticed that, new object is created everytime. It is not changing the original object as we know that strings are immutable.
So, it's actually creating new string object and copying the old one and then appending the new changes..

⇒ That's why, there is so much wastage of memory becoz, all the objects are dereferenced.
It happens like ↓

a, ab, abc, abcd, abcde, abcdef, ------------ ------ abcdefghijklmnopqrstuvwxy

All these above large strings will have ~~no~~ no reference variable. i.e, wastage of memory.

⇒ These are of size ↓
$1 + 2 + 3 + 4 + 5 + 6 + \text{-----------} + N$
$= \frac{N(N+1)}{2} = O\left(\frac{N^2 + N}{2}\right) = O(N^2)$

* Solution ⟶

StringBuilder :- It is a class just like string.
⇒ A datatype that allow us to modify the value.
⇒ It will not create a new object like string. but actually add in the original one.
i.e, StringBuilder is mutable.

```
public static void main(String[] args) {
        StringBuilder builder = new StringBuilder();
        for (int i = 0;  i < 26;  i++) {
                char ch = (char) ('a' + i);
                builder.append(ch);
        }
        System.out.println(builder.toString());
}
```

NOTE :- It gives O(N) complexity.

* String Methods :-

* toCharArray() :→ It converts the String into character array.

* length() :→ gives the length of String

* getBytes()

* toLowerCase() :→ prints the string into lowercase.

There are many more such methods ......