Strings and StringBuilder and StringBuffer

Q What are strings? Strings will be in double quotes?

- Strings are pile of a sequence of characters
- It is a datatype and it is non-primitive so, a non-primitive datatype.
- Syntax: → Everything that starts with a capital letter is a class

```
String name = "Pragni";
sout(name);
```

- It is the most commonly used class in the JAVA's class library.
- Every String that you create is actually an object of type String

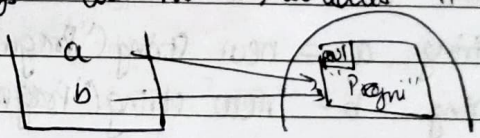
Concept

① String Pool: it is a seperate memory structure inside the heap.

Q Why seperate pool? Why not just putting it out in the heap normally like every other object?

⇒ All the similar values of strings are not recreated in the pool.

Eg:
```
String a = "Pragni";
String b = "Pragni";
```



- String pool makes our program more optimal.

Note: If you try to change this object via this reference variable it will not change for b.

Why?

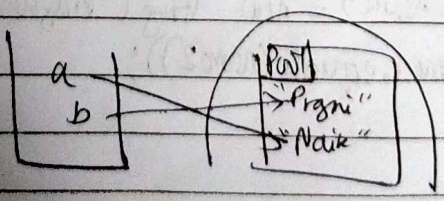
② Immutability

- Strings are immutable in Java, we can't change the object or modify object.
- If you want to rename, you have to create new object for that.
- Strings are immutable for security purpose.

In short

```
String a = "Pragni";
String b = "Pragni";
a = "Naik";
```

↳ creating new obj.



Comparing of Strings

#0 == method

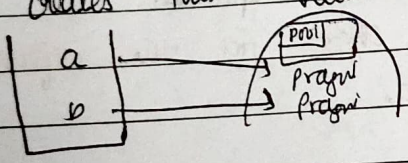
Ex: a → "Kunal" } In this case == will give false.
b → "Kunal" } 3 compares

a → "Kunal" } In this case == will give true.
b →

- Comparator actually checks for value and reference variable. If the reference variable is pointing to the same obj, it will give true.

Q. How to allocate different objects of same values?

String a = new String("Pragni")
String b = new String("Pragni")
//this creates two values outside loop but in heap



a==b //false.

//Even though values are same but these a and b are not pointing to the same object.

#2 When you only need to check value, use .equals method or function.

Ex: String name1 = new String("Pragni");
String name2 = new String("Pragni");
System.out.println(name1.equals(name2));

O/P:
true.

- Class is a name group of properties and functions

- In strings, we can't do
  sout (name1[0]);
  In arrays, we can.
- In strings, we do (index)
  sout (name1.charAt(0));
  ↳ returns character at 0th index

# Output
System.out.println(" ");
↳ variable of type PrintStream
↳ class it has method cla println in it

# Pretty Printing
float a = 453.1234f;
~~sout("Formatted number is %.2f"+a);~~
System.out.printf("Formatted number is %.2f"+a);
↳ Formatted string. (placeholder)

- % means placeholder and till how many decimal value do we want. Fo eg:- 2 ⇒ %.2f (f because it's float). Eg it also rounds off.
  O/P:- 453.12.

- For π
  sout (Math.PI);
  But you want only till 3 (decimal number)
  Then, System.out.printf("Pie: %.3f", Math.PI);
  O/P:- Pie : 3.142

- for string
  System.out.printf("Hello my name is %s and I am %s", "Yash", "student");
  O/P:- Hello my name is Yash and I am student.
  ⇒ So, the order in which you have placed the placeholders, in that order only you've to put the variables.

- Some common format specifiers:

%.c : Character
%.d : Decimal number (base 10)
%.e : Exponential floating point number
%.f : Floating-point number
%.i : Integer (Base 10).
%.o : Octal number (base 8)
%.s : String
%.u : Unsigned decimal (integer) number
%.x : Hexadecimal number (base 16)
%.t : Date/time
%.n : Newline.

String concatenation Operator:

① sout('a' + 'b');
Op: 195
Here the operator is converting character into its Ascii value and then adding it. {In single quotes ho to ye nayegi}

② sout("a" + "b");
O/p: ab
It concatenate two strings {In double quotes ho to ye nayegi}.

③ sout('a' + 3);
op: 100 {a → 97 → 97+3 = 100}.

④ sout((char)('a'+3));
Op: d.
Converted the 100 into a character (casting)

Note:

① When you are doing the addition with characters it converts into its value into a number then it use that numbers to solve the problem.

② But with string it's not doing that, it actually takes the string value.

⑤ sout("a" + 1);

O/P: a1

Note:
1) When an integer is concatenated ("added") with a string it is converted to its wrapper class Integer.
2) This is same as: "a" + "1".

⑥ sout("Kunal" + new ArrayList<>());

O/P: Kunal[]

We know this will be like an object of type integer. Hence, it's calling the toString, which is returning normal square brackets since it is empty it will return an empty array.

⑦ sout(new Integer(56) + new ArrayList<>());

O/P: error.

Operator '+' cannot be applied to integer and arraylist.

(1 expression)

- Operator '+' in Java is only defined for primitives and when any one of these values is a String.
- Operator '+' you can only used with primitives and you can only use this with all the complex objects as well. But the only condition is at least one of these object should be of type String.

Ex. sout(new Integer(56) + "" + new ArrayList<>());

(object of type string)

Here, the entire result will be of string type. O/P: 56[]

→ So, on string object the plus operator is being overloaded, because it concatenate more than one strings.

- In Java operator overloading is not supported even in software engineering but in C++ it is supported.
- So, + operator, you can basically modify what the '+' operator is doing in C++ and also in Python
- You can also make it act like as a multiplication or subtracting subtraction, you can add complex data type as well.
- But this results in poor code, that is why in Java it is not supported.
- Java has only given us operator overloading exception for strings, but you cannot do it on your own if you want to merge two complex objects of your own type like array, hashmaps, it will not allow even for the modification.
- It's the only operator that is intentionally overloaded in Java to support string concatenation or string joining.
- new Integer(): In future, it is going to be removed.

(8) sout("a" + 'a');

Op: aa

If one of the datatype is a string ans will be string.

String Performance (Imp for Interview Points)

```
String series = "";
for(int i=0; i<26; i++) {
    char ch = (char)('a'+i);
    series = series + ch;  //series += ch;
}
sout(series);
```

O/P: abcdefghijklmnopqrstuvwxyz

Loop

① series = ""

② series = "" + 'a' = "a"

③ series = "a" + 'b' = "ab"

:

series = "abcdefghijklmnopqrstuvwxy" + 'z' = "abcdefghijklmnopqrstuvwxyz"

- Here the new object is being created everytime. It is not changing the original object because the strings are immutable. Therefore, this one one copying the old one and then appending it with the new one.
- So much waste of memory because of this and it won't be having any reference variable. So, time complexity = $O(n^2)$ WORST

- SOLUTION: Wouldn't it be great if there was sort of datatype that will allow us to modify its value because strings fails to allow this.

So, the answer is :- StringBuilder.

- Here in StringBuilder one object is made and the changes are done in that object only and the reference is also the same and it will not be changed.
- StringBuilder is a separate class.

```
StringBuilder builder = new StringBuilder();
for (int i=0; i<26; i++) {
    char ch = (char)('a' + i);
    builder.append(ch);
}
sout(builder);
```

⇒ Here str is mutable. (StringBuilder)

Some methods that string provides.

① .toCharArray() :- It convert string into array to characters.

Eg: String name = "Pragni Naik";
sout(Arrays.toString(name.toCharArray()));

O/P: [P, r, a, g, n, i, , N, a, i, k]

Even space will be counted.

② .length() :- It will provide you the length.

③ .toLowerCase() : It will convert into lowercase. Its not actually going to convert the original object because strings are immutable.

④ .indexOf() : It will give the index value.

⑤ .lastIndexOf() : It will give the last index value.

⑥ .strip() :- white spaces are removed.

⑦ .split() :- add as regex first :- After adding regex it will split over there.

Ex :- sout(Arrays.toString(name.split(" ")));
regex

O/P: [Pragni, Naik]

Palindrome String Program.

original ~~string~~ => input.

reverse = " "

for (i = length; i > 0; i--) {

reverse = reverse + original.charAt(i-1);

}

if (reverse.equals(original)) {
print → Palindrome.
} else {
print → not palindrome.
}

Q) You are given two non-negative numbers 'num1' and 'num2' in the form of strings. Find the sum of both the strings.
Note: Do not convert the strings to integers directly

```
BigInteger a = new BigInteger(num1);
BigInteger b = new BigInteger(num2);
BigInteger sum = a.add(b);
return sum.toString();
```

Note: BigInteger will be covered later.

Properties of Strings.

① ⇒ Sub Strings
① As the name suggests, a sub-string refers to a string inside a string
② A sub-string is a continuous string made from another. substring(i, j)
③ We can find the substrings of a string by iterating over every index. The time complexity will be Quadratic.
④ If the len of the string is n, then the total sub-string will be n*(n+1)/2.

② ⇒ Sub sequences.
① Sub sequence is also a string made from another string but it is different from a substring.
② Unlike substring, sub-sequences may or may not have continuous characters. It might be possible that a subsequence does not contain any character at all (empty string).
③ Another name of a subsequence can be a subset.
④ There are 2 possibilities for every position in the string-
- To take that element in the subsequence (represented by 1)
- To not take that element in the subsequence (represented by 0).

⑤ One important property of the sub sequence is that it follows the actual order of characters the same as the original string. Ex. string = "abcdef", "abf" → ✓ subsequence | "cbfc" → ✗ not subsequence.

StringBuffer in Java.

Ex
```
class Main{
    psvm{
        StringBuffer sb = new StringBuffer();
        sb.append("WeMakeDevs");
        String str = sb.toString();
        sout(str);
    }
}
```
o/p: WeMakeDevs

⇒ StringBuffer ⟶ mutable sequence of character unlike String which is immutable.

- Advantages over String
  1. mutable
  2. more efficient
  3. thread safe → मतलब, say दो threads हैं। Both are working on same data. say t1 t2, तो जब एक thread डेटा [illegible] अभी ~~t1~~ use कर रहा है तो दूसरी thread को kaam nhi karne देगा.
     This is cla thread safe.
     (Make it slower)

(StringBuilder is not thread safe)

⇒ Constructor

// constructor 1

jo yaha likha

~~StringBuffer sb = new~~ StringBuffer();

```
//constructor 2
StringBuffer sb2 = new StringBuffer("Kunal Kushwaha");
//constructor 3
StringBuffer sb3 = new StringBuffer(30);
sb.
sb.insert(2, "Rahul");
```

O/P: ~~He Rahul Makedevs~~ WeRahul Makedevs.

```
sb.replace(1,5, "kushwaha");
```

(eRa → Kushwaha)

O/P: WKushwahahul Makedevs

```
sb.delete(1,6);
```

O/P: Whahahul Makedevs.

```
sb.reverse();
```

O/P: sveDekaM luhahaW

```
sout(sb3.capacity());
```

O/P: 30 (b/c specified, agar nhi kiya toh default 16).

Q WAP to print RandomString

```
import java.util.Random;
class RandomString {
    static String generate(int size) {
        StringBuffer sb = new StringBuffer(size);
        Random random = new Random();
        for (int i=0; i<size; i++) {
            int randomChar = 97 + (int)(random.nextFloat() * 26);
            sb.append((char)randomChar);
        }
        return sb.toString();
    }
}
class Demo {
    psvm {
        String name = RandomString.generate(10);
        sout(name);
    }
}
```