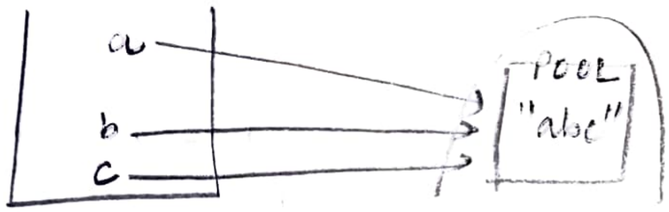
[10-01-22] * strings are stored in heap

STRINGS → sequence of characters → strings are immutable
→ In-built class of java.
→ Data-type :→ Non-primitive

string pool → separate memory structure inside heap memory
→ use-case of string pool : make our program more optimized if more than one object is ~~[illegible]~~ equal to same value, it will not create separate objects of same value. for eg.

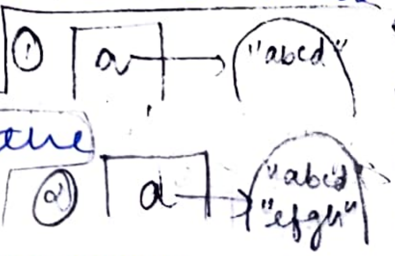


Why POOL seperated? All similar values of string should not be created again.

→ Here, if we make changes to obj 1 → a, so that change will not reflect to obj 2 → b inspite of having same value references. This is because of Immutability.

→ strings are ~~[illegible]~~ declared/defined in double quote " ".
→ syntax → String $\underset{\text{reference v.}}{n}$ = $\underbrace{\text{"abcd efgh"}}_{\text{object}}$;
→ every string value you assign to is not variable but object type of string

* Immutable → can't modify or change the object.
→ strings are immutable for security purpose.



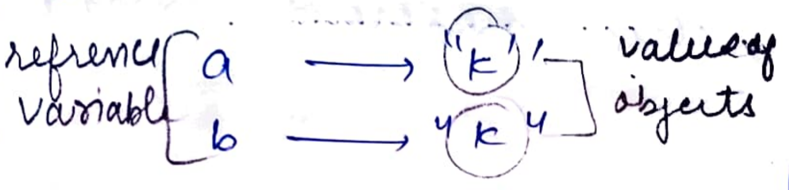
eg. ① String a = "abcd";
sout(a); → // abcd
② a = "efgh"
sout(a) → //efgh

Here we do not change the value rather we created new object "efgh" and assign it to ⓐ.

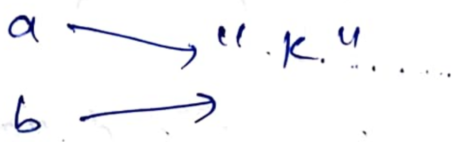
* Comparison of strings :

.charAt(index) → To check or identify character from string.

① (==) method
comparator
use of this comparator
eg -



There are 2 objects having same value, so
a == b → false

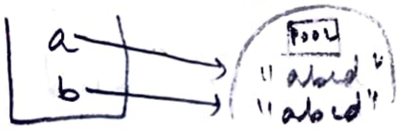


There is one value/object having two different reference variable, so
a == b → true.

comparator "==" checks for value of objects and reference variable, if the reference variable is pointing same object or not.

// How to create different objects of same value.

by using new keyword → creates new object
eg.
```
String a = new String ("abcd");
String b = new string ("abcd");
```
This creates these values of object outside the pool but in heap only.



→ comparators check the object is same or not but when we need to check only value, use .equals method or fn.

a.equals(b) → // true

* Pretty Printing :-

```
float a = 433.1234f;
~~socce~~
system.out.printf("%.2f" + a);
```

printf ↓ print formatted

"%.2f" → placeholder for value; format specifier

→ % → placeholder

→ for different data types, there different formate specifies.

* METHODS STRING PROVIDE

→ toCharArray :- it converts string to char array.

→ toLowerCase :- convert to lower case but by creating a new object as we know abt immutability.

→ indexOf :- give index of particular char in string

→ strip :- white space will be removed from string.

few of them

FORMAT SPECIFIER

- %c → char
- %d → decimal
- %e → exponential
- %f → float
- %i → integer
- %o → octal
- %s → String
- %u → unsigned decimal
- %x → hexadecimal
- %t → date/time
- %n → new line

*operator Overloading :- (not in java but in cpp & python)

→ '+' is only operator, overloaded by java for String concatenation and to add string with data of other types.

:- Addition of characters

'+' operator converts value into no. then ~~[illegible]~~ add it into numeric / ASCII value of character.

:- Addition of String and integer

integer in converted to its wrapper class and then added to String. eg. "a" + 1 → a1

:- '+' operator in java is only defined for primitive data types and when any of these values is string.

* StringBuilder → represents mutable sequence of char. Since String creates immutable sequence, StringBuilder provides an alternative to String class, as it creates mutable.

→ separate class in-built

→ check from video.

→