Strings :- memory mapping

Similar memory mapping to 1-d array

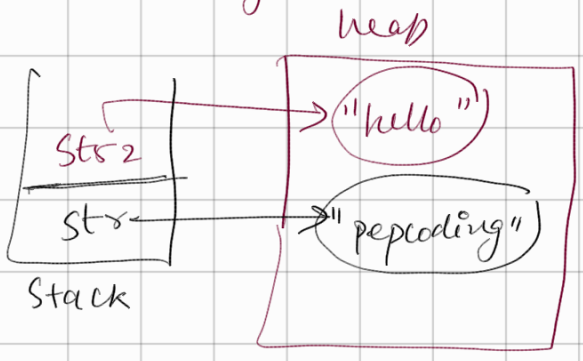
String str = "pepcoding"

String is not only a data structure but an object with its own methods. Eg: length()

str is in stack & stores reference of the heap base address of string memory block

② String str2 = new String("hello"); // str = "pepcoding"

String is made & stored in heap because new key word is used

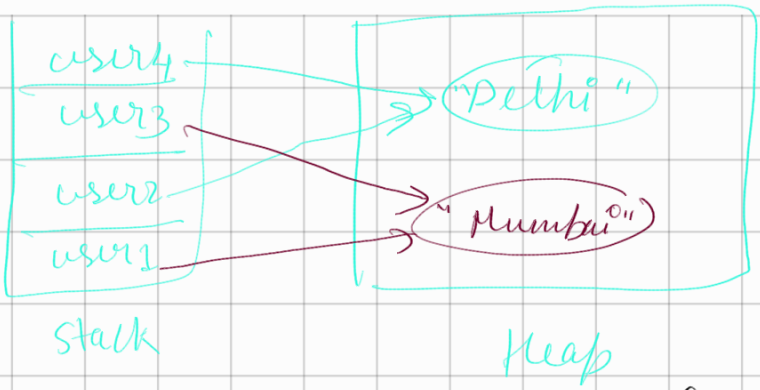


* Properties of strings. ① String Interning

- for any project, 50 - 100% data structures used are strings

Eg: city name : repeated city names will have to be stored making duplicate entries.

So instead, just create one object of city name & all the residents of that city can point to that memory address in heap.



① This is called string interning (Saves space)

```
String u1 = "Delhi";
"      u2 = "Delhi";
:
:
```

This area in heap, where array of characters are stored is called String Pool or String Intern Pool.
It is a special area in Heap.

Whenever a new string is declared, it first checks if the string already exists in the String Pool or String Intern Pool if it does, then only it's reference is stored. If not present new object is made

Implication of String Interning

① difference b/w == & equals()

If val == 10
checks for equality of LHS & RHS

CHECKS FOR ADDRESSES.

str1.equals(str2)
returns true if equal & false if not

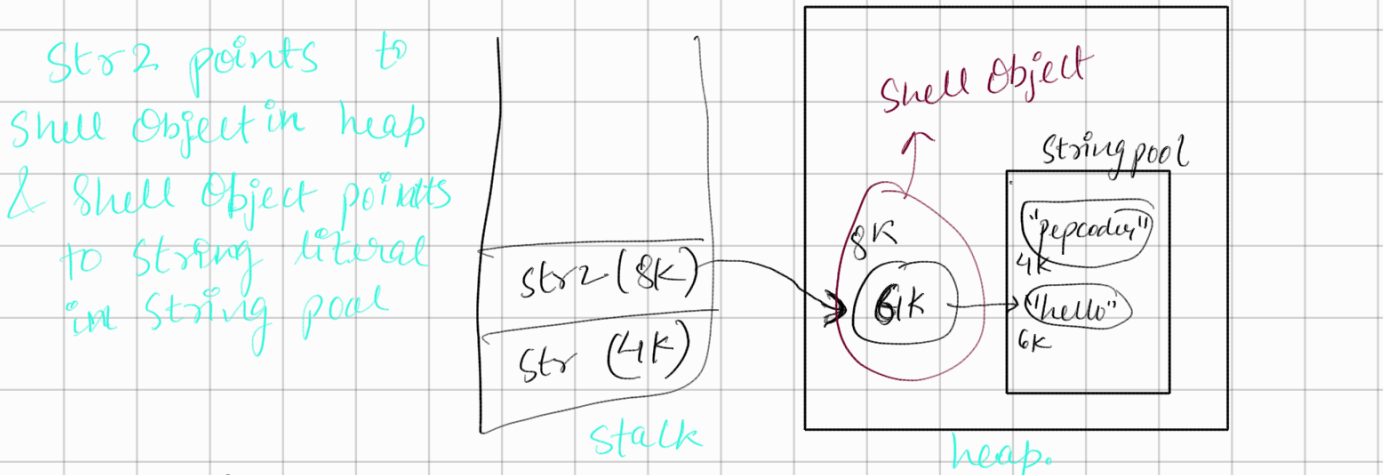
DOES NOT CARE ABOUT ADDRESSES

==	isequals()
Checks for the addresses	checks for pointed string
If both addresses are same then true	If final pointed string is same then true
or false	or false

(2) Immutability: If multiple users use the same literal and one of them want to change it, the object cannot be changed. So instead we have to change the users address.

To avoid object changes, no refurence can change or update the string object.

Whenever String is made in java, interning is always applied for: String str2 = new String ("hello");



Using the above code, we make an object in the heap but outside the string pool which points to the string pool is char array.

Shell Object Stores value of string literal address inside the string pool

However,

String str = "pepcoder";

Only checks for interning & points to string literal address directly.

There is no Shell Object involved.

Now if another string is declared as.

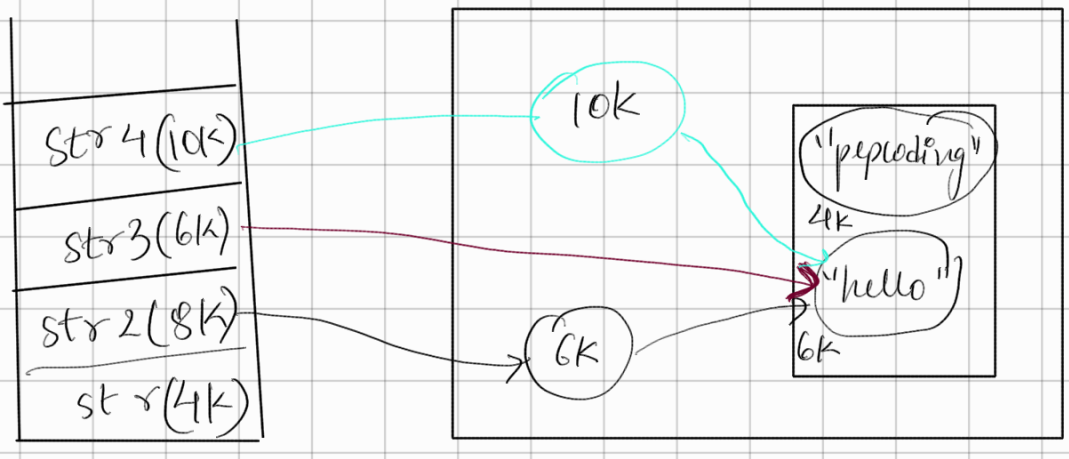
str3 = "hello";

directly points to 6K because interning

But

str4 = new String("hello");

makes a new shell object in heap but outside the string pool pointing to the string literal hello 6K



Stack

So if new key word is used, a Shell object will always be created but not new string literal. & after making Shell object Interning is checked.

Without it, no new shell object is ever made and directly Interning checked.

str5 = "hello".

① str2 == str3

guess
false ✓

② str3 == str5

true ✓

③ str2 == str4

false ✓

④ str2.equals(str3)

true ✓

⑤ str3.equals(str5)

true ✓

⑥ str2.equals(str4)

true ✓

⑦ str == str2 ⟶ false

⑧ str.equals(str2) ⟶ false

Important, ① == can give wrong answer for same string but different address values.

So always use

```
s1.isEquals(s2);
```

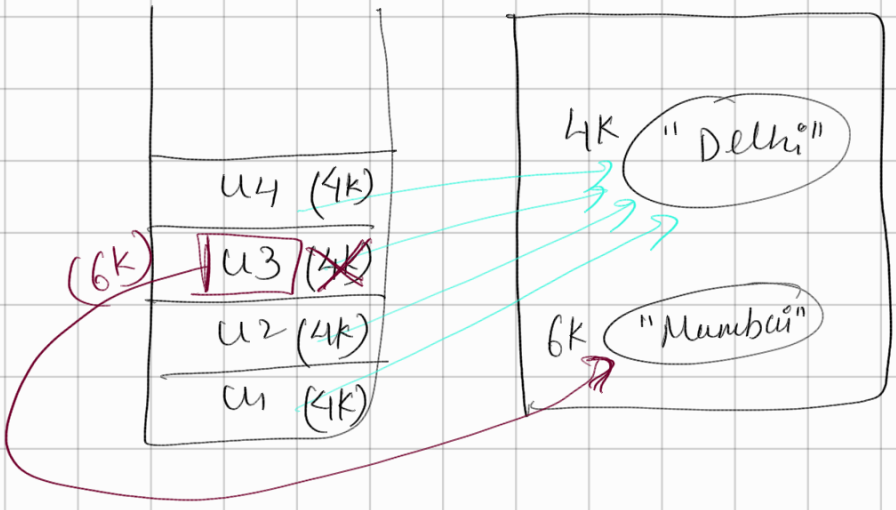
to compare content.

for string declared using "new String()" 2 objects are involved

① Shell Object &
② String literal

② "References are mutable, but instance/object is not mutable."

Updation will change reference and not the object



If u3 changes from delhi to mumbai, its reference changes from 4k to 6k

6k is stored in u3 instead of 4k stored previously

We cannot write S.charAt(i) = 'H' in java.

Until and unless there is at least one reference to an object, it stays fixed & cannot be modified / destroyed.

```
String s = "0";
for(int i=1; i<=1000; i++){
        s= s+i;
}
```

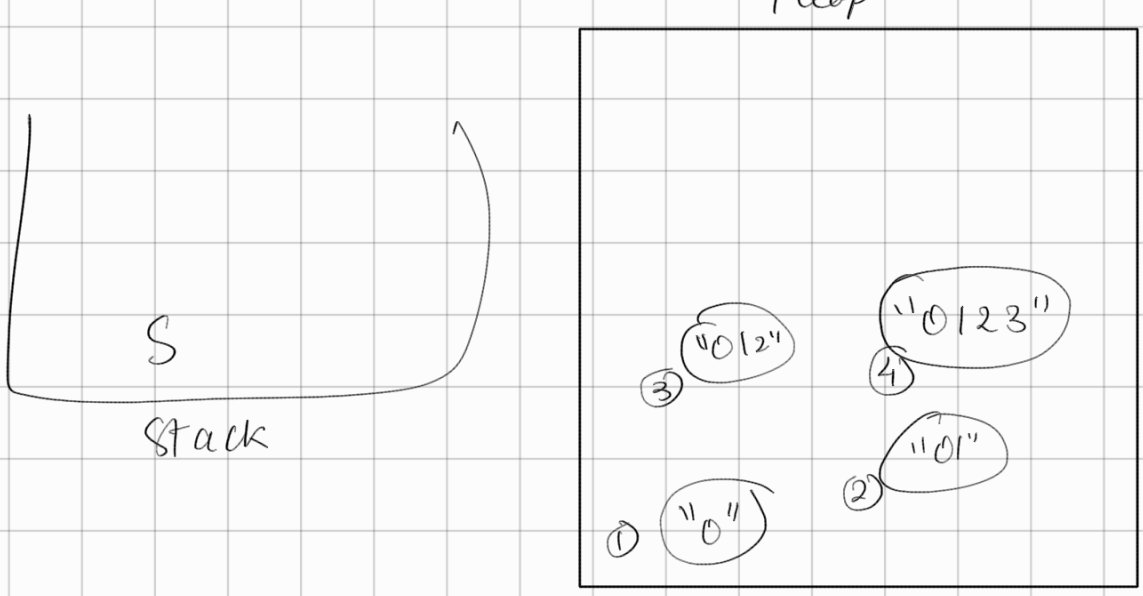
→ Concatenate

Output

"0123 . . . . . 1000".

But strings are immutable in java.

So, to concatenate each time a new object is made and s stores the reference of this new object



s points to ① for i = 1

. . . . . ② . . i = 2

. . . . . ③ for i = 3

So everytime, new string made & s is updated

In Java, appending is O(n) operation (reference not string literal)

$1 + 2 + 3 \cdots 1000 = \left(\frac{1000 \times 1001}{2}\right)$ $O(n^2)$

In C++, same thing runs in $O(n)$ using string & appending in $O(1)$.

Every iteration of above code, the previous string literal is deleted. So memory is not wasted

(11) String Compression.

wwww aaa d e xxxxxx

w4 a3 d e x6

① print wadex

② print w4 a3 de x6

① Approach 1,

```
for(int i=0; i < n-1; i++)
    if (S.charAt(i) != S.charAt(i+1))
        res = res + S.charAt(i+1);
return res
```

CODE:

```
int n = S.length();
String ans = "" + S.charAt(0);
for(int i=0; i < n-1; i++)
    if (S.charAt(i) != S.charAt(i+1))
        ans += S.charAt(i);

return ans;
```

(2) wwww | aaa | d | e | xxxxxx

Approach

Traverse String

if prevval != currval

add lastgrp freq if > 1

add currval in ans

else

freq ++;

CODE:

```
int n = S.length(), freq = 0;
String ans = "" + S.charAt(0);
for(int i=0; i<n ; i++) {
if(i>0 || s.charAt(i) != s.charAt(i-1)) {
        if( i>0 && freq >1)          // if freq>1 only then
            ans = ans + freq;        // add
        ans= ans + s.charAt(i);
        freq =0;                     // update for next group
    }
    freq ++;
}
if( freq >1)  ans += freq;  // for last freq
return ans;
```

③ String with difference of every 2 consecutive characters

pepCODinG

p -11 e 11 p -45 C 12 O -11 D 37 i 5 n -39 G

Homework