Strings

Def: Collection of characters enclosed with double quotes " " → String

↓ Any Object

Non-primitive data types

→ String str = new String("ABC"); } Object

→ String str = "ABC".

String constant Pool: It is a region present inside the Heap Area

"ABC" "ABC"

→ Only string objects gets created inside this region

→ No duplicate copies are allowed inside SCP

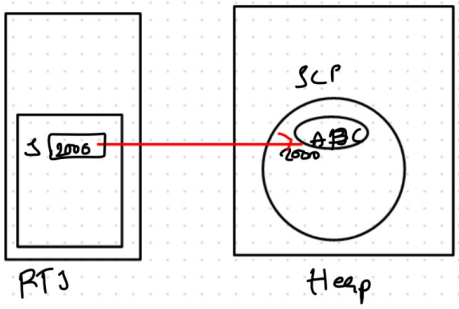
→ Even if no one is refering to this string object in SCP, the memory doesn't deallocate.

String s1 = "ABC"

s1 = null

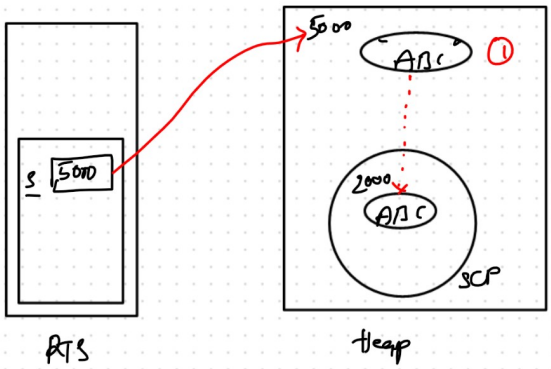
Case 1:

String s = "ABC"



Case 2:

String s = new String("ABC")



Note:

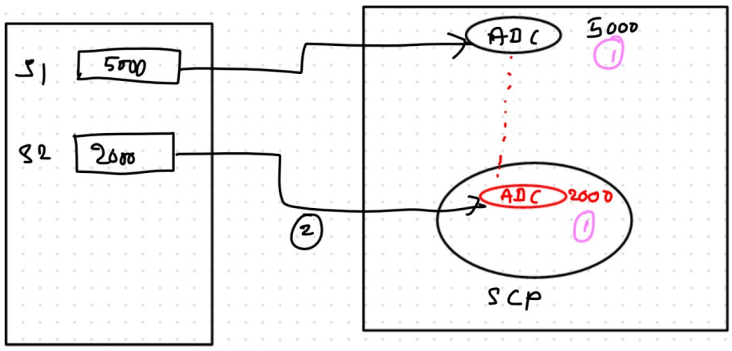
1) when ever there is "new keyword" associated while creating String object it creates a String object outside the Non Constant Pool, but it will be inside the Heap Area & the address will be returned to our local reference

2) copy of the same object will be created inside String Constant Pool only if that object doesn't exist

Case 3

→ String s1 = new String ("ABC")

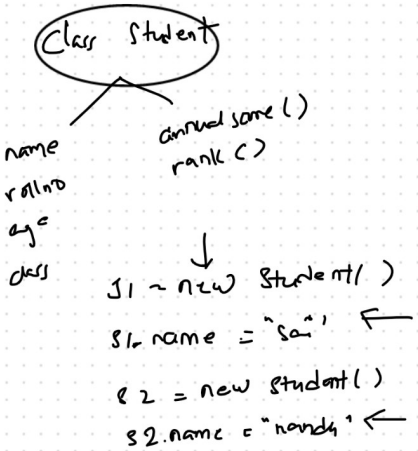
String s2 = "ABC"



Case 4:

String s1 = new String ("ABC")

String s2 = new String ("ABC")

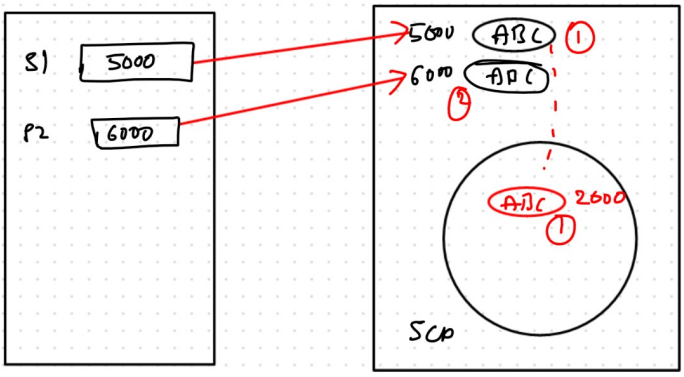


s1 = new Student()

s1.name = "Sai" ←

s2 = new Student()

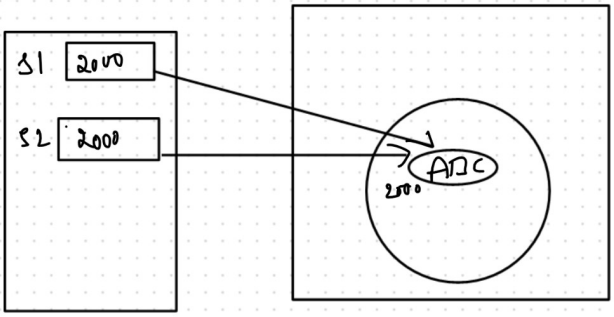
s2.name = "Nandu" ←



Case 5:

String s1 = "ABC"

String s2 = "ABC"



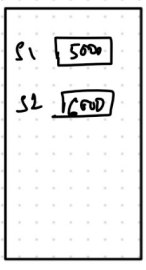
Comparisions

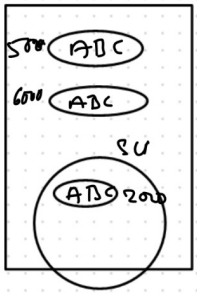
Case 6:

String s1 = new String ("ABC")

String s2 = new String ("ABC")

S.O.P(s1 == s2) // false





Note:

Compare the address

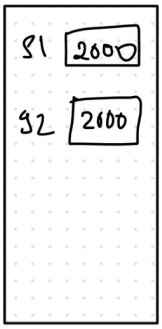
s1 == s2

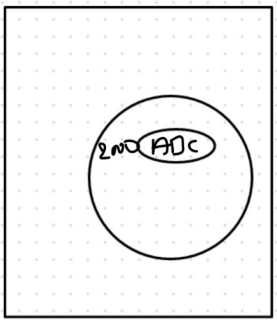
5000 = 6000

Case 1:

```
String s1 = "ABC";
String s2 = "ABC";
S.O.P(s1==s2) // True
```

2000 = 2000





Object always compared ==> the addresses

Notes:

(1) "==" Operator [String comparison]

It doesn't Compare the string literals rather it compares the address/reference in the case of { }

(2) equals method

In case if we want to compare the value, we can use ready made/inbuilt method equals

```
String s1 = new String("ABC");
String s2 = new String("ABC");
S.O.P(s1.equals(s2));
```

Case 2:

```
String str = new String("ABC");
S.O.Pln(str) // ABC → No address
```

Note:

(1) In String it prints the value while printing the refrence variable because toString() inside String Class is overriden

Object
↑
class UserDefined

method overridding

str.toString() → toString(){ X return address return value }

⊕ Operator loading [implicit]

integers 1+1 = 2 [addition will happen]

string "1"+"1" = "11" [concatination]

"abc"+"def" = "abcdef"

"abc"+"---"+"def" = "abc---def"

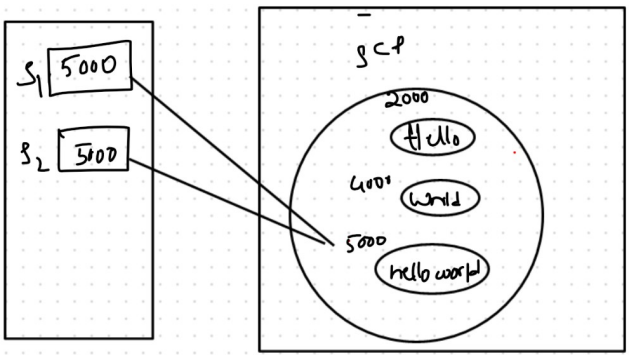
Operator Overloading

[1] + [2]

[1, 2]

Case 9

```
String s = "hello" (1) + "world" (2)
S.O.P(s)
```



Case 10:

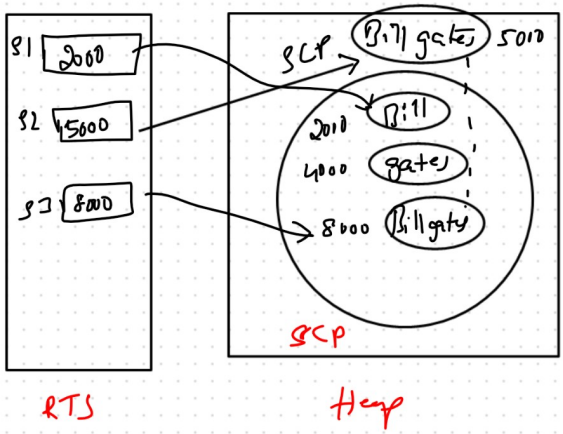
```
String s1 = "Hello" + "World"
String s2 = "Hello" + "World"
S.O.P(s1 == s2) // T
```

Note: During concatination (+) if no string variable is involved the resultant string object will be created inside SCP

Case 11:

→

```
String s1 = "Bill";
String s2 = (s1) + "gates"
String s3 = "Billgates"
S.O.P(s2 == s3) F
S.O.P(s1 == s3) F
Bill == Billgates
```



Notes

→ Because of concatination (+) with a variable s1 so object created outside SCP

s1 + "Gates"

→ During concatination of the string if any string variable is involved the resultant string object will be created outside the SCP and copy of it present inside the SCP

Case 12

```
String s = "Bill"
s = "gates"
S.O.P(s) // gates
```

y = "Bill"

Mutable Immutable