Strings

Continuation

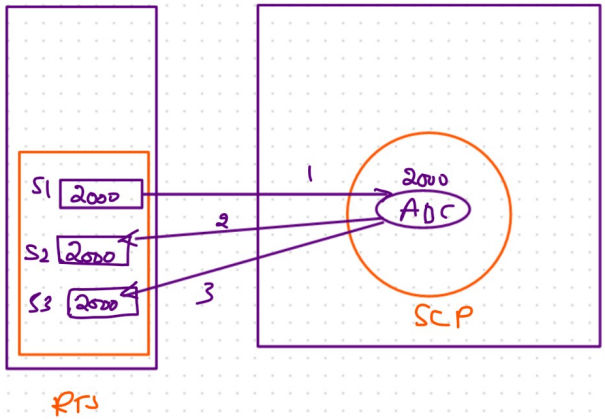
Case 5:

Memory mapping

String s1 = "ABC"

String s2 = "ABC"

String s3 = "ABC"



Case 6:

String s1 = new String("ABC");

String s2 = new String("ABC");

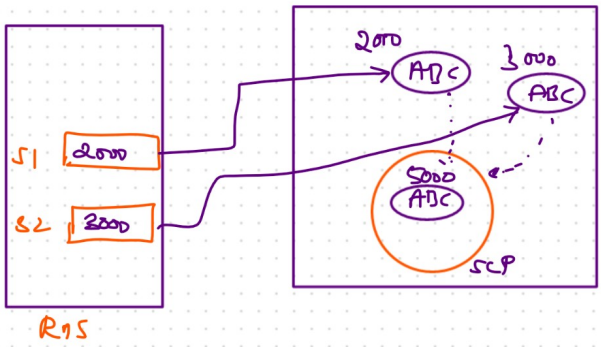
S.O.P (s1 == s2) // False

s1 = 2000

s2 = 3000

2000 == 3000

False



Case 7:

String s1 = "ABC"

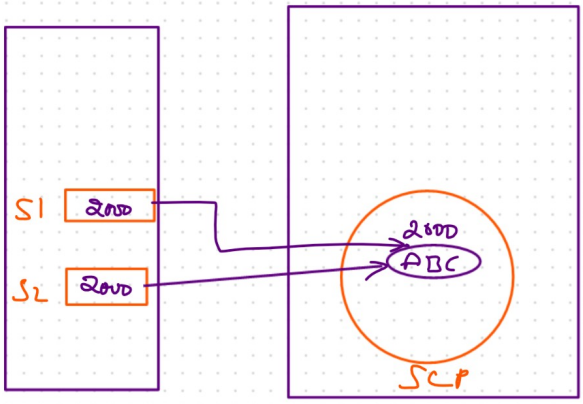
String s2 = "ABC"

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

s1 = 2000

s2 = 2000

(2000 == 2000) T



(1) `==' operator [String Comparison] a ==b → operator
op1 op2

String → object
= "ABC
= new String

It doesn't compare the string literals rather it compares reference/address in the case 6 & 7

(2) Equals method:

String s1 = "ABC"

String s2 = new String("ABC")

s1 == s2 false

→ s1.equals(s2) // true

In case if we want to compare the value we can use a ready made method called equals

Case 8:

String s = new String("ABC")

System.out.println(s) // ABC Not Address ✗

Note:

→ In case of strings if we try to print the reference variable it would print the value rather than printing fully qualified class @address

→ In string it prints the value while printing the reference variable because toString() in String Class is overridden

Operator Loading [Implicit]

+, <, >, =, <=

addition → 1 + 1 = 2 ← addition [Both are primitive type, then addition is done]

op1 op2

Concatination → 1 + "1" = "11" [it behave like Concatination]

i.e here '+' operator is overloaded implicitly

Concatination

1 + 1 = 2

1 + "1" = "11"

"1" + 1 = "11"

"1" + "1" = "11"

1 + 1 + 1 = 3

1 + 1 + "1" = 2 + "1" = "21"

["1" + 1] + 1 = "11" + 1 = "111"

1 + "1" + 1 = "111"

"1" + "1" + 1 = "111"

1 + "1" + (1 + 1) =

Operator precendence → 1 + "1" + 2

"11" + 2 = "112"

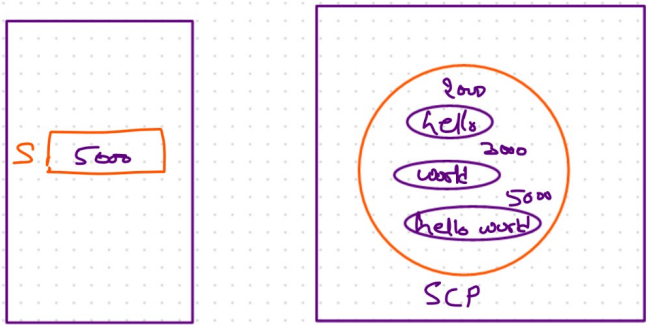
Case 9 :

String s = "hellu" + "world" literal

(1) concatination [hello + world]

(2) "hello world"

Memory mapping

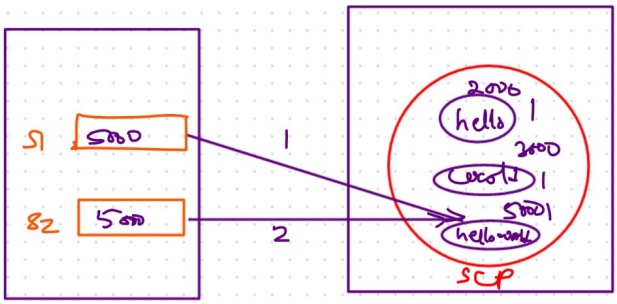


Case 10

```
String s1 = "hello" + "world"
String s2 = "hello" + "world"
S.O.P (s1 == s2) // true
```

Note:

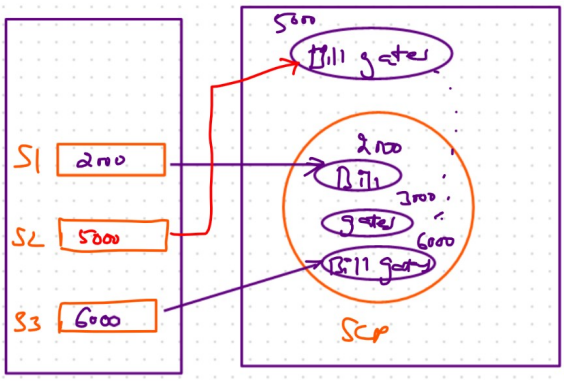
During Concatenation if no string variable is involved the resultant string will be created inside the SCP



Case 11

"Bill" + "gates"

```
String s1 = "Bill"
String s2 = [s1 + "gates"]
String s3 = "Bill gates"
```



Notes

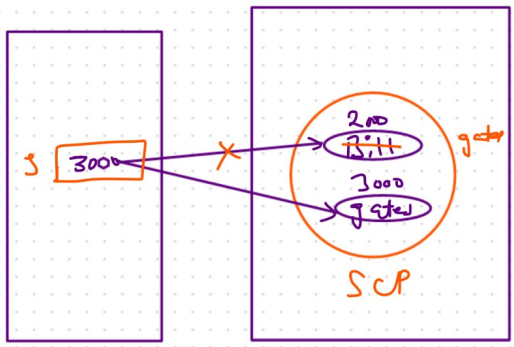
→ Because of Concatination with a variable s1, so object created outside the SCP

→ During Concatination the string if any string variable is involved the resultant string [output] will be object created outside the SCP & copy of it present inside the SCP & returns the address of the object which is outside the SCP

Case 1:

Mutable / Immutable

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



Case 3

```
String s = new String ("ABC");
s = new String ("ABC");
```