String , String Builder and String Buffer

String is immutable. The contents of string can not changed later.

```
case 1 :  String s = new String ("Durga");  → // 1
          s.concat ("Software");            // 2
          S.O.P (s);                        // 3
```

#1 : one obj create in SCP. s → Durga

DurgaSoftware

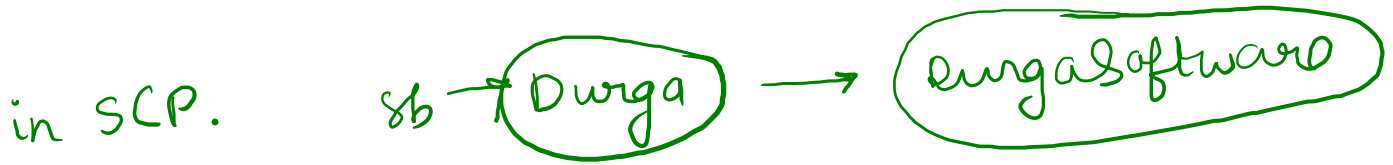
#2 : becoz of immutability new obj create in SCP.

#3 : output : Durga ? [s → pointing to Durga obj , not and obj]

non - changeable behaviour is called immutability.

Case 1 : (b) StringBuffer is mutable. [contents can be changed later]

```
StringBuffer sb = new StringBuffer("Durga");   //1
sb.append("Software");                         //2
S.O.P(sb);                                     //3
```



#1 : create a one obj in SCP.

#2 : append will change contents of sb obj (becoz of mutability)

#3 : output : DurgaSoftware

case 2 :

equals() Vs == operator

equals() → checked contents

== → checked references

equals() except object class

StringBuffer sb1 => new StringBuffer("Durga");

StringBuffer sb2 => new StringBuffer("Durga");

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

S.O.P(s2.equals(s1)); // False ?

equals() → checked address

object class

String class equals() checked contents (overridden)

StringBuffer equals() not overridden by default use object class equal()

heap

sb1 → Durga

sb2 → Durga

SCP

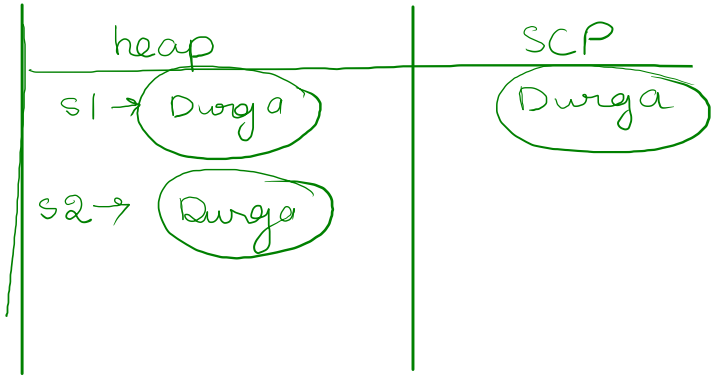
Durga

as equals() check reference only sb1 & sb2 are having different memory address.

Both StringBuilder & StringBuffer does not override the equals method.

Case 2:

```
String s1 = new String ("Durga"); //1
String s2 = new String ("Durga"); //2
S.O.P( s1 == s2); //3        False
S.O.P( s1.equals(s2)); //4    True
```



{ SCP can be extend in size due to present in heap }

String s = new String ("Durg");

J 1.6V → SCP (permgen) [method area]

J 1.7V SCP move to heap for memory efficiency.

#1 & #2 s1 & s2 obj create in heap memory. And one 'Durga' literal obj create in SCP.

#3) equal operator is checked reference only.

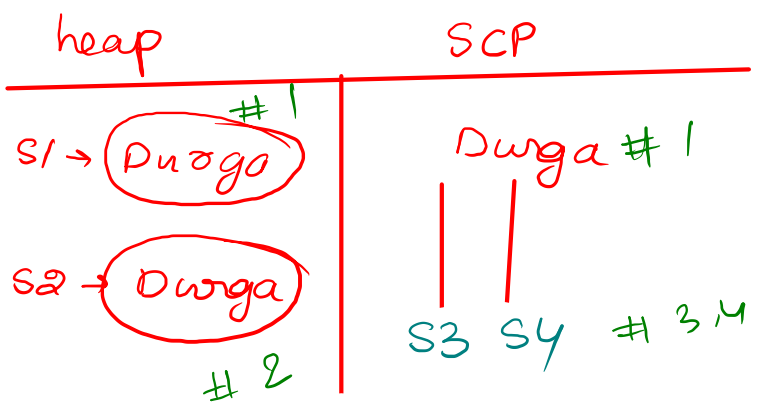
#4) equals method check contents of string :

String class Vs StringBuffer → ⊕ equals() method difference

⊕ immutable vs mutable

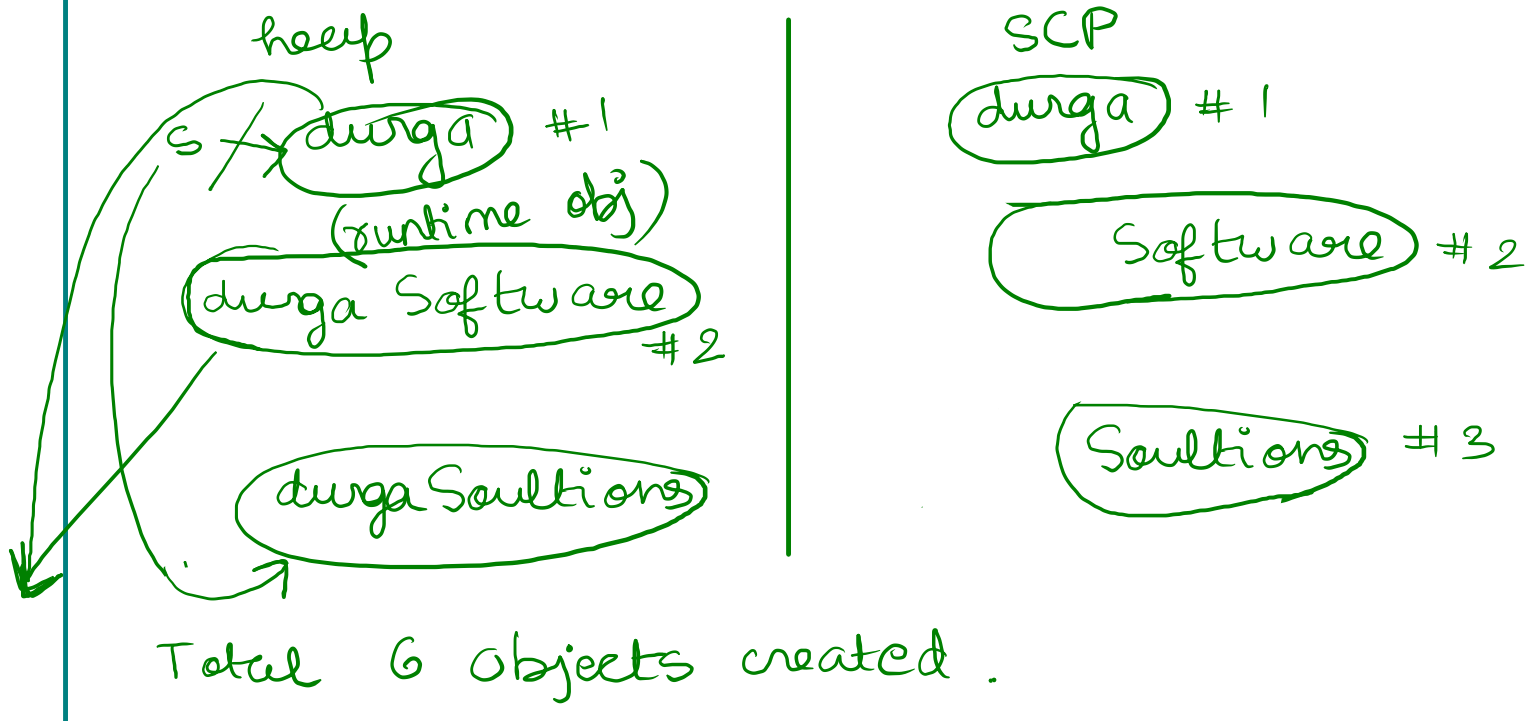
case 3:

```
String s1 = new String ("Durga");   // 1
String s2 = new String("Durga");    // 2
String s3 = "Durga"                 // 3
String s4 = "Durga";                // 4
```



case 4:

```
String s = new String("durga");   // 1
s.concate ("Software");           // 2
s = s.concate (" Solutions");     // 3
```



eligible for garbage collector

Case 4 :

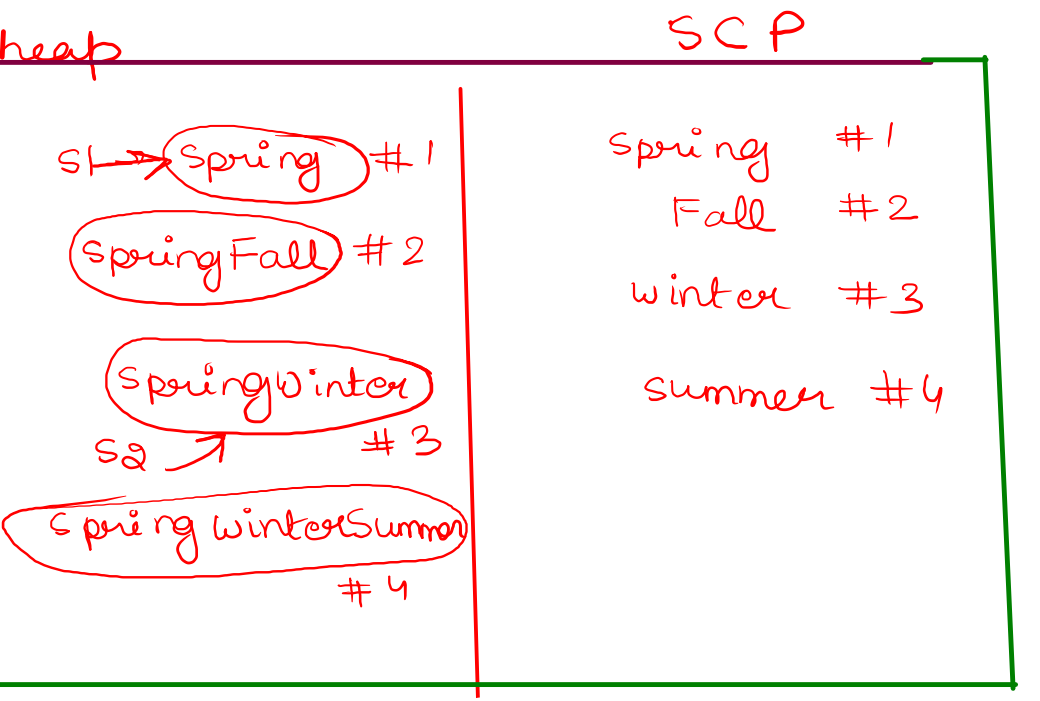
```
String s1 = new String ("spring");   // 1
s1.concat ("Fall");                  // 2
String s2 = s1.concate ("winter");   // 3
s2.concat ("summer");                // 4
                                     // 5
s.o.p (s1);
s.o.p (s2);                          // 6
```

runtime obj

output :

#5 Spring

#6 SpringWinter



Final Vs Immutability

```
final StringBuffer sb = new StringBuffer ("durga");
      sb.append ("softwares");
      S.O.P(sb);                    // durgaSoftware
sb = sb.append ("Solutions");       // CE : cannot assign value to final variable
```

by adding final , still value of sb is changed.

final is for variable where reassignment is not allow.

Immutability for object where contents can not be changed.

case 5:

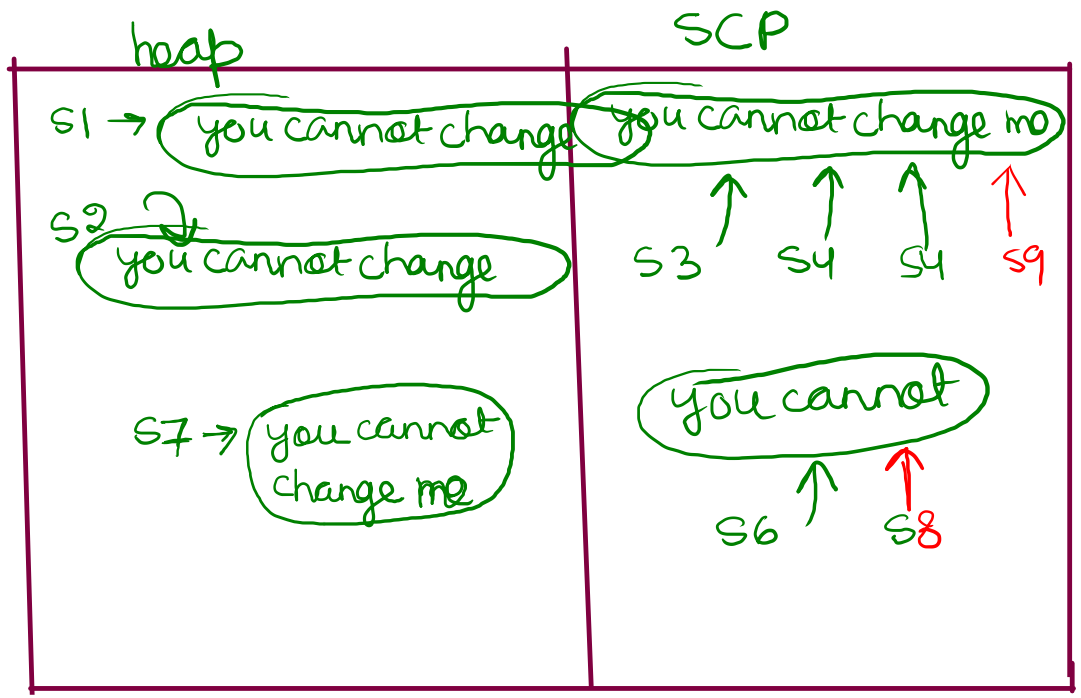
```
String s1 = new String ("you cannot change me");   // 1
String s2 = new String ("you cannot change me");   // 2
s.o.p (s1 == s2);                                  // 3   False
String s3 = "you cannot change me");               // 4
s.o.p (s1 == s3);                                  // 5   False

String s4 = "you cannot change me");               // 6
s.o.p (s3 == s4);                                  // 7   True
String s5 = "you cannot" + "change me";            // 8
s.o.p (s5 == s4);                                  // 9   True
String s6 = "you cannot";                          // 10
String s7 = s6 + "change me";                      // 11
s.o.p (s4 == s7);                                  // 12  False
```

#8 at compile time resolve as literals (JVM)

At least one is variable, then it execute at runtime. #11

```
final String s8 = "you cannot ";                   // 13
String s9 = s8 + "change me";                      // 14
s.o.p (s9 == s4);                                  //  True
```



final is a constant, line 14 execute at compiletime. line 14 = line 55

Advantages of SCP

+ Reuseability : same obj can be referenced by multiple objects so memory utilisation is improved
+ performance is increased.
+ But if many objects point to same literal in SCP, then any obj change the literal content then automatically, reflected to other objects also. So Java People introduce the 'immutability' concept.

---

1. Why SCP concept is available only for string not for string Buffer?

String is a like regular customer (mostly used object) that why Jvm provide special memory management for String.

---

2. Why string are immutable where String buffer are not?

String uses for multiple object where reuseability concept are required that why make string immutable but in case of string buffer there is such type of requirements.

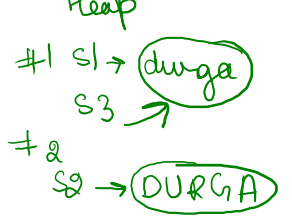
3. In addition to string is any other things in java follow immutable?

Yes, All wrapper class upto certain range follow immutability.

case6:

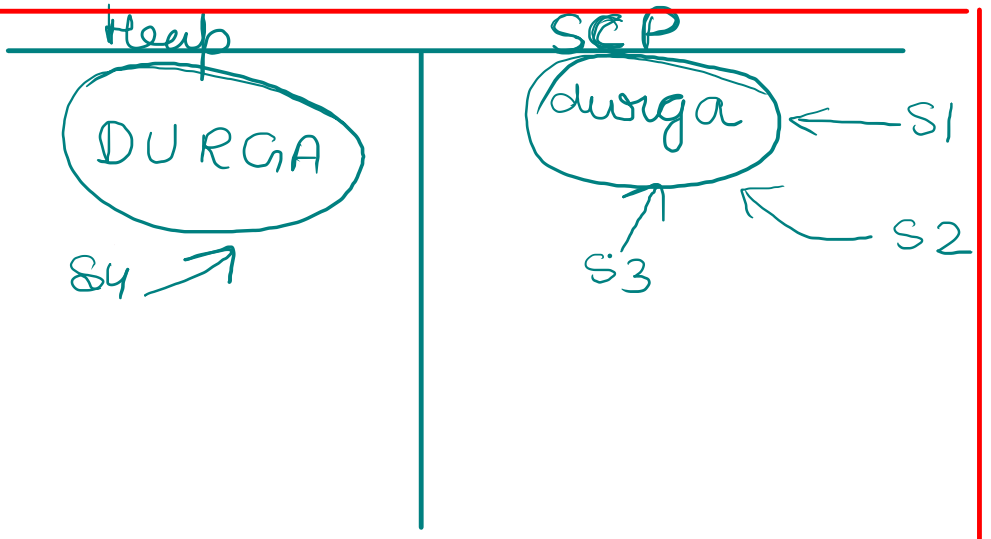
```
1 String s1 = new String ("durga");
2 String s2 = s1.toUppercase();
3 String s3 = s1.toLowercase();
4 S.O.P (s1 == s2); // False
5 S.O.P (s1 == s3); // True
```

→ runtime object (no need to create in SCP)
→ This is same as s1 [as no contents change]



toString() & tolowercase()
→ does not change any String content.

```
String s1 = "durga";
String s2 = s1.toString();
String s3 = s1.toLowercase();
String s4 = s1.toUppercase();
S.O.P (s1 == s2); True
S.O.P (s2 == s3); True
S.O.P (s1 == s4); False
```



Q4. Given

```
public class Test
{
    public static void main(String[] args)
    {
        String s="Hello World";
        s.trim();
        int i1=s.indexOf(" ");
        System.out.println(i1);
    }
}
```

What is the result?

A. An exception is thrown at runtime
B. -1
C. 5
D. 0

Answer: C

Explanation:
String objects are immutable. Once we creates string object we cannot perform any changes in the existing object.
If we are trying to perform any changes with those changes a new object will be created.
trim() method will remove spaces present at beginning and end of string but not middle blank spaces.

StringBuffer Capacity if use a string

String st = new String("durga");

length = 5 capacity = 5

StringBuffer sb = new StringBuffer("durga");

length = 5 capacity = 16 → if

we add 17th item, new cap = (16+1)*2 = 34

= (34+1)*2 = 70 so on.

StringBuffer sb = new StringBuffer(String s);

sb = new StringBuffer(st);

length = 5 cap ⇒ (st.length + 16)

cap ⇒ 5+16 = 21

StringBuffer	StringBuilder
Every Method Present In Stringbuffer Is Synchronized.	No Method Present In Stringbuilder Is Synchronized.
At A Time Only One Thread Is Allow To Operate On Stringbuffer Object And Hence It Is Thread Safe.	At A Time Multiple Thread Are Allowed To Operate On Stringbuilder Object And Hence It Is Not Thread Safe.
Threads Are Required To Wait To Operate On Stringbuffer Object And Hence Relatively Performance Is Slow.	Threads Are Not Required To Wait To Operate On Stringbuilder Object And Hence Relatively Performance Is High.
Introduced In 1.0 Version.	Introduced In 1.5 Version.

Qb. Given the code fragment:

```
public class Test
{
    public static void main(String[] args)
    {
        StringBuilder sb=new StringBuilder(5);
        String s="";
        if(sb.equals(s))
        {
            System.out.println("Match 1");
        }
        else if(sb.toString().equals(s.toString()))
        {
            System.out.println("Match 2");
        }
        else
        {
            System.out.println("No Match");
        }
    }
}
```

What is the result?

A. Match 1
B. Match 2
C. No Match
D. NullPointerException is thrown at runtime

Answer: B

Explanation:
In StringBuilder equals() method is not overridden and hence object class equals() method will be executed. If arguments are different type then equals() method return false. Hence sb.equals(s) returns false.
But String class equals() method meant for content comparison.
Hence sb.toString().equals(s.toString()) returns true.

Q) which statement will empty the contents of a StringBuilder variable named sb?

A) sb.deleteAll()

B) sb.delete(0, sb.size())

C) sb.delete(0, sb.length())

D) sb.removeAll()

Q) Given the

```
class myString
{
    String msg;
    myString(String msg)
    {
        this.msg = msg;
    }
}
public class Test
{
    p s v main(String[] args)
    {
        Sopln("Hello" + new StringBuilder("Java SE 8"));
        Sopln("Hello" + new myString("Java SE 8"));
    }
}
```

what is the result?

A) Hello Java SE 8
Hello myString@<hashcode>

B) He... SE 8
Java SE 8

C) Hello java.lang.StringBuilder@<hashcode>
Hello myString@<hashcode>

D) Compilation Fails

Q) Given the following code

```
class myString
{
    String msg;
    myString(String msg)
    {
        this.msg = msg;
    }
}
public class Test
{
    p s v main(String[] args)
    {
        Sopln("Hello" + new StringBuilder("Java SE 8"));
        Sopln("Hello" + new myString("Java SE 8"));
    }
}
```

S
SB
SB
wrapper classes
Collection class

Hello myStri-g@<hashcode>

what is the result?

A) Hello Java SE 8
Hello myString@<hashcode>

B) Hello Java SE 8
Hello Java SE 8

C) Hello java.lang.StringBuilder@<hash
Hello myString@<hashcode>

D) Compilation Fails

Q) which statement will empty the contents of a StringBuilder variable named sb?

A) sb.deleteAll()

B) sb.delete(0, sb.size())

C) sb.delete(0, sb.length())

D) sb.removeAll()

sb.delete(begin, end)
begin index to
end-1 index

StringBuilder sb = new S

sb.delete(0, 5);

from 0 t

all ch

Q10. You are developing a banking module. You have developed a class named MaskTest that has a mask method.
Given the code fragment:

```
class MaskTest
{
    public static String mask(String creditCard)
    {
        String x="XXXX-XXXX-XXXX-";
        //Line-1
    }
    public static void main(String[] args)
    {
        System.out.println(mask("1234-5678-9101-5979"));
    }
}
```

You must ensure that mask method returns a String that hides all digits of the credit card number except last four digits(and the hyphens that seperate each group of 4 digits)

Which two code fragments should you use at line 1, independently to achieve the requirement?

A.
```
StringBuilder sb=new StringBuilder(creditCard);
sb.substring(15,19);
return x+sb;
```

B.
```
return x+creditCard.substring(15,19);
```

C.
```
StringBuilder sb=new StringBuilder(x);
sb.append(creditCard,15,19);
return sb.toString();
```

D.
```
StringBuilder sb=new StringBuilder(creditCard);
StringBuilder s=sb.insert(0,x);
return s.toString();
```