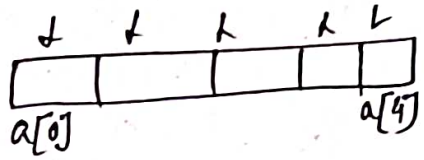
Types of Array in JAVA

1. Single - Dimensional Array
2. Multi - Dimensional Array (Array of Array)

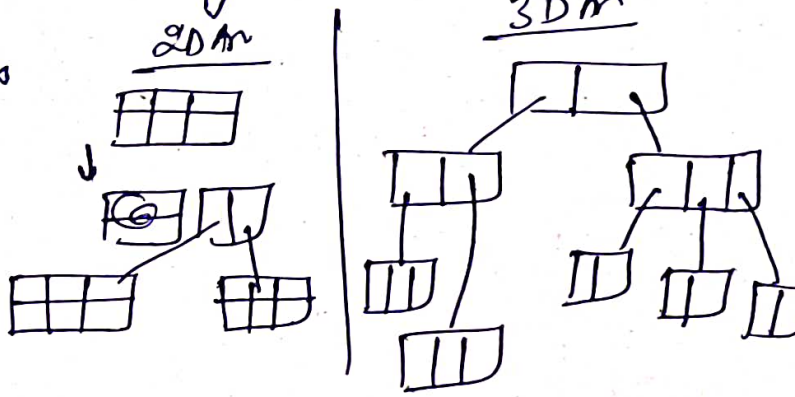
1. Single - Dimensional Array



An array that has only one subscript or one dimension is known as a SDA. It is just a list of the same data type variables.

→ One dimensional (1D) array can be either one row & multiple columns or multiple rows and one column.

```
int marks[] = {50, 60, 70, 80, 90};
or
int marks[];    // declares an array
marks = new int[5];   // allocating memory
// initialising array using index
marks[0] = 50;
marks[1] = 60;
marks[2] = 70;
marks[3] = 80;
marks[4] = 90;
```




```
String steArray[] = {"Python", "Java", "C++", "C", "PHP"};
for (String i : steArray)
{
  S.O.P (i + " ");
}
S.O.P ("\n length of array: " + steArray.length);
}
```

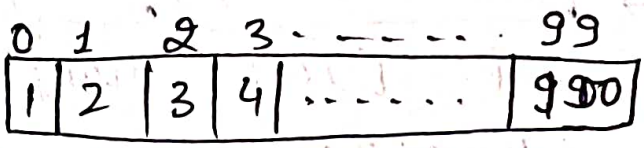
Array

② Stored in Heap due to object

```
int emp_id [ ] = new int [100];
```

↓

① Object



→ 0 to (n-1) index

Adv : multiple values stored
Array is fast, directly stored object in the array

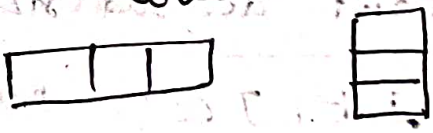
dis : size fixed [100];
not inc/dec in run-time
→ size 100 then values comes 50, then its wastage of memory
→ only stord similar data types. int = 100 ✓ int = a ✗

```
String emp_name [ ] = new String [100];
```

→ `company c[] = new company [100];`

Types of Arrays

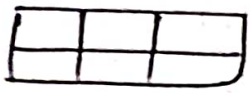
① Single Dimensional → Single row / single column
↳ 1D



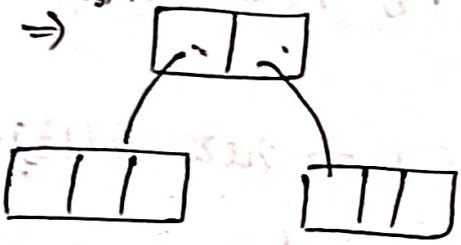
② Multi Dimensional → multiple rows / multiple columns (Arrays of Arrays)
↳ 2D
↳ 3D

Multi Dimensional

2D Array



Position



```
int[][] a;
```

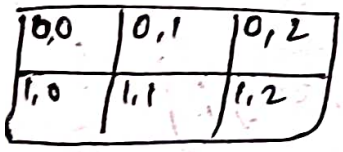
⇒ `int [][] a;`

`int[] [] a;`

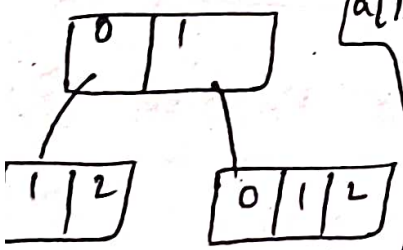
`int a[][];`

`int[] a[];`

```
int [] [] a;
a = new int[2][3];
```



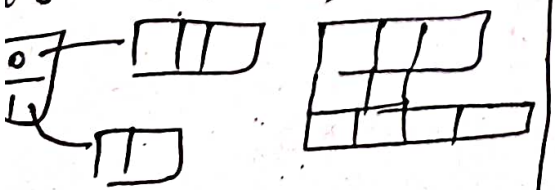

```
a[0][0]=10;
a[0][1]=20;
a[1][1]=40;
```



MATRIX (3,3)

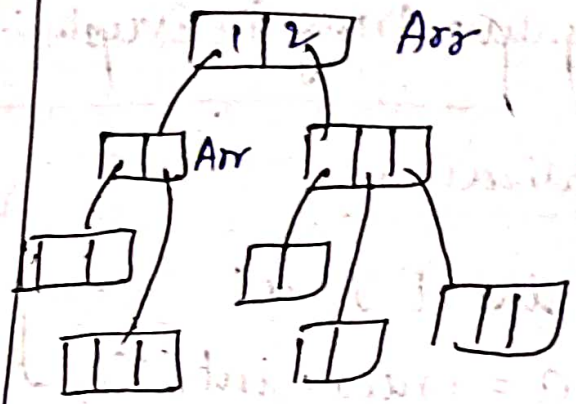
JAGGED ARRAY

```
[0] = new int[3];
[1] = new int[2];
```



```
[][] a={{10,20,30},{40,50,60}};
```

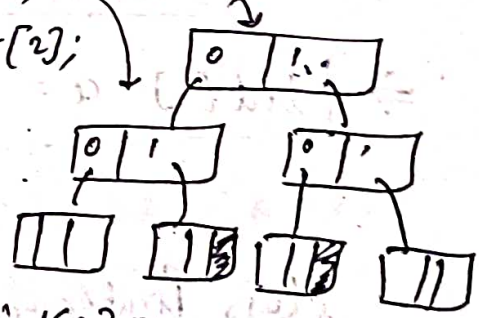
3D Array




```
int[][][] a;
```

```
int[] [][] a;
int[] [] a[];
int [][] a[];
int [] a [][];
```

```
int[][] [] a;
a = new int[2][][];
a[0] = new int[2][];
a[0][0]=new int[3];
a[0][1] = new int[2];
```




```
a[1] = new int[2][];
a[1][0] = new int[2];
a[1][1] = new int[3];
```

State the data in rows & column

② 2(Two) Dimensional Array

3D.

n columns

a[0][0]	a[0][1]	a[0][2]	a[0][3]	... a[0][n-1]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][n-1]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][n-1]

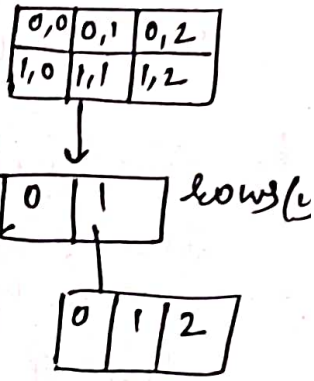
m rows

a mxn
row col

Declare - int[][] a;
Init - a = new int[2][3];

Syntax

type[][] arrayname;
or
type [][] arrayname;

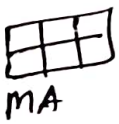


eg: int[][] z;
or int x[][];

→ int[][] Array = new int[2][5] declr

int arr[][] = {{1,2,3,4,5},{6,7,8,9,0}};

row=i j=col



multiple 2D Array

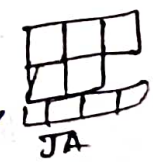
③ 3D Array:

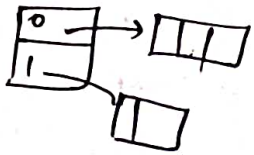
int[][][] x;
int arr[][][]
~~int matrix[][][~~

Anonymous Array in Java

int my3DArray[3][4][2] =
{
{{10,11},{12,13},{14,15},{16,17}},
{{20,21},{22,23},{24,25},{26,27}},
{{28,29},{30,31},{32,33},{34,35}};
};

MATRIX] ARRAYS
no of col same JAGGED
a = new int[2][];
a[0] = new int[3];
a[1] = new int[4];





Constructors

1. Constructor is a block (similar to method) having same name as that of class name.
2. Constructor does not have any return type, not even 'void'.
3. Only 4 modifiers are applicable for constructors i.e. public, protected, default & private (static, final, synchronized etc cannot be used with constructors).
4. Constructor executes automatically when we create an object.

Use of Constructor : Constructor is used to initialize an object (not to create object)

Types of Constructors

Three types of Constructors:

1. Default Constructor or no-arg constructor (Created by compiler)
2. No-Arg constructor (Made by programmer)
3. Parameterized constructor (create by programmer)

→ this() & super() Keyword

→

```
class Test
{
    public void Test()
    {
    }
}
```

2 ways to call constructor:

① `Test t = new Test();`

② `new Test();` (no need to create reference i.e. t)

```
class Test
{
  public Test()
  {
  }
  p s v m (String[] args)
  {
    Test t = new Test();
    t.Test(); X (no need to write)
  }
```

automatically

3 ways to initialize an object ~~to intro~~

① By using reg. variable

② By method

③ By using constructor

Without constructor

```
class Employee
{
  String name; // Instance var
  int emp_id;

  p s v m (String[] args)
  {
    Employee e1 = new Employee();
    Employee e2 = new Employee();
    :
    Employee e100 = new Employee();
  }
}
```

① name=null, empid=0 ← e1

② → emp_id = 0 (same) then this is not good programming

③ e1.name = "deepak"; e1.empid = 101;

name = deepak, empid = 101 ← e1

Heap

name=null, empid=0 ← e2

name = deepak, emp_id = 101 ← e2

String name = "deepak"; X

int empid = 101;

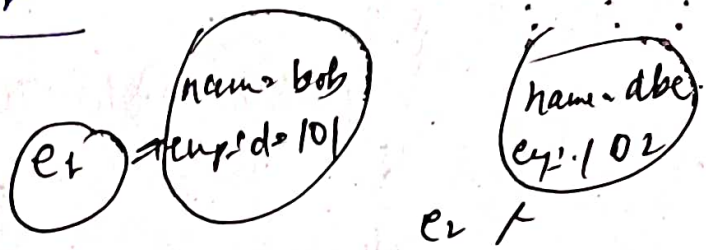
2000

e2.name = "abc"; // X

e2.empid = "102";

1000 of employee (extra line)

By using constructor




```
class Employee
{
    String name;
    int emp_id;
    Employee (String name, int emp_id)
    {
        this.name = name;
        this.emp_id = emp_id;
    }
    p s v m (String[] args)
    {
        Employee e1 = new Employee ("Bob", 101);
        Employee e2 = new Employee ("abc", 102);
    }
}
```

use → unique value to initialize
→ to initialize an object.

Types of constructor

① Default constructor (no-arg constructor)

compiler → ~~JVM~~

```
class Test
{
    Test ()        // created by compiler
    {
        super();   // default constructor
    }
    p s v m (String [] arg)
    {
        Test t = new Test();
    }
}
```

② No-arg constructor (user defined) Programmer defined

```
class Test
{
    Test ()        // created by user/programmer defined constructor
    {
    }
    p s v m (String [] arg)
    {
        Test t = new Test();
    }
}
```

③ Parametrized constructor

```
class Test
{
    Test (String name)
    {
    }
    p s v m (String arg[])
    {
        Test t = ~~new~~ Test ();
        Test t = new Test ("abc");
    }
}
```

① no return type constructor - because

~~void~~ Test (), ~~void~~ Test (String ne) → main works to initialize an object

when values is initialize then no need to return value;

② In default constructor compiler cannot be judge the return type which is given by user. already

Q: Diff b/w this & Super keyword & this () & super()

String Handling

90% object is used in Project (not int)

① String is non-Primitive Data-Type

② String is the sequence of characters (Array of characters)

eg:

```
char[] c = {'d','e','f','g','h','i'};
String s = new String(c);
```

↑ represents

→ JAVA is implement the interface i.e. CharSequence (Interface).

③ String is a class

```
class String
{
  == Method defined
     more
}
```

↓ Proper syntax

```
public final class String extends Object
{
  ==
}
```

Parent (class)

next

```
public final class String extends Object implements
        CharSequence, Serializable, Comparable
{
  ==
}
```

Parent class of the all class in Java

↑ inherit

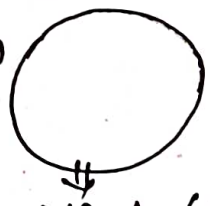
3 interface implements

Serializable interface is used when we need to store a copy of the object and send them to another process which runs on the same system or over the network.

Comparable interface is used to order the objects of the class. It has only one method i.e. compareTo().

④ String s = new String(); (Object create)

String is an Object
IMMUTABLE



object (Immutable)

→ We can create String class object by:
String s = new String();
It will create an IMMUTABLE object.

String s = "abcd";

Q: Difference b/w String s = "abcd"; & String s = new String();

⑤ → To create String, there are three classes:-

① String ② StringBuffer ③ StringBuilder

~~⑤ T~~ StringBuilder s = new StringBuilder

(Actual to stored string objects) String Constant Pool (or String Literal Pool)

→ String Constant Pool (or String Literal Pool) is an area in heap memory where Java stores String literal values.

SCP was present in Method Area until 1.6 version, but after 1.6 version it was shifted to Heap Area.

Memory Areas in JVM

PERMAEN SCP

1. Method Area — 1.6 version (SCP)
2. Heap Area — 1.7 version (SCP) SCP is Present in Heap Area.
3. Stack Area
4. PC Register
5. Native Method Area

String CP is special Memory Area stored in.

String Constant Pool (HEAP AREA)

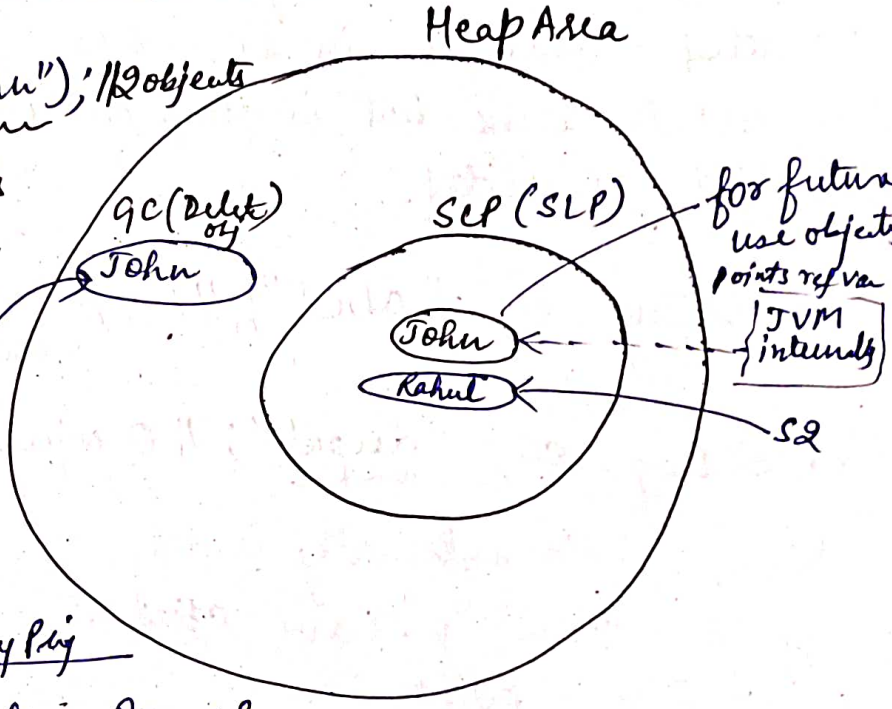
① `String s1 = new String("John");` // 2 objects

Literals

When we create literals in String is create one more SCP

② `String s2 = "Rahul";` // 1 object

JVM internally points ref. var.



Diff b/w String s1 ... & String s2
More object is created then Heavy Obj

Q: In SCP is Garbage Collection occurs?

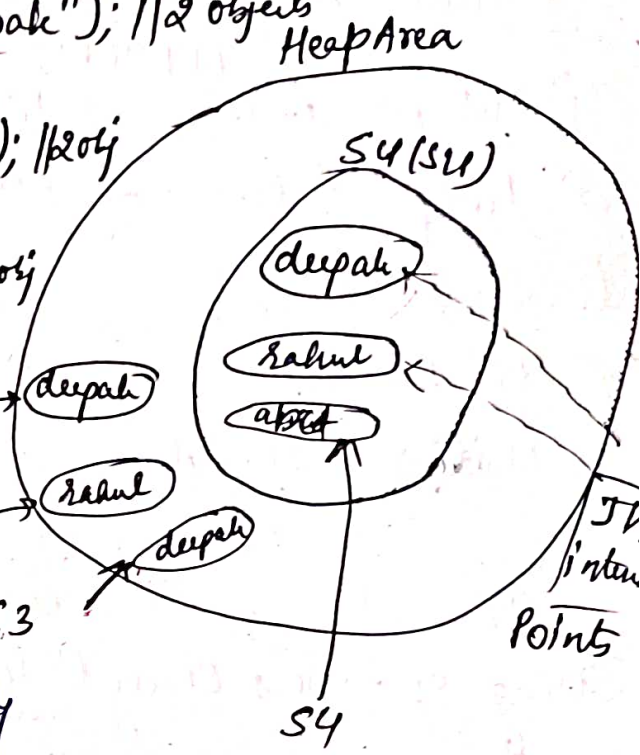
→ The String Objects present in SCP are not applicable for Garbage Collection because a reference variable internally is maintained by JVM.

→ String Constant Pool is not applicable for Garbage Collection as JVM internally creates reference variables for each String literal objects.

String s1 = new String ("deepak"); //2 objects

String s2 = new String ("rahul"); //2obj

String s3 = new String ("deepak"); //1obj



→ When we have same literals but different objects then in SCP no another object is created. like as in s1 and s3 literal is same but in Heap Area only 1 object is created

→ String s4 = "abcd"; // 1 object in this case only 1 object is created in SCP.

→ String s5 = "deepak"; // 0 object

Now, JVM internally is not point now, we have object s5 then it point its.

→ String s6 = "deepak"; // 0 object is created

In this case s6 is also point object

New Keyword used then object is created in Heap Area & SCP (SLP) when it not then only one SCP.

① String Introduction?

② String Constant Pool & String Object Creation in Java

③ What is String Immutability Concept? and why String Objects are immutable?

Abstraction

↳ 1. By abstract class (abstract methods)

↳ 2. By Interface

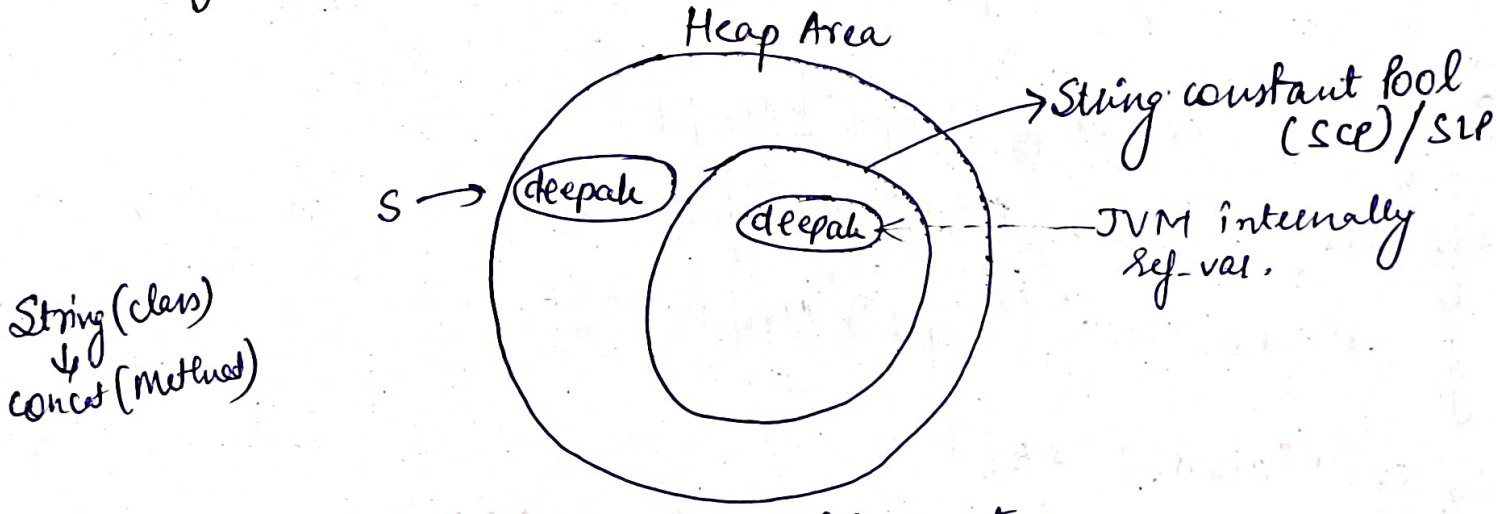
```
String s1 = new String ("deepak"); // 2 object
String s2 = new String ("rahul"); // 2 object
String s3 = new String ("deepak"); // 1 object
String s4 = "abcd"; // 1 object
String s5 = "deepak"; // 0 object
String s6 = "deepak"; // 0 object
```

String Immutable

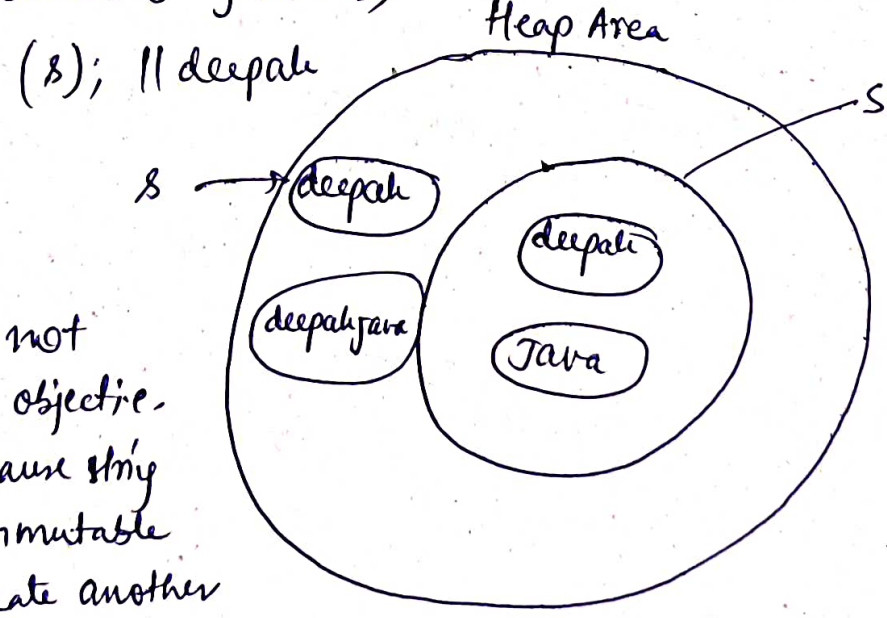
Immutable means "Unchangeable".

→ Immutability concept is used for "String Objects" i.e. String objects are immutable. It means once String Objects is created; its data or state can't be changed but a new String object is created.

```
String s = new String("deepak"); // 2 object is created
```

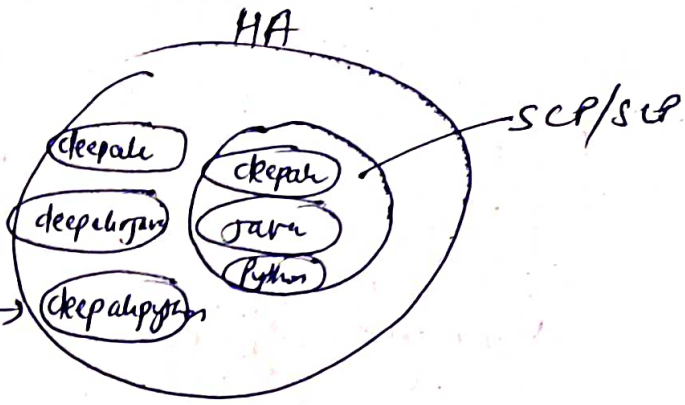



```
s.concat("Java");   // (+) operator
S.O.P.(s); // deepak
```



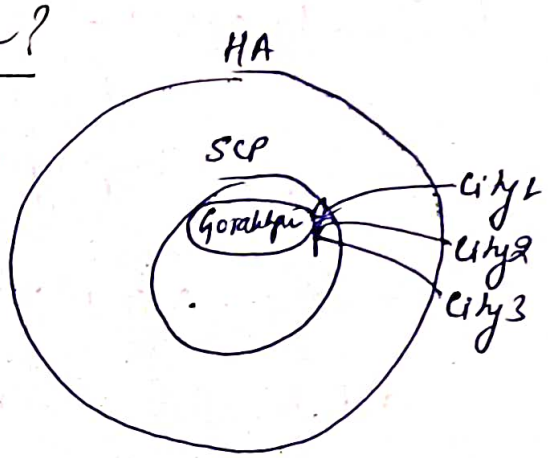
i.e. it will not change the objective. deepak because string object is immutable and its create another object and perform concat (append) with

deepakjava




```
s = s.concat(" Python");
S.O.P(s); // deepak Python
```

Why String Objects are Immutable?



P1 `String city1 = "Gorakhpur";`

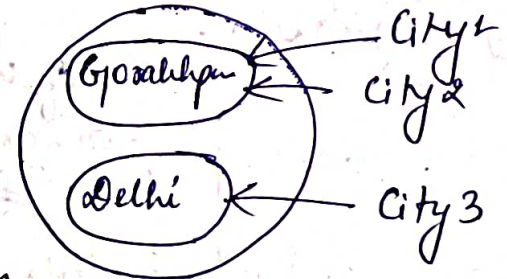
P2 `String city2 = "Gorakhpur";` (check)

P3 `String city3 = "Gorakhpur";`

P1000 If we have 1000 Person then only one Object (Gorakhpur) is if only 1

`String city3 = "Gorakhpur";` (Delhi)

If we change the value then it reflect to another object i.e. City1, City2 & City3 is already Gorakhpur & city3 is now change to Delhi.



Ans: Strings are Immutable in Java because String objects are cached in String pool. Since Cached String literals are shared between multiple persons there is always a risk, where one person's action would affect all another persons

for.ex: If one persons changes its city from "Gorakhpur" to "Delhi", all other persons will also get affected.

Real world ex: gmail → city fill
facebook → city fill
company work exp - "

String city4 = "delhi";

details filling → ref. only 1 object

→ String is old & special in java.

Q) Why String class is "final" & Difference between final and immutability (object)

class variable ~~and~~ Diff b/w

Final is the keyword which is used with class, method and variables.

```
class Abc
{
}

class Test extends Abc
{
}
```

Final class
X ↑ extends
class

→ If we want one class is not extends another then we used final ~~class~~ keyword.
So, then we can't extends
final class cannot be extend

```
final class Abc
{
}
class Test extends Abc
```