String.

*) String is non primitive datatype. because
*) it references a memory location where data is stored in the heap memory.
*) its size is not fixed.
*) String is the sequence of characters.

char c[] = {'k', 'i', 'i', 't'} // interface charSequence

↳ String s = new String(c);

is same as s = "kiit";

*) String is a class.

syntax:- public final class String extends Object implements charSequence, Serializable, Comparable.

already an Interface is created in java.

{
// methods
}

*) We can create an object of a String class.

String s = new String();

*) This object is immutable object.

*)

s → Immutable object.

*) To create String there are three main classes:
→ String
→ StringBuffer
→ StringBuilder.

String constant Pool, or literal pool

*) String constant pool is an area in heap memory where java stores String literal values.

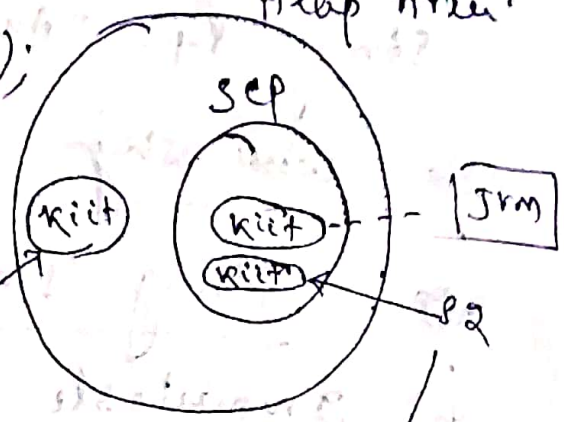
Memory Areas - 5 types

1) Method Area
2) Heap Area
3) Stack Area
4) PC Register
5) Native Method Area.

e.g.

```
class test
{
  int a=10;          (instance variable) → Heap Area
  static int b=20;   (static var) → Method Area
  void show()        → Stack Area
  { int c=30;        (local var) → Stack Area
  }
p.s.v.m()            → Stack Area
  { test t= new test();   (object) → Heap Area
    t.show();             → Stack Area
  }
}
```

1) String s1 = new String ("kiit");



literal

When we are creating

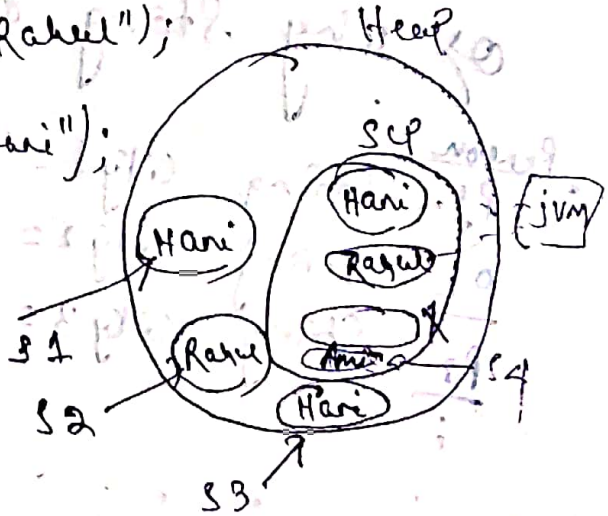
* a object using 'new' keyword it will create an object in heap area & when we are passing a literal it will create an object in String Constant Pool. So two objects are created. JVM internally create the Reference for this in constant pool.

2) String s2 = "kiit";

This type of declaration will create only one object in SCP. because new keyword is not used. This type of declaration is preferable.

* *) Objects in SCP are not applicable for Garbage collection.

Eg: String s1 = new String ("Hari");
String s2 = new String ("Rahul");
String s3 = new String ("Hari");



If same literal is used (Hari & Hari) then new object in SCP is not created because already one is created inside SCP.

String s4 = "Amit";

Q) ~~why String is final~~ ?

String Immutable

*) Immutable means we cannot change. i.e. once String object is created its data or state cannot be changed but a new String object is created.

e.g. String s = new String ("java");

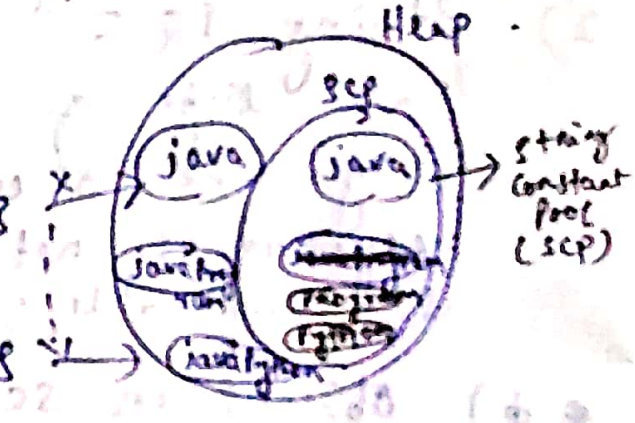
→ s.concat ("program");

s.o.p(s); → o/p: java

→ s = s.concat ("Python");

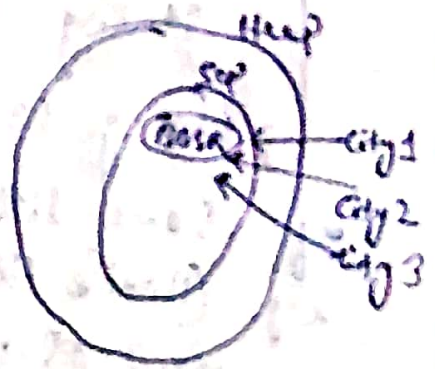
Now s will point to JavaPython but not java.

s.o.p(s) → javaPython.



Q) Why String objects are immutable ?

Person
P1 - String city1 = "BBSR"
P2 String city2 = "BBSR"
P3 String city3 = "BBSR"



Flow

If we change P3 is shifted to Delhi then new object is created for Delhi.

but inside SCP "OMER" will not change.

Q) Why String is final ?

- String class is final due to its characteristics of immutability, security, synchronization and caching. final is associated with class & variables and methods; immutability is associated with object.

Methods

1) length() :- It provides counts the no. of characters in a String

   e.g. String s = "Hello";
   S.o.p (s.length()); → 5

2) isEmpty() :- Returns true if the given String is empty.

   e.g. :- S.o.p(s.isEmpty()); → false
   == true);

3) trim() :- It eliminates only leading & trailing spaces.

   e.g. → String name = s.trim( );
   S.o.p(name);

   e.g. → String name = " abc ";
   String s = name.trim();
   S.o.p(s); → abc

   If (name.trim().length() == 0)
   {
   }

String class constructor.

class String
{
1) public String () → No Arg
{
}
2) public String (String s) {-} → string literal const.
3) public String (StringBuffer sb) {-} → string buffer const.
4) public String (StringBuilder sb) {-} → string builder const.
5) public String (char ch[]) {-} → string char array const.
6) public String (byte b[]) {-} → string byte Array const.
}

~~String Extraction (SubString)~~.

concat

class test
{
P.S.V.M (..)
{
String s1 = "Deepak";
String s2 = "java";
S.O.P(s1+s2); → Deepakjava
S.O.P(s1+10); → Deepak10
S.O.P(s1+10+20); → Deepak1020
S.O.P(10+20+s1); → 30Deepak
S.O.P(s1+20/2); → Deepak2
S.O.P(s1+10-5); → X
S.O.P(s1.concat(s2)); → deepakjava

join → join is the static method of String class.

Syntax:- join (charSequence delimeter, charSequence.... elements);

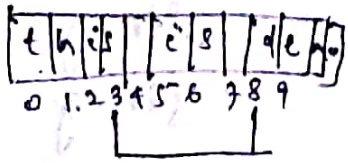
e.g:- S.O.P (String.join (",", s1, s2));

→ Deepak,java

SubString:

String s = "this is demo";

→ S.O.P (s.subSequence (3, 9));



Start index, End Index.

→ S.O.P (s.subString (3)); → s is d

→ S.O.P (s.subString (3)); → s is demo

Begining

→ S.O.P (s.subString (3, 11)); s is dem

Start Index, End Index

Replacing:

→ S.O.P (s.replace ("is", "was");

OLd char, new char → thwas was demo

→ S.O.P (s.replacefirst ("is", "was");

→ thwas is demo

Searching:

~~→ S.O.P (s.indexOf('e'));~~ String s = "deepak";

S.O.P (s.indexOf('e'); → 1

S.O.P (s.indexOf('ep'); → 2

S.o.p (s.lastIndexOf('e')); ⟶ 2

→ S.o.p (s.charAt(3)); ⟶ p

→ S.o.p (s.contains("ep")); ⟶ True

→ S.o.p (s.startsWith("d")); ⟶ True

→ S.o.p (s.endsWith("a")); ⟶ false

Comparing two String:

```
String s1 = "deepak";
String s2 = "Amit";
S.o.p (s1.equals(s2)); ⟶ false
```

```
s1 = "deepak";
s2 = "Deepak";
S.o.p (s1.equalIgnoreCase(s2)); ⟶ true.
S.o.p (s1.equals("")); ⟶ false
```

Searching:

```
String s = "deepak";
S.o.p (s.toUpperCase()); DEEPAK
S.o.p (s.toLowerCase()); deepak
int a = 10;
String s1 = String.valueOf(a);
```

→ a is converted to String.