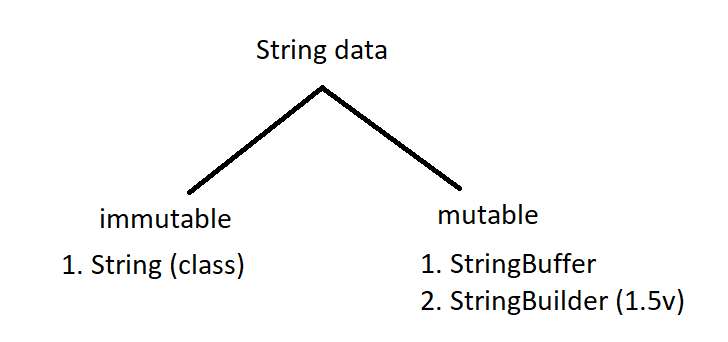
String

String is basically a inbuilt class present in java. lang. String package for which object can be created (user defined collection of characters enclosed in double quotes).

Eg: String\_Object\_Eg1



Since String is a class it may consists of instance variables and methods .

String :

class String{

// instance variables

// methods

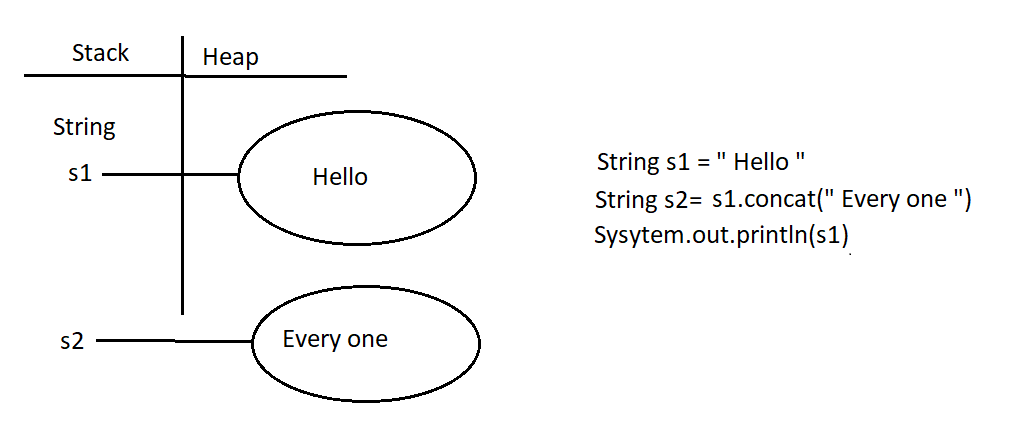
}

To access the string methods we should create an object and call them .

The object is the user defined string , with that reference variable of object we can call String methods

Eg: String\_Method\_Concat

Since string is immutable you cannot add anything to it once it is declared , but if we use concat method jvm creates a new object to concat . if that object is not collected by user , garbage collector will take that object .



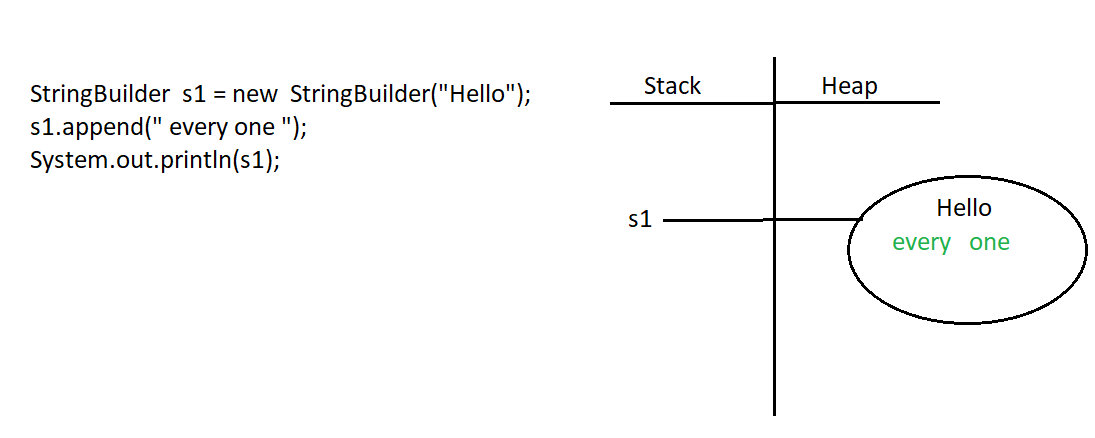
If concat method is used in the print statement there is no need of creating reference variable to collect that concat object.

If not used in the print statement or created reference variable for it , garbage collector will clear that.

StringBuilder

String builder is mutable , when it is declared an object is created . if we want to add something to that object we can use append() method of StringBuilder class . unlike String class jvm will not create another object to add new one . instead it adds to old object.

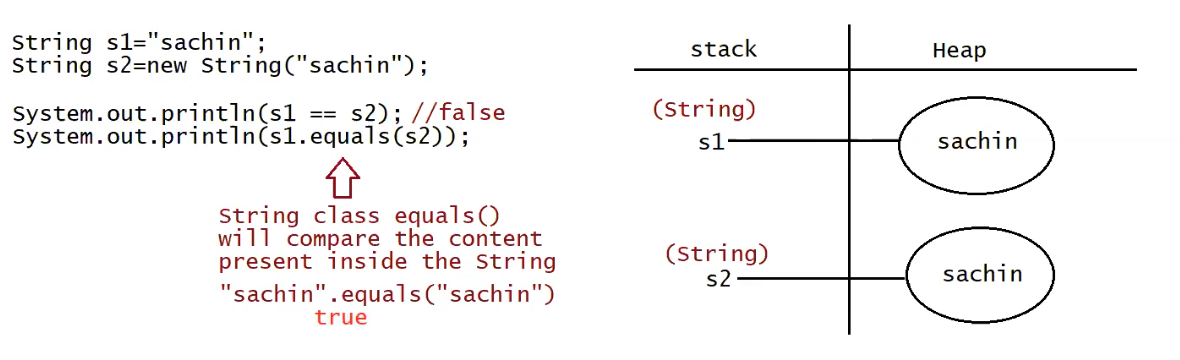
Eg: String\_Builder\_Eg3



String class equals method

Eg: String\_Equals\_Method\_Eg4

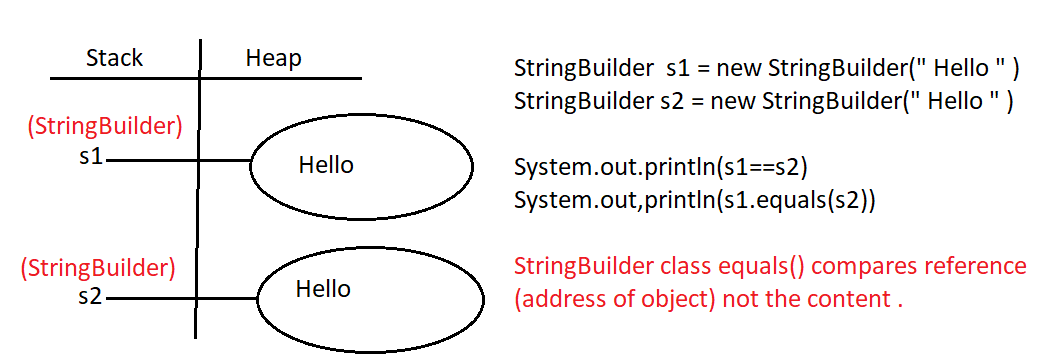
// go through the program



String \_Builder\_Equals \_Method

equals() method in String builder class compares the referece objects not the content in the objects.

Eg: String\_Builder\_Equals\_Method\_Eg5

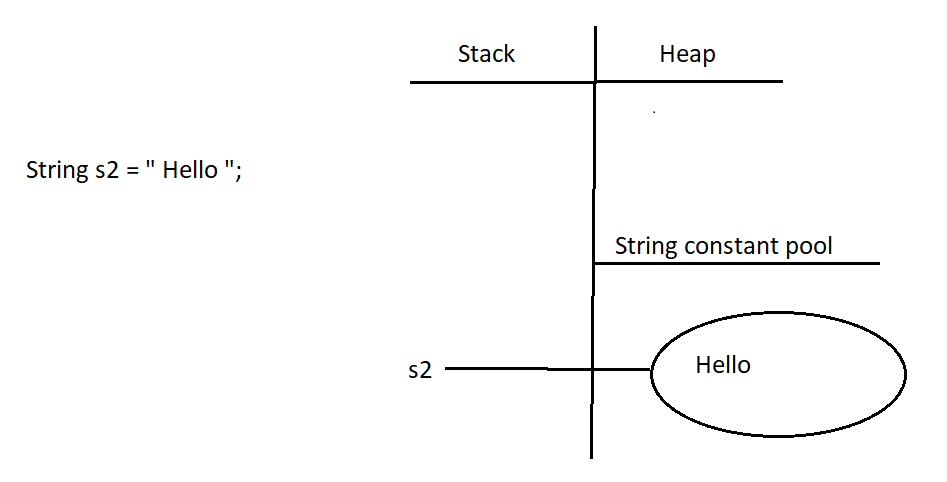


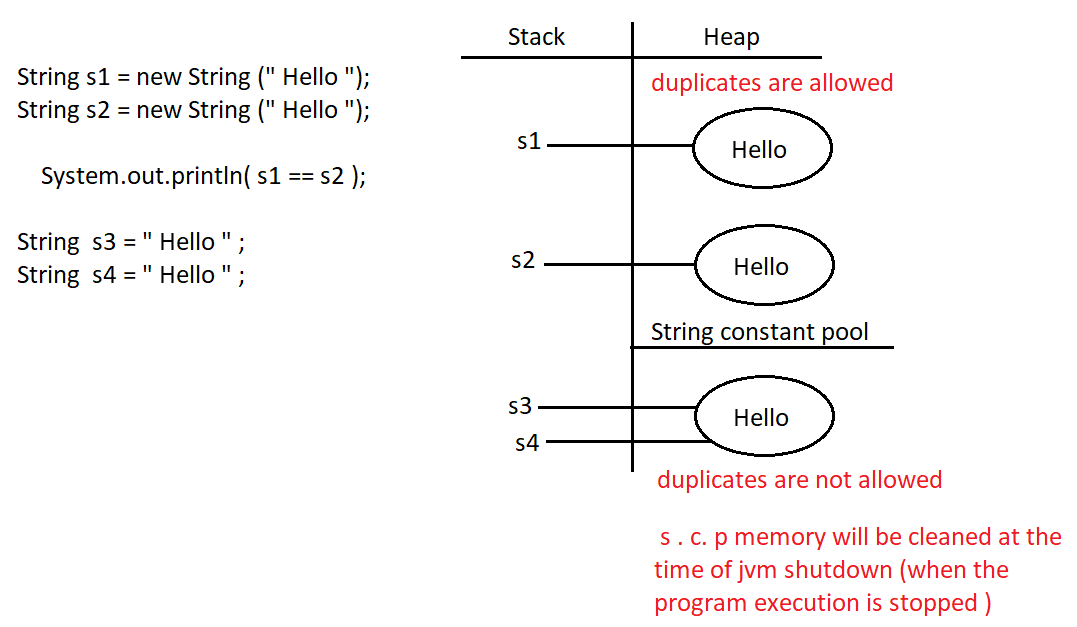
String s1 = new String (“Hello”);

In this case two objects will created one in the heap , and the other in the String constant pool ( s . c . p ) . The reference always points out to the heap.

String s2 = “ Hello”;

In this case one object will be created in the string constant pool and it points out to the refernce





Eg: Strings\_Eg6

Explanation :

Here an object “hello” is created in the heap and its reference point to s1 in stack , the s.c.p also an object is created with same data. And it will have no reference but garbage collector cannot delete it .

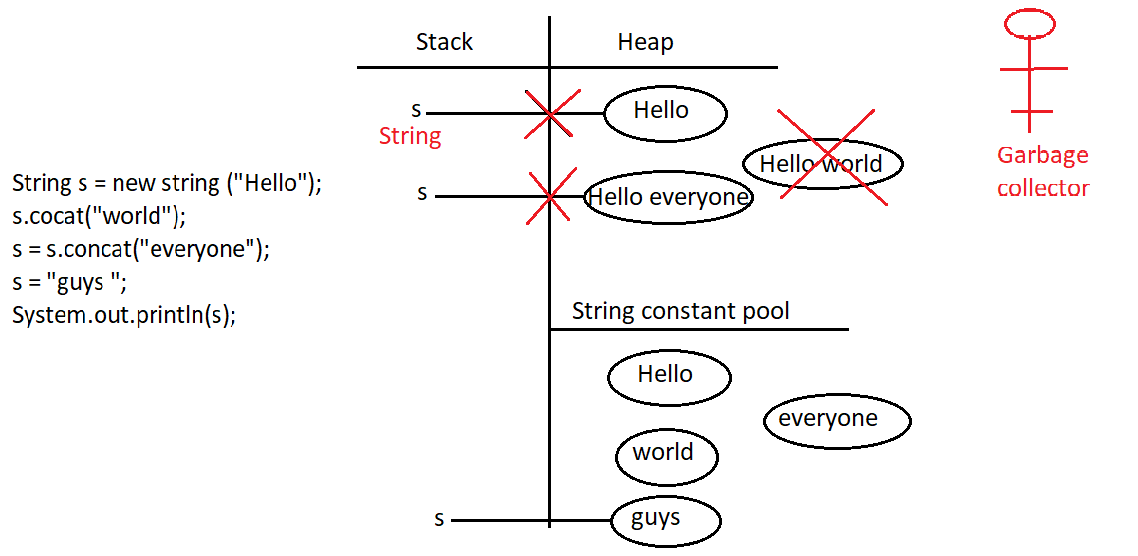
In heap area duplicates are allowed , so another object of “hello” is created in heap and points to s2 , but duplicates are not allowed in the in the s.c.p , since s1 and s2 has object with same data jvm will not create another object in s.c.p .

If new keyword is not used object is created in the s.c.p and it points to reference in stack . Now for s3 there is an object with same data available in s.c.p . so jvm simply points out that object to s3.

Similarly for s4

s.c.p memory will be cleaned only when program execution is stopped (jvm shoutdown) .

So if compared s3 and s4 ( s3==s4 ) output will be true . since same reference variables and same object .

,

Eg: Strings\_Eg7

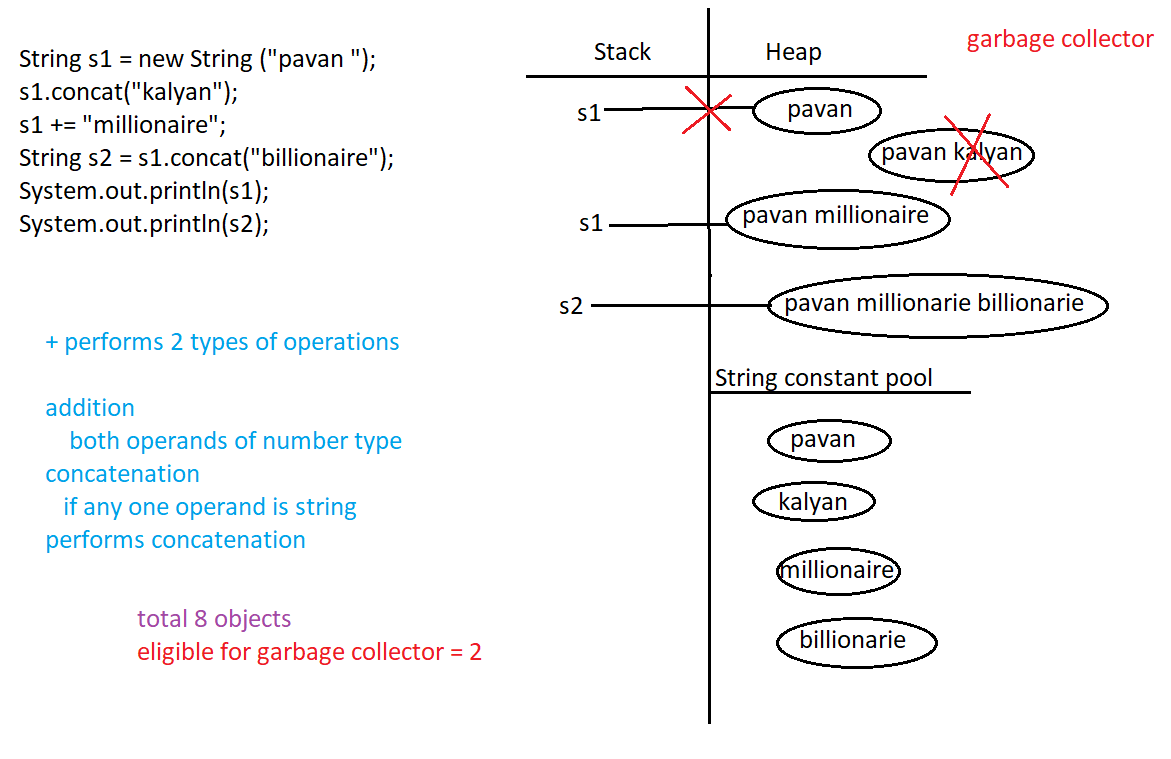
Explanation :

When line 1 is executed ,since there is new keyword the object is created in the heap area and the s.c.p

In line 2 since literal is used, object is created in the s.c.p ,here concat method used on s invoked by jvm during runtime. because of runtime operation if an object has to be created , it will be created compulsorly in heap area . since there is no reference variable to collect it , it is cleaned by the garbage collector .

In line 3 again , since literal is used object will be created in the s.c.p , and we are performing operation during runtime and creating an object , it is place in the heap area , and we are collecting in the reference variable .

In line 4 literals are used ,so object will be created in the s.c.p , and same reference s is used to store the object . So heap area reference will be deleted . if there is no reference for object garbage collector will clear those object.

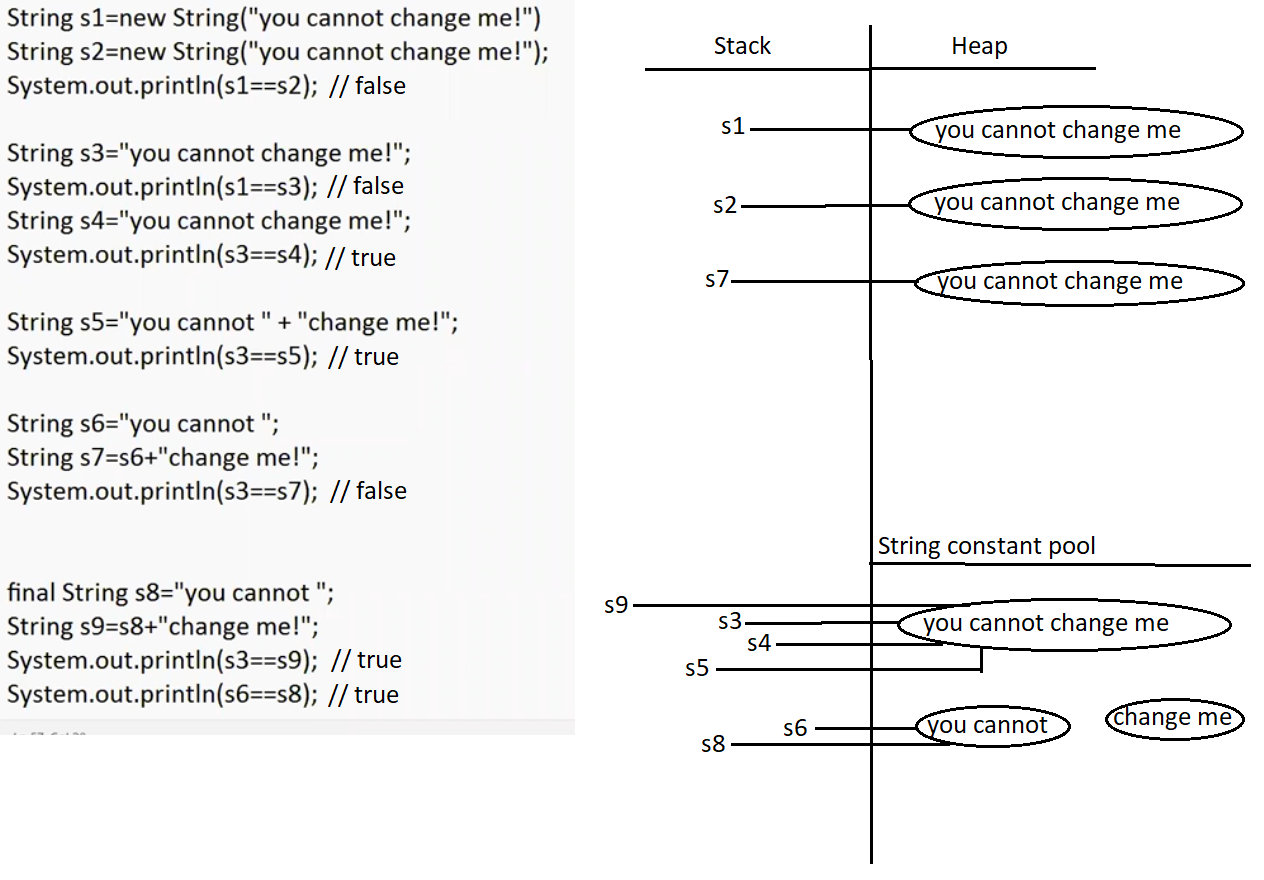


Explanation :

Process is same as previous one .

In line 3 operand is involved that operation is done at runtime , and since operation is done at runtime , jvm creates the memory for it in the heap area .

Eg: String\_Eg8



Explanation :

Process is same

In string s5 since both are literals , they are created in the string constant pool . since there are no variables are operations (methods) to do at the runtime , their evaluation is done at the compile time , and after concatenation memory is also allocated at the string constant pool . since there is already similar data object , it is allocated to s5.

In String s7, literal is create in the s.c.p and since variable is involved , it is evaluated during runtime by jvm, since runtime operation is involved their memory is allocated in the heap area.

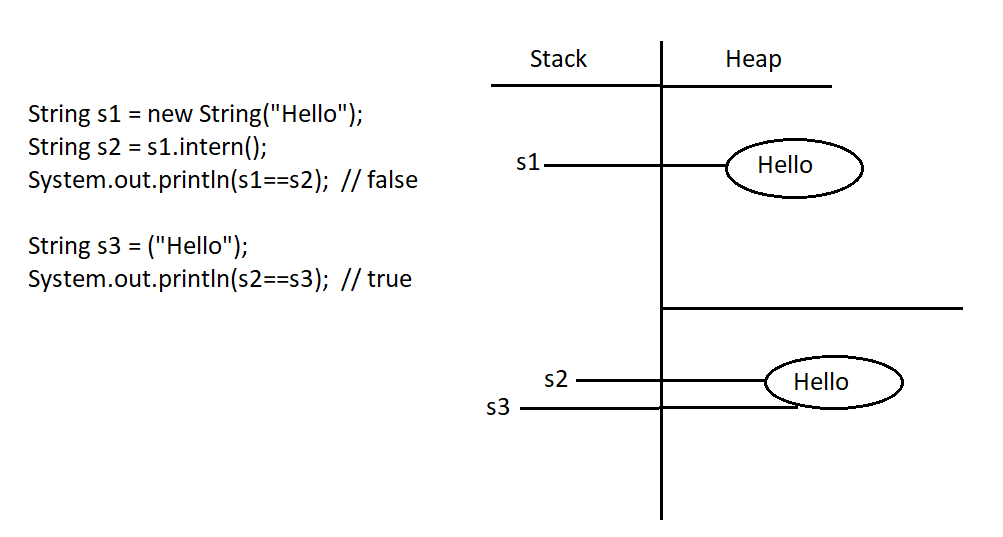
In String s9, since s8 is a final variable it value is known to the compiler , and during compile time it is evaluated and “change me “ is a literal , memory for it is allocated in the s.c.p . And after concatenation of s8 and literal memory for s9 is allocated in s.c.p , since there is no runtime operation involved . Their concatenation result is already present in the s.c.p it is pointed to the s9.

Eg: String\_Eg9

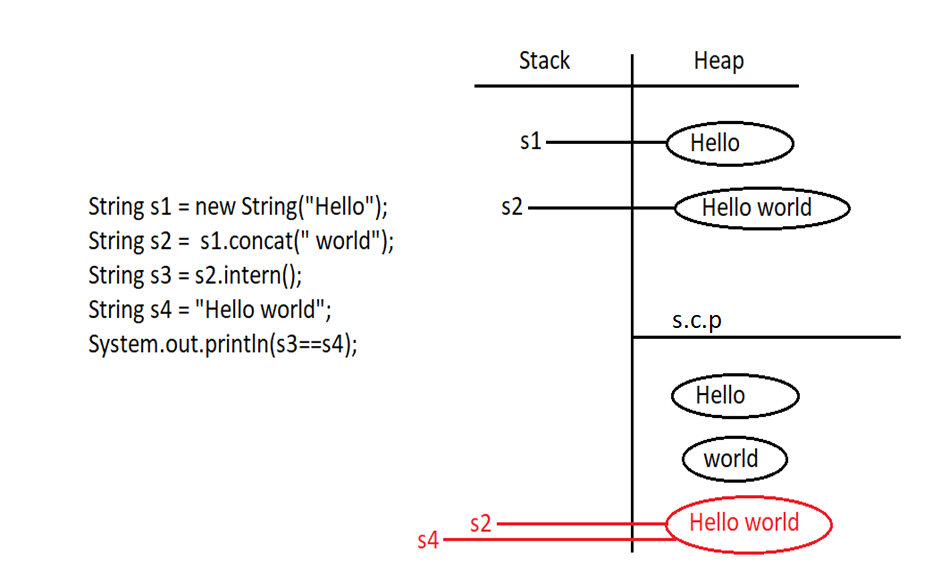
String s1 = new String (“Hello”)

The above line creates two objects one in the heap area and the other in the s.c.p

To check the objects in the s.c.p whether they are present are not we have a method intern().



Eg: String\_Eg10



Explanation :

In line 2 , literal is used so memory for literal “world” is allocated in the s.c.p , and runtime operation (s1.concat(“world”)) is performed so memory for it is allocated in heap area .

In line 3 , intern method is called on s2 , but there is no space allocated in the s.c.p . the role of intern() method is to show something happens in s.c.p . if the reference you are searching is found , you can assign that to another reference variable . if that reference is not found, jvm will create a space for it in the s.c.p just as above ( red colored one is done by jvm)

Eg: Strings\_Eg11