Strings, StringBuffer and StringBuilder(Main Part - 3)

Strings in Java

=> String in java refers to Collection of Characters.

When we can Store collection of characters in char[], why do we need String class in java?

=> if we use String as Array of charactes, then as a java programmer to perform some operation on the array data we need to write a method

=> Writing up a logic and working with that logic is always tough for programmer.

=> To reduce the burden for a developer, SUNMS team had made "Array of characters" data as "Object".

=> Since they made Array of characters as "Object", we need a blue print for an Object called "class" and this blueprint is only "String".

eg#1.

```
char[] name = {'s','a','c','h','i','n'};
System.out.println(name);//sachin
convertToUpper(name);
convertToLower(name);
public void convertToUpper(char[] name)
{
    //logic
}
public void convertToLower(char[]name)
{
    //logic
}
```

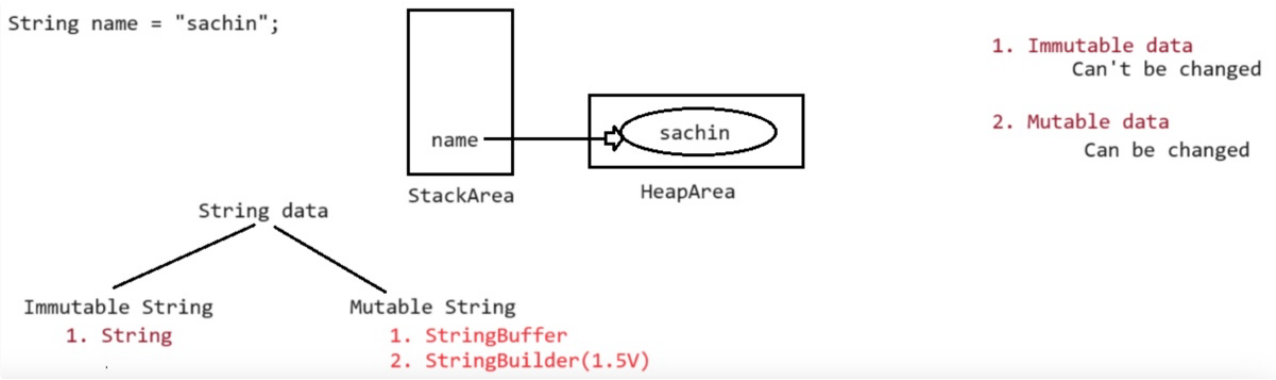
java.lang.String

String it refers to an Object in java present in package called java.lang.String. String refers to collection of characters

eg:: String s= "sachin";

```
System.out.println(s);//sachin
String s =new String("sachin");
System.out.println(s);//sachin
```

In java String object is by default immutable,meaning once the object is created we cannot change the value of the object, if we try to change then those changes will be reflected on the new object not on the existing object.



case 1::

String s= "sachin";

s.concat("tendulkar");(new object got created with modification so immutable)

System.out.println(s);

output::sachin

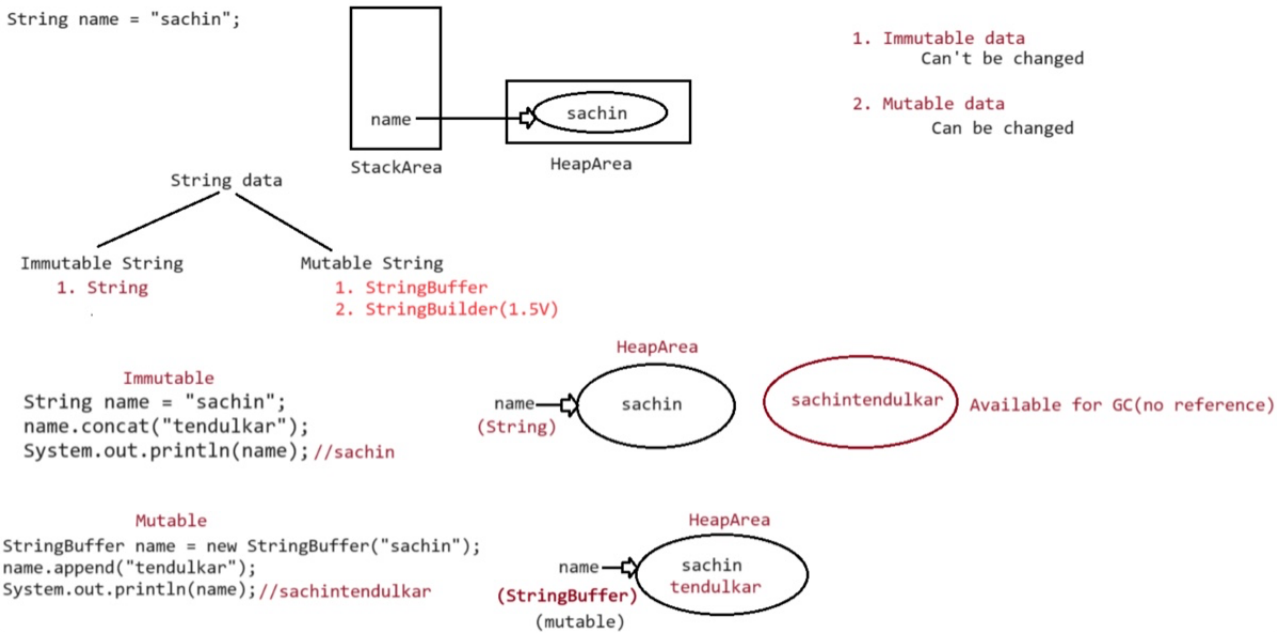
vs

StringBuilder sb=new StringBuilder("sachin");

sb.append("tendulkar");(on the same object modification so mutable)

System.out.println(sb);

output:: sachintendulkar



Note:

public class Object{

public boolean equals(String data){

Compares the reference

if both are equal -> return true

```
        otherwise -> return false
    }
    public String toString(){
        print the address of the object
    }
}
public final class String extends Object{
    public String concat(String data){}
    public String toUpperCase(String data){}
    public String toLowerCase(String data){}
    @Override
    public boolean equals(String data){
        compares the content present inside the object
        if both the data are equal -> return true
        otherwise -> return false
    }
    @Override
    public String toString(){
        //prints the data present inside the object
    }
}
public final class StringBuffer extends Object{
    public String append(String data){}
    public String toUpperCase(String data){}
    public String toLowerCase(String data){}

    //use it from object class
    public boolean equals(String data){
        Compares the reference
        if both are equal -> return true
        otherwise -> return false
    }
    @Override
    public String toString(){
        //prints the data present inside the object
    }
}
```

++

String s1 = new String("sachin");

System.out.println(s1); // when we print a refrence than automatically toString() method is called for that class. This is same for all the refrences not only for strings.

++

case 2::

String s1 = new String("sachin");

String s2 = new String("sachin");

System.out.println(s1==s2); **//false**

System.out.println(s1.equals(s2)); **//true**

=> String class .equals method will compare the content of the object if same return true otherwise return false

vs

StringBuilder sb1 = new StringBuilder("sachin");

StringBuilder sb2 = new StringBuilder("sachin");

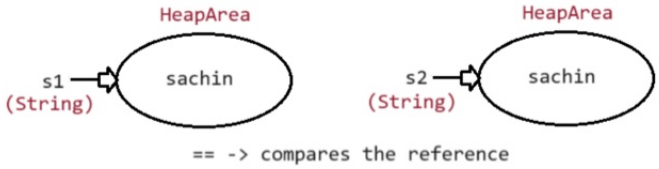
System.out.println(sb1==sb2); **//false**

System.out.println(sb1.equals(s2)); **//false**

=> StringBuilder class .equals method is not overriden so it will use Object class .equals() which is meant for reference comparison. if differnt object returns false,even if the contents are same.

```
String s1 = new String("sachin");
String s2 = new String("sachin");
System.out.println(s1==s2);//false
System.out.println(s1.equals(s2));//true
```

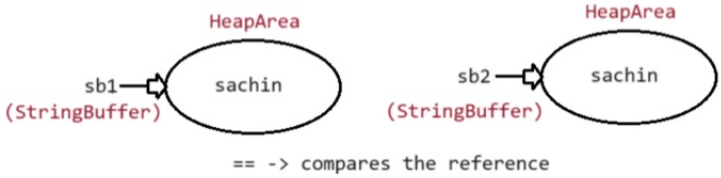
String




```
StringBuffer sb1 =new StringBuffer("sachin");
StringBuffer sb2 =new StringBuffer("sachin");

System.out.println(sb1==sb2); //false
System.out.println(sb1.equals(sb2));//false

System.out.println(sb1); //sachin
System.out.println(sb2); //sachin
```



Snippets

Consider below code of Test.java file:

```
public class Test {
    public static void main(String [] args) {
        boolean flag = false;
        System.out.println((flag = true) | (flag = false) || (flag = true));
```

```
            System.out.println(flag);
    }
}
```

What is the result of compiling and executing Test class?

A. true
   false

B. false
   true

C. true
    true

D. false
   false

E. CompilationError

Answer: A

Consider below code of Test.java file:

```
public class Test {
    public static void main(String [] args) {
        boolean status = true;
        System.out.println(status = false || status = true | status = false);
        System.out.println(status);
    }
}
```

What is the result of compiling and executing Test class?

A. true
   false

B. false
   true

C. true
   true

D. false
   false

E. CompilationError

Answer: E

Q>

Consider below code of Test.java file:

```
public class Test {
    public static void main(String [] args) {
        int a = 3;//2
        int b = 5;//4
        int c = 7;//7
        int d = 9;//9
        boolean res = --a + --b < 1 && c++ + d++ > 1;
        System.out.printf("a= "+a+",b= "+b+",c= "+c+",d="+d+",res="+res);
    }
}
```

A. a=2,b=4,c=7,d=9,res=false

B. a=2,b=4,c=8,d=10,res=false

C. a=2,b=4,c=7,d=9,res=true

D. a=2,b=4,c=8,d=10,res=true

E. a=3,b=5,c=8,d=10,res=false

F. a=3,b=5,c=8,d=10,res=true

Answer: A

res = --a + --b < 1 && c++ + d++ > 1;

= 2 + 4 < 1 && c++ + d++ > 1

= 6 < 1 && c++ + d++ > 1

= false &&

= false

Case3:

String s =new String("sachin");

In this case 2 objects will be created one in the heap and the other one in the String Constant Pool, the reference will always point to Heap.

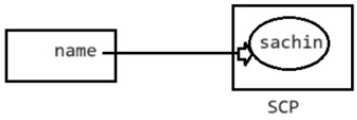
vs

String s ="sachin";

In this case only one object will be created in the SCP and it will be refered by our reference.

```
String name = new String("sachin");
```


```
String name = "sachin";
```



Note:: Object creation in SCP is always optional,1st jvm will check is any object already created with required content or not.

If it is already available then it will reuse the existing object instead of creating the new Object.

If it is not available only then new object will be created, so we say in SCP there is no chance of existing 2 objects with the same content. In SCP duplicates are not permitted.

Garbage Collector cannot access SCP Area, Even though Object does not have any reference still object is not eligible for GC.

All SCP objects will be destroyed only at the time of JVM ShutDown.

eg#1.

String s1 = new String("sachin");

String s2 = new String("sachin");

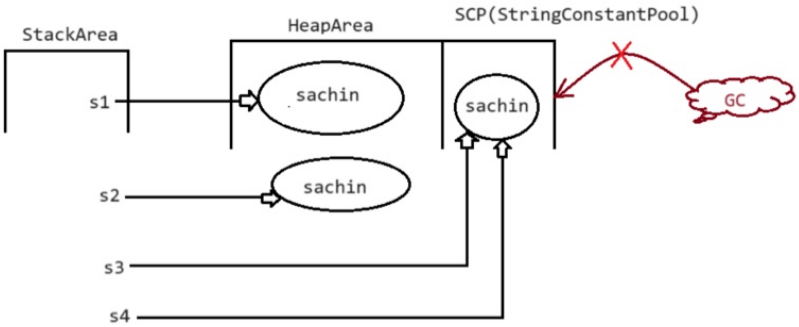
String s3 = "sachin";

String s4 = "sachin";

Output:: Two objects are creted in the heap with data as "sachin" with reference as S1,S2

One object is created in SCP with the reference as S3,S4.

```
String s1 = new String("sachin");
String s2 = new String("sachin");
  String s3 = "sachin";
  String s4 = "sachin";
```



case 4:: String s = new String("sachin");

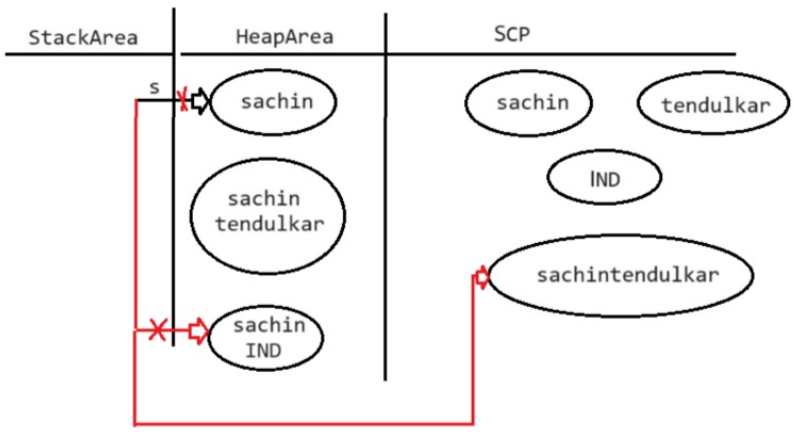
s.concat("tendulkar");

s=s.concat("IND");

s="sachintendulkar";

Output:: Direct literals are always placed in SCP,Because of runtime operation if object is required to create compulsorily that object should be placed on the Heap,but not on SCP.

```
String s = new String("sachin");
   s.concat("tendulkar");
   s=s.concat("IND");
   s="sachintendulkar";
```



case 5::

String s1= new String("sachin");

s1.concat("tendulkar");

s1+="IND";

String s2=s1.concat("MI");

System.out.println(s1);

System.out.println(s2);

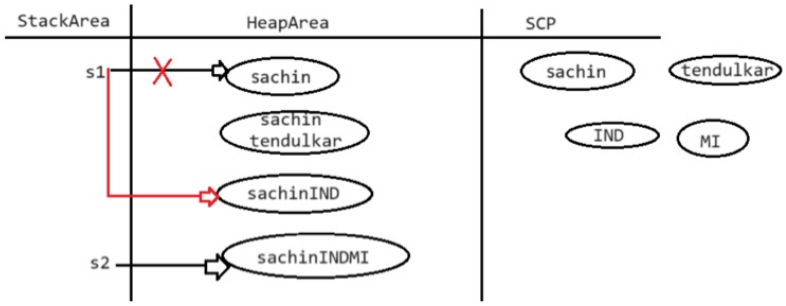
How many objects are eligible for GC?

total :: 8 objects

GC Eligible:: 2 objects

```
String s1= new String("sachin");
     s1.concat("tendulkar");
     s1+="IND";
String s2=s1.concat("MI");
System.out.println(s1);  //sachinIND
System.out.println(s2);  //sachinINDMI

  HeapArea = ? 4
  SCP      = ? 4
  How many are eligible for GC? 2
```



Q>

String s1=new String("you cannot change me!")

String s2=new String("you cannot change me!");

System.out.println(s1==s2);

String s3="you cannot change me!";

System.out.println(s1==s3);

```
String s4="you cannot change me!";
System.out.println(s3==s4);
String s5="you cannot " + "change me!";
System.out.println(s3==s5);
String s6="you cannot ";
String s7=s6+"change me!";
System.out.println(s3==s7);
final String s8="you cannot ";
String s9=s8+"change me!";
System.out.println(s3==s9);
System.out.println(s6==s8);
```

Output

```
false
false
true
true
false
true
true
```

Q>

```
public class Test {
public static void main(String[] args) {
final String fName = "James";
String lName = "Gosling";
String name1 = fName + lName;
String name2 = fName + "Gosling";
String name3 = "James" + "Gosling";
System.out.println(name1 == name2);
System.out.println(name2 == name3);
}
}
```

What will be the result of compiling and executing Test class?

A. true

true

B. true

false

C. false

false

D. false

true

Q> System.out.print(""=="");//true

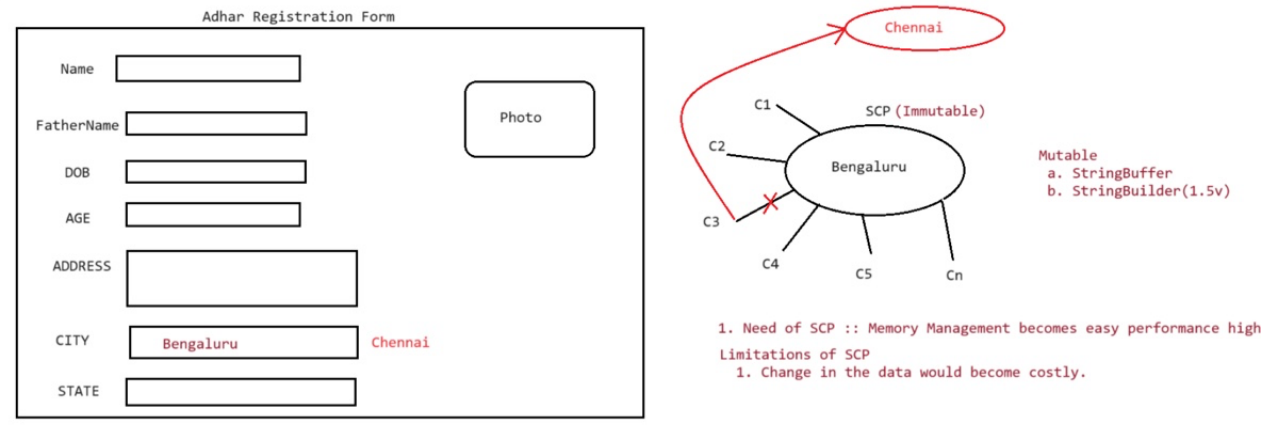
System.out.print(" ");//

System.out.print("A"=="A");//true

System.out.print("aA");//aA

Importance of SCP

1. In our program if any String object is required to use repeatedly then it is not recommended to create multiple object with same content it reduces performance of the system and effects memory utilization.
2. **We can create only one copy and we can reuse the same object for every requirement. This approach improves performance and memory utilization we can achieve this by using "scp".**
3. **In SCP several references pointing to same object the main disadvantage in this approach is by using one reference if we are performing any change the remaining references will be impacted. To overcome this problem SUNMS people implemented immutability concept for String objects.**
4. **According to this once we creates a String object we can't perform any changes in the existing object if we are trying to perform any changes with those changes a new String object will be created hence immutability is the main disadvantage of scp.**



case7 :: Interning=> Using Heap object reference, if we want to get Corresponding SCP Object, then we need to use intern() method.

eg1::

String s1 =new String("sachin"); // One in heap(s1) and the other one in SCP

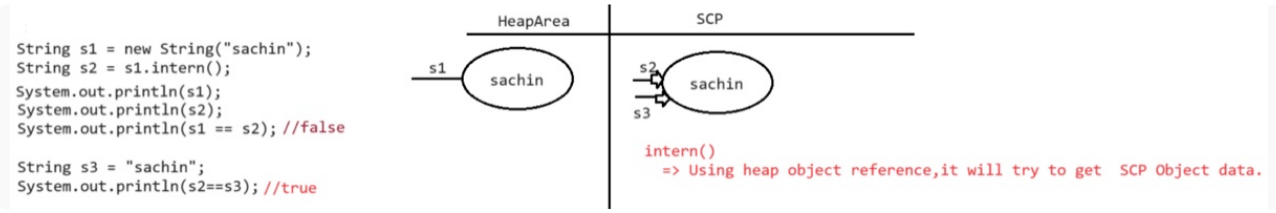
String s2=s1.intern(); //using s1 access object in **SCP which has no reference**

System.out.println(s1==s2);//false

String s3="sachin";

System.out.println(s2==s3); //true

Using heap object reference, if we want to get the corresponding SCP object and if the Object does not exists, then intern() will create a new object in SCP and it returns.



eg2::

String s1=new String("sachin");// One in heap(s1) and the other one in SCP

String s2=s1.concat("IND");// One in SCP(IND) and the other one in heap(s2)

String s3=s2.intern();

String s4="sachinIND";

System.out.println(s1 == s3);//false

System.out.println(s2 == s3);//false

System.out.println(s3 == s4);//true

String class Constructor

String s =new String() **=> Creates an Empty String Object**

String s =new String(String literals) **=> Creates an Object with String literals on Heap**

String s =new String(StringBuffer sb) **=> Creates an equivalent String object for StringBuffer**

String s =new String(char[] ch) **=> Creates an equivalent String object for character array**

String s =new String(byte[] b) **=> Creates an equivalent String object for byte array**

eg:

char[] ch={'a','b','c'} ;

String s=new String(ch);

System.out.println(s);//abc

eg:

byte[] b={100,101,102};

String s=new String(b);

System.out.println(s)//def

Q>

```
public class DemoApp{
    public static void main(String... args){
        if(args[0].equals("hello") ? false : true)
            System.out.println("success");
        else
            System.out.println("failure");
    }
}
```

What is the output if the program is executed in the following style?

```
java DemoApp hello
     |
    DemoApp.main(new String{"hello"})
```

A. success

B. failure

C. CE

D. ArrayIndexOutOfBoundsException

E. StringIndexOutOfBoundsException

Answer: B (if(false))

Important methods of String

1.public char charAt(int index)

2.public String concat(String str)

3.public boolean equals(Object o)

4.public boolean equalsIgnoreCase(String s)

5.public String subString(int begin)

6.public String subString(int begin,int end)

7.public int length()

8.public String replace(char old,char new)

9.public String toLowerCase()

10.public String toUpperCase()

11.public String trim()

12.public int indexOf(char ch)

13.public int lastIndexOf(char ch)

Important methods of String

Note: **Even though String data is stored internally as "Arrays", as a programmer we can't access the data at the index level directly. we need to use methods.**

1.public char charAt(int index)

eg:: String s="sachin";

System.out.print(s.charAt(0));//s

System.out.print(s.charAt(-1));**//StringArrayIndexOutOfBoundsException**

System.out.print(s.charAt(10));**//StringArrayIndexOutOfBoundsException**

2.public String concat(String str)

eg::

String s="sachin";

System.out.println(s.concat("tendulkar")); **//sachintendulkar**

s+="IND";
s=s+"MI";
System.out.print(s); **//sachinINDMI**

3.public boolean equals(Object o)

It is used for Content Comparison,In String class equals() method is Overriden to check the content of the object

4.public String subString(int begin)

It gives the String from the begin index to end of the String.
String s="Ineeuron";
System.out.print(s.substring(2)); **//searching from 2 to end of the string**

5.public String subString(int begin,int end)

It gives the String from the begin index to end-1 of the String.
String s="Ineeuron";
System.out.print(s.substring(2,6)); **//searching from 2 to 5 will happen**

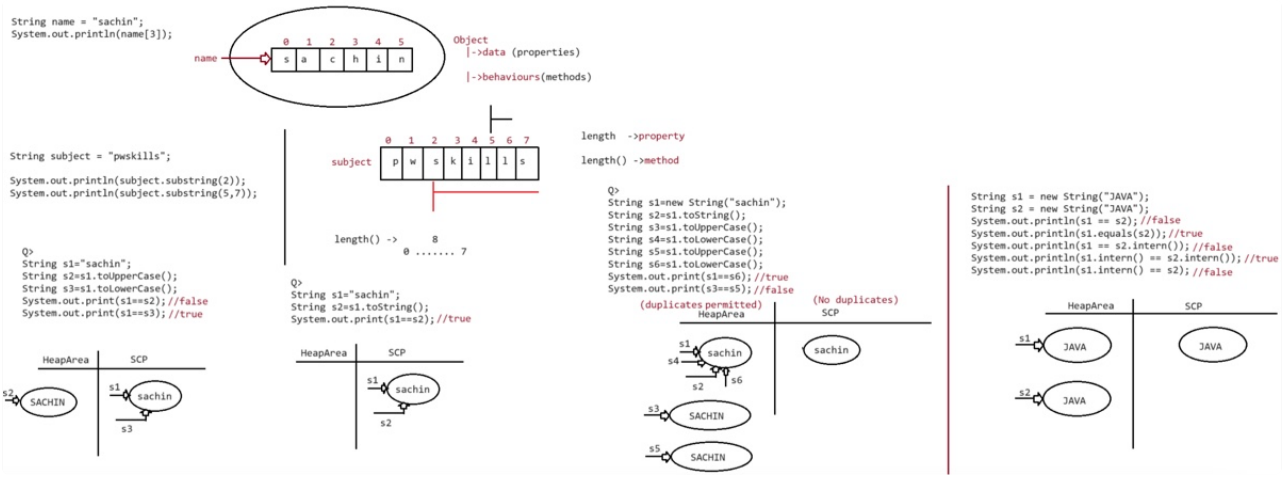
6.public int length()

It returns the no of characters present in the String.
String s="pwskills";
System.out.print(s.length()); **//8**
System.out.print(s.length); **//Compile time error**

8.public String replace(char old,char new)

String s="abababab";
System.out.print(s.replace('a','b')); **//bbbbbb**

9.public String toLowerCase()
10.public String toUpperCase()



String internally is stored in the form of array so then will name[0] (access by index) will work??
Ans - No as at programmer level string is an object and we can access objects only with properties and methods.

Predict the output

Q>

String s1="sachin";

String s2=s1.toUpperCase();

String s3=s1.toLowerCase();

System.out.print(s1==s2);

System.out.print(s1==s3);

Q>

String s1="sachin";

String s2=s1.toString();

System.out.print(s1==s2);

Q>

String s1=new String("sachin");

String s2=s1.toString();

String s3=s1.toUpperCase();

String s4=s1.toLowerCase();

String s5=s1.toUpperCase();

String s6=s1.toLowerCase();

System.out.print(s1==s6);

System.out.print(s3==s5);

Q>

String string = "string".replace('i', '0'); str0ng

System.out.println(string.substring(2, 5));

Q>

String s1 = new String("JAVA");

String s2 = new String("JAVA");

System.out.println(s1 == s2);

System.out.println(s1.equals(s2));

System.out.println(s1 == s2.intern());

System.out.println(s1.intern() == s2.intern());

System.out.println(s1.intern() == s2);

11.public String trim()

To remove the blank spaces present at the begining and end of string but not the blank spaces present at the middle of the String.

12.public int indexOf(char ch)

It returns the index of 1st occurance of the specified character if the specified character is not available then it returns -1.

String s="sachinramesh";

System.out.print(s.indexOf('a')); //1

System.out.print(s.indexOf('z')); //-1

13.public int lastIndexOf(char ch)

It returns the index of last occurance of the specified character if the specified character is not available then it returns -1.

String s="sachinramesh";

System.out.print(s.lastIndexOf('a'));//7

System.out.print(s.lastIndexOf('z'));//-1

=> Because of runtime operation, if there is a change in the content with those changes a new Object will be created only on the heap, but not in SCP.

=> If there is no change then the same object will be reused.

=> This rule is applicable for Objects present in both SCP and Heap.

Note: isBlank vs isEmpty

```
String name = "";
System.out.println(name.isEmpty());  //true
System.out.println(name.isBlank());   //true

System.out.println("*****************");

String data = " ";
System.out.println(data.isEmpty());  //false
System.out.println(data.isBlank());   //true
```

Q>

```
String str =" ";
str.trim();
System.out.println(str.equals("")+ " " + str.isEmpty());
```

What is the result?

A. true false

B. true true

C. false true

D. false false

Answer: D(Because String is immutable)

Q>

```
String s = "SACHIN TENDULKAR";
int len= s.trim().length();
System.out.println(len);
```

What is the result?

A. 10

B. 9

C. 8

D. compilation fails

E. 15

F. 16

Answer: F

Q> String s= "Hello world";

s.trim();

int i = s.indexOf(" ");

System.out.println(i);

What is the result?

A. Exception at runtime

B. -1

C. 5

D. 0

Answer: C

Q> String s1= "Java";

String s2=new String("java");

//line-1

{

System.out.println("equal");

}

else

{

System.out.println("not equal");

}

To print equal which code fragment should be inserted?

A. s1=s2;

if(s1==s2)

B. if(s1.equalsIgnoreCase(s2))

C. String s3= s2;

if(s3.equalsIgnoreCase(s3))

D. if(s1.toLowerCase() == s2.toLowerCase())

Answer : A,B,C

String str = " PW\tSkills\tPrivateLmtd\tKnown\tfor\tjava ".strip();

System.out.println(str); //PW Skills Private Lmtd Known for java

if("string".toUpperCase() == "STRING"){

    System.out.println(true);

```
}else{
    System.out.println(false);  //false
}
String str1 = "1";
String str2 = "22";
String str3 = "333";
System.out.println(str1.concat(str2).concat(str3).repeat(3)); // 122333122333122333
```

Mutable

=> If we try to make a change with that change no new object will be created, changes will happen on the same object.

=> To create a mutable String in java we have 2 classes

a. StringBuffer

b. StringBuilder

StringBuffer

1. If the content will change frequently then it is not recomonded to go for String object becoz for every new change a new Object will be created.
2. To handle this type of requirement, we have StringBuffer/StringBuilder concept

1. **StringBuffer sb=new StringBuffer();**

creates a empty StringBuffer object with default intital capacity of "16".

Once StringBuffer reaches its maximum capacity a new StringBuffer Object will be created new **capacity = (currentcapacity+1) * 2;**

eg1::

```
StringBuffer sb = new StringBuffer();
System.out.println(sb.capacity());//16
sb.append("abcdefghijklmnop");
System.out.println(sb.capacity());//16
sb.append('q');
System.out.println(sb.capacity());//34
```

2. **StringBuffer sb=new StringBuffer(initalCapacity);**

It creates an Empty String with the specified inital capacity.

eg1::

```
StringBuffer sb = new StringBuffer(19);
```

System.out.println(sb.capacity());//19

3. **StringBuffer sb=new StringBuffer(String s);**

It creates a StringBuffer object for the given String with the capacity = s.length() + 16;

eg1::

StringBuffer sb = new StringBuffer("sachin");

System.out.println(sb.capacity());//22

When to go for String and When to go for StringBuffer?

String -> Frequently if there is no change of data then go for String[Immutable]

StringBuffer -> Frequently if there is a change in the data then go for StringBuffer(mutable)

Important methods of StringBuffer

a. public int length()

b. public int capacity()

c. public char charAt(int index)

d. public void setCharAt(int index,char ch)

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("sachin");
        System.out.println(sb.length());
        System.out.println(sb.capacity());
        System.out.println(sb.charAt(3));
        System.out.println(sb.charAt(30));
    }
}
```

Output

6

22

h

Exception in thread "main" java.lang.StringIndexOutOfBoundsException

eg#2.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
        sb.setCharAt(8,'B');
        System.out.println(sb);
    }
}
```

Output

sachinraBeshtendulkar

e. public StringBuffer append(String s)

f. public StringBuffer append(int i)

g. public StringBuffer append(long l)

h. public StringBuffer append(boolean b)

i. public StringBuffer append(double d)

j. public StringBuffer append(float f)

k. public StringBuffer append(int index,Object o)

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer();
        sb.append("The value of PIE is :: ");
        sb.append(3.1414);
        sb.append(" This is exactly" );
        sb.append(true);
        System.out.println(sb);
    }
}
```

Output

The value of PIE is :: 3.1414 This is exactlytrue

Overloaded method

l. public StringBuffer insert(int index,String s)

m. public StringBuffer insert(int index,int i)

n. public StringBuffer insert(int index,long l)

o. public StringBuffer insert(int index,double d)

p. public StringBuffer insert(int index,boolean b)

q. public StringBuffer insert(int index,float s)

r. public StringBuffer insert(int index,Object o)

To insert the String at the specified position we use insert method

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("abcdefgh");
        sb.insert(2,"xyz");//abxyzdefgh
        sb.insert(11,"9");
        System.out.println(sb);//abxyzdefgh9
    }
}
```

Output

abxyzcdefgh9

Methods of StringBuffer

public StringBuffer delete(int begin,int end)

It deletes the character from specified index to end-1.

public StringBuffer deleteCharAt(int index)

It deletes the character at the specified index.

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("sachintendulkar");
        System.out.println(sb);  //sachintendulkar
        sb.delete(6,12);  //tendul
        System.out.println(sb);   //sachinkar
        sb.deleteCharAt(6);
        System.out.println(sb);   //sachinar
    }
}
```

Output

sachintendulkar

sachinkar

sachinar

public StringBuffer reverse()

It is used to reverse the given String.

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("pwskills");
        sb.reverse();
        System.out.println(sb);  //sllikswp
    }
}
```

public void setLength(int Length)

It is used to consider only the specified no of characters and remove all the remaining characters.

public void trimToSize()

This method is used to deallocate the extra allocated free memory such that capacity and size are equal.

public void ensureCapacity(int capacity)

It is used to increase the capacity dynamically based on our requirement.

eg#1.

```
class Test{
    public static void main(String... args) {
        StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
        sb.setLength(6);
        System.out.println(sb);

        System.out.println();

        StringBuffer sb1 =new StringBuffer(100000);
        System.out.println(sb1.capacity());
        sb1.append("sachin");
        System.out.println(sb1.capacity());

        sb1.trimToSize();
        System.out.println(sb1.capacity());

        System.out.println();
```

```
        StringBuffer sb2 = new StringBuffer();
        System.out.println(sb2.capacity());
        sb2.ensureCapacity(10000);
        System.out.println(sb2.capacity());
    }
}
```

Output

sachin

100000

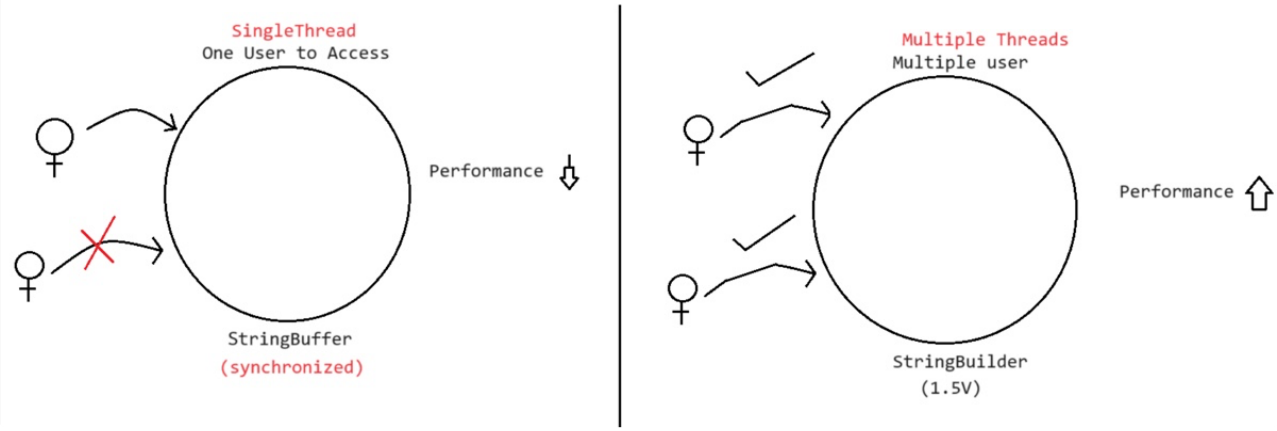
100000

6

16

10000

EveryMethod present in StringBuffer is synchronized, so at a time only one thread can are allowed to operate on StringBuffer Object, it would increase the waiting time of the threads it would create performance problems, to overcome this problem we should go for StringBuilder.



StringBuilder(1.5v)

StringBuilder is same as StringBuffer(1.0V) with few differences

StringBuilder

No methods are synchronized

At at time more than one thread can operate so it is not ThreadSafe.

Threads are not requried to wait so performance is high.

Introduced in jdk1.5 version

StringBuilder vs StringBuffer

StringBuffer

-> JDK1.0 version

-> Every method present in StringBuffers is synchronized

-> Only one thread is allowed to operate on StringBuffer object.

-> ThreadSafety.

-> increases the waiting time so performance is low.

StringBuilder

-> JDK1.5 version

-> Every method present in StringBuilder is not synchronized.

-> Multiple thread are allowed to operate on StringBuffer object.

-> Not ThreadSafety.

-> no waiting so performance is high.

When to go for String,StringBuffer,StringBuilder

String vs StringBuffer vs StringBuilder

String => we opt if the content is fixed and it wont change frequently

StringBuffer => we opt if the content changes frequently **but ThreadSafety is required**

StringBuilder => we opt if the content changes frequently **but ThreadSafety is not required**

MethodChaining

Most of the methods in String,StringBuilder,StringBuffer **return the same type only,** hence after **applying method on the result we can call another method which forms method chaining.**

eg::

```
StringBuffer sb = new StringBuffer();
sb.append("sachin").insert(6, "tendulkar").reverse().append("IND").delete(0, 4).reverse();
System.out.println(sb);
```

final vs Immutability

=> final is a modifer applicable for variables, where as immutability is applicable only for Objects.

=> If reference variable is declared as final,it means we cannot perform reAssignment for the reference variable, it doesnot mean we cannot perform any change in that object.

=> By declaring a reference variable as final, we wont get immutablity nature.

=> final and Immutablity is differnt concept.

eg::

```
final StringBuilder sb=new StringBuilder("sachin");
sb.append("tendulkar");
System.out.println(sb);
```

sb=new StringBuilder("dhoni"); **//CE::Cannot assign a value to final variable**

Note::

final variable(valid),final object(invalid),immutable variable(invalid), immutable object(valid).

StringBuilder,StringBuffer is Mutable.

All Wrapper classes(Integer,Float,Double,Byte,Short,Character,Boolean) and String are by default Immutable.

- **immutable variable (invalid):** Similar to final object, immutability is typically a property of objects, not individual variables. A variable can hold a reference to an immutable object, but the variable itself can point to a different object.
- **immutable object (valid):** This is a very important concept. An immutable object is an object whose state cannot be modified after its creation. Any attempt to change its internal data will result in a new object being created with the updated data. This ensures data integrity and simplifies reasoning about object behavior.
- **final object (invalid):** This concept doesn't exist in most programming languages. There's no direct way to declare an object as final. However, you can achieve immutability in objects, which is explained below.

Question::

1. Difference b/w String and StringBuilder?
2. Difference b/w String and StringBuffer?
3. Other than Immutablity and Mutablity what is the difference b/w String and StringBuffer?
4. What is SCP?
5. What is the advantage of SCP?
6. What is the disadvantage of SCP?
7. Why SCP is applicable only for String and not for StringBuilder?
8. Is their any Object which is Immutable just like String?
9. What is interning?
10. 10.Difference b/w final and Immutablity?

Questions

String s1 = "null"+null+1;

System.out.println(s1);

Output : nullnull1

String s1 = 1+null+"null";

System.out.println(s1);

Output: CE: bad operand type '+'

String str = "sachin ramesh tendulkar";

System.out.println(str.indexOf('a') + str.indexOf("dulkar"));//18

String s1="sachinrameshtendulkar";

System.out.println(s1.replace('a', 'A').indexOf('a'));//-1

String str = "pwskills Private Limited";

System.out.println(str.indexOf('i', 5));//5

String str = "ineurontechnologyprivatelimited";

System.out.println(str.charAt(str.length()));//SIOBE

StringBuilder sb = new StringBuilder(-32);

sb.append("ABC");

System.out.println(sb);//NegativeArraySizeException

Q>

StringBuilder sb = new StringBuilder("0123456789");

System.out.println(sb.delete(3, 6).deleteCharAt(4).deleteCharAt(5));// 01268

Q>

String str1 = "123321123";

System.out.println(str1.replaceFirst("123", "321").replaceAll("12", "21").substring(3, 6));//321321213 => 321

Q>

String str1 = "OnE tWo ThReE fOuR";

String str2 = "oNeTwOtHrEeFoUr";

System.out.println(str1.trim().equalsIgnoreCase(str2));//false

Q>

StringBuffer sb = new StringBuffer("11111");

System.out.println(sb.insert(3, false).insert(5, 3.3).insert(7, "One"));//111fa3.One3lse11

Q>

String str1 = "Java J2EE Spring Hibernate SQL";

String str2 = "Python Java Scala C C++";

System.out.println(str1.contains("HTML") == str2.contains("HTML"));//true

Q>

StringBuffer sb = new StringBuffer("One Two Three Four Five");

System.out.println(sb.reverse().indexOf("Two"));//-1