

Squinting

27 Nov - 2022

String

java.lang.String

String it refers to an Object in java present in package called java.lang.String(C)
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 same 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
```

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

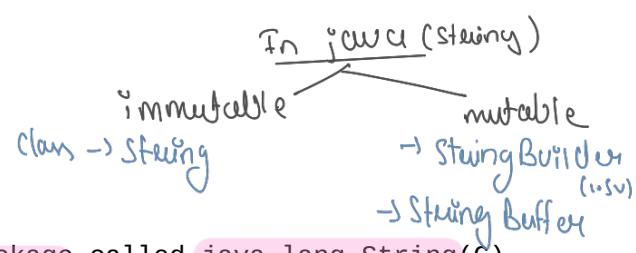
```
StringBuilder sb1 = new StringBuilder("sachin");
StringBuilder sb2 = new StringBuilder("sachin");
System.out.println(sb1==sb2); //false
System.out.println(sb1.equals(sb2)); //false
=> StringBuilder class .equals method for reference comparison
if different object returns false, even if the contents are same.
```

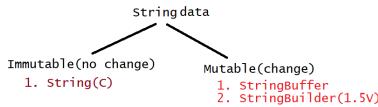
case 3:: String s =new String("sachin");

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

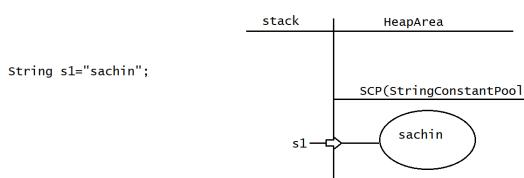
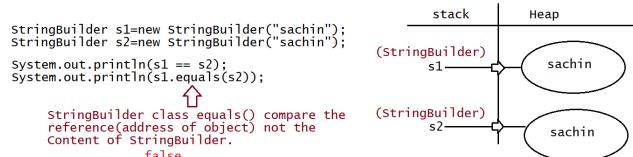
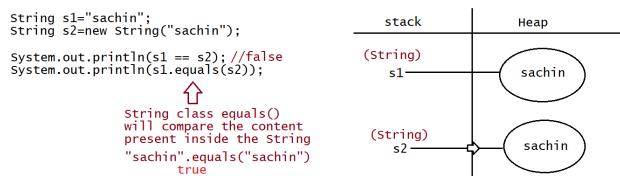
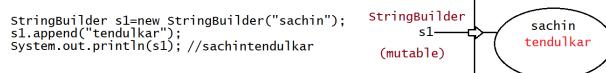
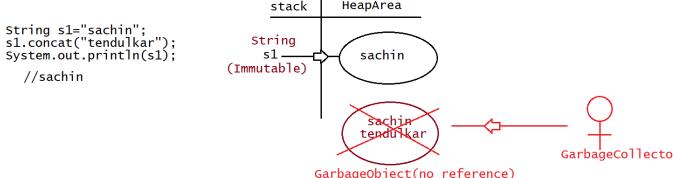
```
String s ="sachin";
In this case only one object will be created in the SCP and it will be
referred
by own reference.
```





```

class String
{
    //instance variables
    //methods
}
    
```



3rd Nov 2022

Note:: Object creation in SCP is always optional, 1st JVM will check if 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:: String s1=new String("dhoni");
      String s2=new String("dhoni");
      String s3="dhoni";
      String s4="dhoni";
```

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

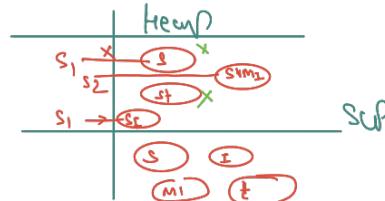
One object is created in SCP with the reference as S3,S4.

```
case 4:: String s = new String("sachin");
           s.concat("tendulkar");
           s=s.concat("IND");
           s="sachintendulkar";
```

JVM

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.

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



Find out totally how many objects are created and how many are eligible for garbage collection?

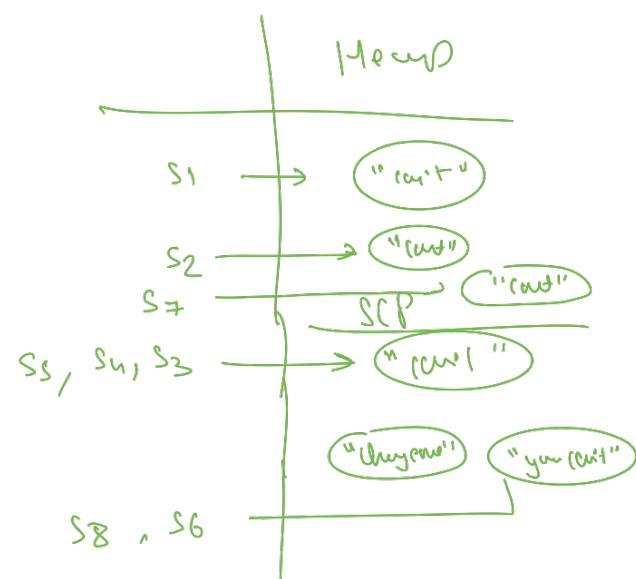
output: 8 and gc eligible is 2

```
Q>
String s1=new String("you cannot change me!");
String s2=new String("you cannot change me!");
System.out.println(s1==s2); false
```

```
String s3="you cannot change me!";
System.out.println(s1==s3); false
String s4="you cannot change me!";
System.out.println(s3==s4); false
```

```
String s5="you cannot " + "change me!"; compiler will check only
System.out.println(s3==s5); true
```

```
String s6="you cannot ";
String s7=s6+"change me!";
System.out.println(s3==s7); false
```



```

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

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

```

eg2::

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(s3 == s4); //true

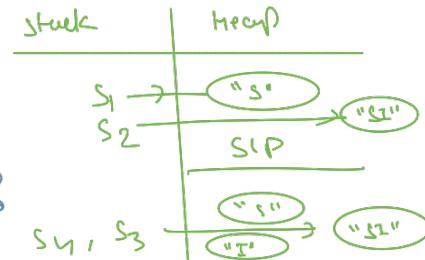
```

why SCP for String?

Note:
Importance of SCP

why mutable & immutable?

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 sun 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.**



Note: if the class name and method name is same in a class then that method is called "Constructor".

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  
eg: String str = new String("sachin");  
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 = {'j','a','v','a'};  
String s1 =new String(ch);  
System.out.println(s1);//java
```

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

```
byte[] b = {65,66,67,68};  
String s2= new String(b);  
System.out.println(s2);//"ABCD"
```

```
StringBuffer sb =new StringBuffer("sachin");  
System.out.println("StringBuffer data is :: "+sb);
```

```
String s1 =new String(sb);  
System.out.println("String data is : "+s1);
```

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)

Note:

```
package java.lang;  
class String  
{  
    //method name  
    public int length()  
  
}
```

```
Integer Array Class(proxy class)
=====
class [I
{
    //property name
    int length;
}

eg:
String s ="sachin";
System.out.println(s.length());//6

int[] arr= {10,20,30};
System.out.println(arr.length());//3
```

```

String s1=new String("dhoni");
String s2=new String("dhoni");
System.out.println(s1==s2); //false
String s3="dhoni";
String s4="dhoni";
System.out.println(s3==s4); //true

```

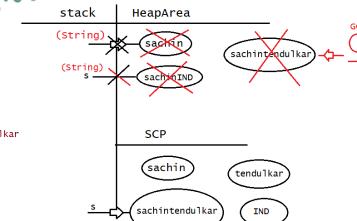
memory will be cleaned at the time of jvm shutdown

*Immutable means
not change in same object.*

```

String s = new String("sachin");
s.concat("tendulkar");
s=s.concat("IND");
s="sachintendulkar";
System.out.println(s); //sachintendulkar

```



```

String s1= new String("sachin");
s1.concat("tendulkar");
s1="IND"; s1 = s1 + "IND";
String s2=s1.concat("IND");

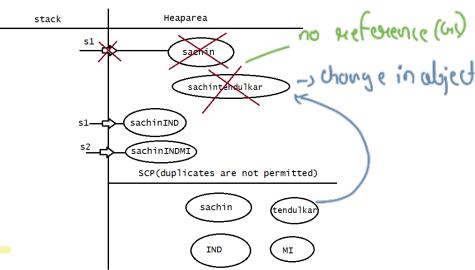
System.out.println(s1); //sachinIND
System.out.println(s2); //sachinINDME

```

```

String + "IND"
↑
addition
both operands are number type
addition
if one operand is string then concatenation

```

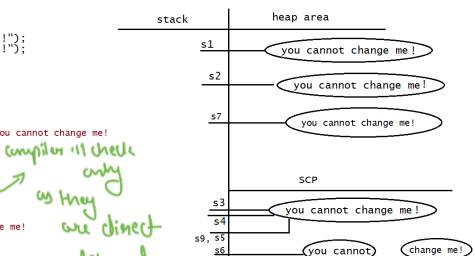


```

String s1=new String("you cannot change me!");
String s2=new.String("you cannot change me!");
System.out.println(s1==s2); //false
String s3="you cannot change me!";
System.out.println(s1==s3); //false
String s4="you cannot change me!";
System.out.println(s3==s4); //true
String s5="you cannot " + "change me!"; //you cannot change me!
System.out.println(s3==s5); //true
String s6="you cannot ";
String s7=s6+"change me!"; //you cannot change me!
System.out.println(s3==s7); //false
final String s8="you cannot ";
String s9=s8+"change me!"; //you cannot change me!
System.out.println(s3==s9); //true
System.out.println(s6==s8); //true

```

computer will check only they are direct literal.



```

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

```

```

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

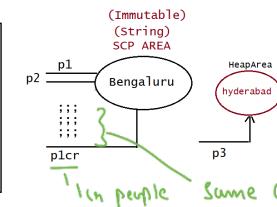
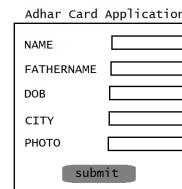
```

```

String s1 = new String("sachin");
String s2 = s1.concat(' tendulkar');
String s3 = s2.intern();
String s4 = "sachintendulkar";
System.out.println(s3==s4); //true

```

Need of SCP

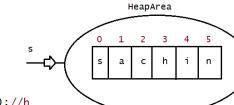


*same data but
multiple object*

```

String s = new String("sachin");
System.out.println(s[3]); //CE
System.out.println(s.charAt(3)); //h
System.out.println(s.charAt(-1)); //stringIndexOutOfBoundsException
System.out.println(s.charAt(500)); //stringIndexOutOfBoundsException

```



why SCP for String?

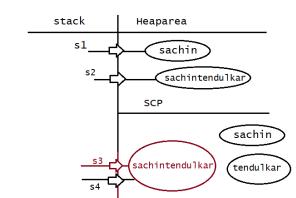
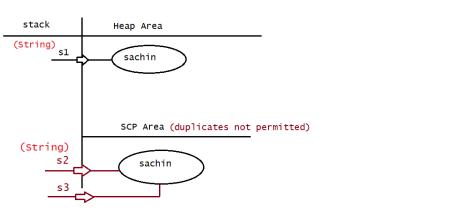
why mutable & immutable?

ScP is
*I copy pointed by
multiple objects in ur
area.*

total objects = 8(4+4)

eligible for gc = 2

A/A



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)

```

3. public boolean equals(Object o)

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

4. public boolean equalsIgnoreCase(String s)

It is used for Content Comparison without comparing the case.

eg#1.

```

public class Test {
    public static void main(String[] args) {
        String s ="java";
        System.out.println(s.equals("JAVA")); //false
        System.out.println(s.equalsIgnoreCase("JAVA")); //true
    }
}

```

Assignment

```

credentials(gmail)
username:nitin@ineuron.ai(not case sensitive)
password :******(case sensitive)

```

5. 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

```

6. 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

```

eg#1.

```

public class Test {
    public static void main(String[] args) {
        String s ="0123456789sachinINDI";
        System.out.println(s.length());
        System.out.println();
        System.out.println(s.substring(9)); // MI
    }
}

```

(2,6)

```

        System.out.println(s.substring(0,8)); sachin IN
        System.out.println(s.substring(0,9)); sachin IN
    }

}

```

```

8. public String replace(char old,char new)
    String s="ababab";
    System.out.print(s.replace('a','b')); //bbbbbb

```

eg#1.

```

public class Test {
    public static void main(String[] args) {
        String name ="sbchin";
        System.out.println(name.replace('b','a'));//sachin

        String data = "ababab";
        System.out.println(data.replace('a','b'));//bbbbbb
    }
}

```

all a replace with b

```

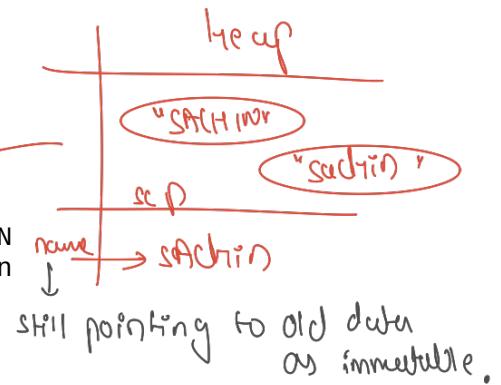
9. public String toLowerCase()
10. public String toUpperCase()

```

```

eg:
public class Test {
    public static void main(String[] args) {
        String name ="sAchIn"; //mixedCase
        System.out.println(name.toUpperCase()); //SACHIN
        System.out.println(name.toLowerCase()); //sachin
    }
}

```



```

11. public String toString();

```

Note: When ever we print any reference, by default JVM will call
toString() on the reference

```

eg: System.out.println(name);
    System.out.println(name.toString());

```

```

eg#1.
//userdefined class
class Student{
    String name = "sachin";
    int id = 10;
}
public class Test {
    public static void main(String[] args) {

        Student std = new Student();
        System.out.println(std); //Student@HexaDecimalValue
        System.out.println(std.toString()); //Student@HexaDecimalValue

        System.out.println();

        String name =new String("dhoni");
        System.out.println(name); //dhoni
        System.out.println(name.toString()); //dhoni
    }
}

```

```
}
```

Predict the output

```
=====
```

Q>

```
String s1="sachin"; // s1,s3 -> sachin (SCP)
String s2=s1.toUpperCase(); // s2->SACHIN(heap area)
String s3=s1.toLowerCase();
System.out.print(s1==s2); //false
System.out.print(s1==s3); //true
```

Note: object upon runtime open, new object will be created in

Heap only & only if there is a change in an object.

```
Q> String s1="sachin"; // s1,s2-> sachin (SCP)
String s2=s1.toString();
System.out.print(s1==s2); //true
```

```
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); //true
System.out.print(s3==s5); //false
```

final vs Immutability

```
=====
```

=> final is a modifier applicable for variables, whereas immutability is applicable only for Objects.

=> If reference variable is declared as final, it means we cannot perform reAssignment for the reference variable,

it does not mean we cannot perform any change in that object.

=> By declaring a reference variable as final, we won't get immutability nature.

=> final and Immutability is different 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:

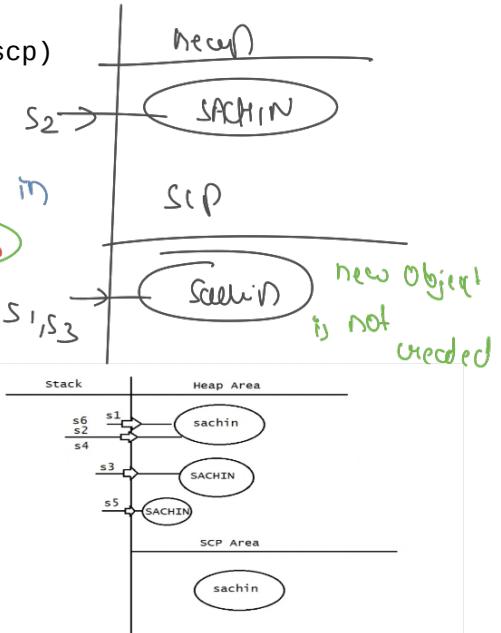
```
final variable(valid),
final object(invalid),
immutable variable(invalid)
immutable object(valid)
```

StringBuilder, StringBuffer are by default mutable.

All Wrapper classes(Byte, Short, Long, Integer, Float, Double, Boolean, Character) are by Default Immutable.

Mutable -> can be changed

Immutable => can't be changed same object, but new object we create & referent to same object reference.



XX

```
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); //true
System.out.print(s3==s5); //false
```

XX

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

```

not assign new
object.

SCP not define here only

Content of stringBuffer can be changes becoz it is Mutable

StringBuffer

1. If the content will change frequently then it is not recommended 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

Constructors of StringBuffer

1. `StringBuffer sb=new StringBuffer();`
creates a empty StringBuffer object with default initial 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("abcdefghijklmnp");
System.out.println(sb.capacity());//16
sb.append('q');
System.out.println(sb.capacity());//34`

2. `StringBuffer sb=new StringBuffer(initialCapacity);`
It creates an Empty String with the specified initial 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`

Important methods of StringBuffer/StringBuilder

- a. `public int length()`
- b. `public int capacity()`
- c. `public char charAt(int index)`
- d. `public void setCharAt(int index,char ch)`
- 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)`
- 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 f)`
- r. `public StringBuffer insert(int index,Object o)`

```
=====
public StringBuffer delete(int begin, int end)
public StringBuffer deleteCharAt(int index)
public StringBuffer reverse()
public void setLength(int Length)
public void trimToSize()
public void ensureCapacity(int capacity)
```

eg::

```
StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
System.out.println(sb.length()); //21
System.out.println(sb.capacity()); //21 + 16 = 37
System.out.println(sb.charAt(20)); //'r'
System.out.println(sb.charAt(100)); //StringIndexOutOfBoundsException
```

eg::

```
StringBuffer sb1 = new StringBuffer("kohlianushka");
sb1.setCharAt(5, 'A');
System.out.println(sb1); //kohliAnushka
```

7 Nov 2022

Anagram: String

- ① WAP to copy & str to another.
- ② WAP to convert string "Yash" to "yAsh", Uppercase to lower & lower to uppercase.
- ③ WAP to Reverse a String.
- ④ Reverse words of a string given?
- ⑤ Palindrome of a string? ("aba") $\xrightarrow{\text{reverse}}$ ("aba") } string ans = "";
- ⑥ Anagram of a string, "ate" \rightarrow "eat" ✓
- ⑦ Pangram of a string?

"Hi Yash How" / OIP
Hi say
words (string str)
word

String sarr[] = str.split(" ");
for (String elem : sarr)
{ for (int i = elem.length() - 1; i-- ;)
 ans += elem.charAt(i);
}

ans += " ";

return ans;

String s = "Yash";

String to chararr.

char[] ch = s.toCharArray();

6	18	6	0	25
0	1	2		

6	18	6	0	25
0	1	2		

e.g. "THE QUICK BROWN FOX JUMPS OVER LAZY DOG"
→ char[] arr = new char[26]; , , ,
// remove space

arr = str.replace(" ", "");
for (char ch : str.toCharArray())

 arr[ch - 65]++;
 if (arr contains any letter O) then
 not a pangram.

7th nov 2022

String, StringBuffer & StringTokenizer all are **final class**, so can't be inherited

StringBuffer

(synchronized)
(thread safe)

1. If the content will change frequently then it is not recommended to go for String object becoz for every new change a new Object will be created. - **cost of changing again & building new obj at m/o level is "costly".**
2. To handle this type of requirement, we have StringBuffer/StringBuilder concept

```
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)
```

append method is overloaded method, methodName is same but change in the argument type.

eg::

```
StringBuffer sb = new StringBuffer();
sb.append("PI value is :: ");
sb.append(3.1414);
sb.append(" This is exactly ");
sb.append(true);
System.out.println(sb); // PI value is ::3.1414 This is exactly true
```

Overloaded methods

```
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::

```
StringBuffer sb = new StringBuffer("sacin");
sb.insert(3, 'h');
System.out.println(sb); // sachin
sb.insert(6, "IND");
System.out.println(sb); // sachinIND
```

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::

```
StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
sb.delete(6, 12);
System.out.println(sb); // sachintendulkar
sb.deleteCharAt(13);
```

```

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

public StringBuffer reverse()
    It is used to reverse the given String.

eg:: StringBuffer sb=new StringBuffer("sachin");
    sb.reverse();
    System.out.println(sb); //nihcas

public void setLength(int Length)
    It is used to consider only the specified no of characters and remove all the
remaining characters.

eg:: StringBuffer sb=new StringBuffer("sachinramesh");
    sb.setLength(6);
    System.out.println(sb); //sachin

public void trimToSize()
    This method is used to deallocate the extra allocated free memory such that
capacity
    and size are equal.

eg:: StringBuffer sb = new StringBuffer(1000);
    System.out.println(sb.capacity()); //1000

    sb.append("sachin");
    System.out.println(sb.capacity()); //1000

    sb.trimToSize();

    System.out.println(sb); //sachin
    System.out.println(sb.capacity()); //6

public void ensureCapacity(int capacity)
    It is used to increase the capacity dynamically based on our requirement.

eg:: StringBuffer sb = new StringBuffer();
    System.out.println(sb.capacity()); //16
    sb.ensureCapacity(1000);
    System.out.println(sb.capacity()); //1000

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 required to wait so performance is high.
    Introduced in jdk1.5 version

```

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

Method Chaining

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);
```

eg#2.

```
class TestApp {
    public static void main(String[] args) {
        String name ="sachin";
        String data = name.toUpperCase();
        int count = data.length();
        System.out.println(count);

        //method chaining
        System.out.println(name.toUpperCase().length());

        StringBuffer sb = new StringBuffer("virat");

        //method chaining
        sb.append("kohli").
            insert(10, "anushka").
            reverse().
            append("IND").
            insert(sb.length(), "RCB").
            reverse().delete(0, 6);
        System.out.println(sb);

        StringBuffer sb1 =new StringBuffer("dhoni");
        sb1.length().append("CSK");//CE:int can't be dereferenced
    }
}
```

