String(java.lang)
==========
We classify String into 2 types
a. Immutable
b. Mutable(StringBuffer(1.0V),StringBuilder(1.5v))

Case1:
String s=new String("sachin"); // 2 object (SCP and other one in heapArea)
s.concat("tendulkar");// 1 object(SCP)
System.out.println(s);
Since String object are immutable, if we try to change the object, that change would not happen in the same memory
the changes will happen in the new memory this concept we call it as "Immutable".

VS

StringBuffer sb=new StringBuffer("sachin"); // 2 objects(SCP and Other one in heapArea)
sb.append("tendulkar"); // 1 object(SCP)
System.out.println(sb);
Since StringBuffer is mutable, if we try to change the object, that change would happen in the same memory, this
mechanism is called "mutable".

Case2:
String s1=new String("sachin");// 2 objects(SCP and Other one in heapArea)
String s2=new String("sachin");// 1 Object (heapArea)
System.out.println(s1.equals(s2)); // equals method is implemented to check the content of String

vs

StringBuffer sb1=new StringBuffer("sachin");
StringBuffer sb2=new StringBuffer("sachin");
System.out.println(sb1.equals(sb2); //equals method is not available in StringBuffer class,it is a part of Object class
so the implementation is coming from Object class which compares the reference(address)
not the data.

Code
====
```
StringBuffer sb1= new StringBuffer("sachin");
StringBuffer sb2= new StringBuffer("sachin");
System.out.println(sb1==sb2);//compares the refernce so => false
System.out.println(sb1.equals(sb2));//false
```

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

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

final keyword(access modifer)
==========

It is an access modifier which can be applied at 3 levels

a. variable(primitive and reference)
b. class level.
c. method level.

If the variable is made as final then the value for those variables can't be changed,if we try to change
it would result in "CompileTimeError".
final variables would be resolved at the compile time only by the compiler.

eg:

```
class Test{
      public static void main(String[] args){
                  final int a=10;
                  int b=20;
                  b++;
                  a++;// a= a+1; change for the variable:CE(compiletime
error)
                  System.out.println(a);
                  System.out.println(b);

      }
}
```

final vs Immutablity
==========

```
class Test{
      public static void main(String[] args){
                  final StringBuffer sb=new StringBuffer("sachin");
                        sb.append("IND");
                  System.out.println(sb);//sachinIND

                        sb=new StringBuffer("tendulkar");//CE
                  System.out.println(sb);
      }
}
```

If the variable is of primitive type and if it is final then the value of the variable should not be changed,if we try to

change it would result in "CompileTimeError".

If the variable if of reference type and if it is of mutable nature then as per its mutable nature the object data can be changed,it would not result in "CompileTimeError",but if we try to reassign the reference variable with a new object address then it would result in "CompileTimeError".

note:
  final variable(valid concept)
  final object(not valid)
  immutable variable(not valid)
  immutable object(valid)

primitive
=========
 int,float,byte,short,long,double,....

reference
=========
 String,StringBuffer,StringBuilder,Object,.......

StringBuffer
============
   It is avaialble in java.lang package.

Methods of StringBuffer
=======================
   capacity() => the default capacity of StringBuffer is 16
                        if the capacity is filled internally jvm will increase the size using the following formulae
                             newCapacity= (currentcapcity + 1) * 2

eg:
```
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());// (16+1) * 2

sb.append("rstuvwxyz");
System.out.println(sb.capacity());// 34
```

eg:
 StringBuffer sb = new StringBuffer(19);//here the integer no specifies the capacity of StringBuffer

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

eg:
StringBuffer sb = new StringBuffer("sachin");//here the capacity will be
(length of String + 16)
System.out.println(sb.capacity());
System.out.println(sb.length());

Important methods of StringBuffer
================================
  length() => it counts the no of characters present in the StringBuffer
Object
  append(String)
  append(Boolean) => it appends the given data to the Old StringBuffer
object
  append(float)

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

sb.append("The value of PIE IS :: ");
sb.append(3.1414);
sb.append(", This is  exactly ::");
sb.append(true);

System.out.println(sb);
System.out.println(sb.capacity());//70
System.out.println(sb.length());//54


  insert(int,String)
  insert(int,int)    => it inserts the String at the specified index
  insert(int,long)

eg:
StringBuffer sb = new StringBuffer("abcdefgh");
System.out.println(sb.capacity());// 8 + 16 => 24

sb.insert(2, "xyz");
System.out.println(sb);// abxyzcdefgh

sb.insert(11, 9);
System.out.println(sb);      // abxyzcdefgh9


    delete(int,int)
    deleteCharAt(int)

eg:
 StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
 System.out.println(sb.capacity());// 21+16 = 37
 System.out.println(sb.length());//21
```

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

  sb.delete(6,12);// start to end-1
  System.out.println(sb);//sachintendulkar

  sb.deleteCharAt(6);
  System.out.println(sb);//sachinendulkar

  sb.deleteCharAt(21);//SIOBE(StringIndexOutOfBoundsException)
```

reverse() => It is used to reverse the StringBuffer Object

eg:

```
  StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
  System.out.println(sb.capacity());// 21+16 = 37
  System.out.println(sb.length());//21

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

setLength(int) => it is possible to reduce the length of the String at the runtime

eg:

```
  StringBuffer sb = new StringBuffer("sachinrameshtendulkar");
       System.out.println(sb);//sachinrameshtendulkar
       System.out.println(sb.length());// 21

       sb.setLength(6);

       System.out.println(sb.length());//6
       System.out.println(sb);//sachin
```

trimToSize() => It will change the capacity to the length of the String

eg:

```
  StringBuffer sb = new StringBuffer(1000);
   System.out.println(sb.capacity());// 1000
            sb.append("iNeuron");
   System.out.println(sb.capacity());// 1000
          sb.trimToSize();
   System.out.println(sb.capacity());//7
```

ensureCapacity(int) => it is used to increase the capacity to the specific limit

eg:

```
 StringBuffer sb = new StringBuffer();
  System.out.println(sb.capacity());//16

  sb.ensureCapacity(10000);
```

```
  System.out.println(sb.capacity());//10000


Difference b/w StringBuilder and StringBuffer
=============================================
    StringBuffer  => 1.0 V(1996)
                              All the methods present are synchronized
                              At a time on StringBuffer object only one
thread can operate
                              Since only one thread operate it is
"ThreadSafety".
                              Performance is low.


    StringBuilder => 1.5 V
                              All the methods present are not
synchronized
                     At a time on StringBuilder object many threads can
operate
                              So it is not "ThreadSafety".

                              Performance is high.
```