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

  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 requried 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

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);
```

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


      }

}
```