StringBuffer Class

- StringBuffer class is used to create a mutable string object. It means, it can be changed after it is created. It represents growable and writable character sequence.
- It is similar to String class in Java both are used to create string, but stringbuffer object can be changed.
- So StringBuffer class is used when we have to make lot of modifications to our string. It is also thread safe i.e multiple threads cannot access it simultaneously. StringBuffer defines 4 constructors.
- **StringBuffer():** It creates an empty string buffer and reserves space for 16 characters.
- **StringBuffer(int size):** It creates an empty string and takes an integer argument to set capacity of the buffer.
- **StringBuffer(String str):** It creates a stringbuffer object from the specified string.
- **StringBuffer(charSequence []ch):** It creates a stringbuffer object from the charsequence array.

```
 class StrinBufferDemo1
{
        public static void main(String[] args)
            {
                StringBuffer sb = new StringBuffer("java");
                System.out.println(sb);
               sb.append("Programming");
                System.out.println(sb);                // Output: JavaProgramming
            }
}
```

Difference between String and StringBuffer

```
class StringVsStringBufferDemo2
  {
 public static void main(String args[])
 {
  String s = "Java";
  s.concat("Programming");
  System.out.println(s);      // Output: Java

  StringBuffer sb = new StringBuffer("Java");
  sb.append("Programming");
  System.out.println(sb);             // Output: JavaProgramming
 }
}
```

Methods of StringBuffer class

- ### append() method

This method will concatenate the string representation of any type of data to the end of the StringBuffer object. append() method has several overloaded forms.

StringBuffer append(String str)
StringBuffer append(int n)
StringBuffer append(Object obj)
The string representation of each parameter is appended to StringBuffer object.

```
class StringBufferDemo3 {
        public static void main(String[] args) {
                StringBuffer sb = new StringBuffer("JDK");
                sb.append(17);
                System.out.println(sb);           //O/P:  JDK17
        }
}
```

- ### insert() method

This method inserts one string into another. Here are few forms of insert() method.

StringBuffer insert(int index, String str)
StringBuffer insert(int index, int num)
StringBuffer insert(int index, Object obj)

Here the first parameter gives the index at which position the string will be inserted and string representation of second parameter is inserted into StringBuffer object.

```
 class StringBufferDemo4 {
	public static void main(String[] args) {
		StringBuffer sb= new StringBuffer("JAVA");
		sb.insert(2, 2020);
		System.out.println(sb);			//O/P: JA2020VA
	}
}
```

- **reverse() method**

This method reverses the characters within a StringBuffer object.

```
 class StrinBufferDemo5 {
	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer("JAVA");
		sb.reverse();
		System.out.println(sb);   //	O/P: AVAJ
	}
}
```

- **replace()**

This method replaces the string from the specified start index to the end index.

```
 class StringBufferDemo6 {
	public static void main(String[] args) {
		StringBuffer str = new StringBuffer("Hello World");
		str.replace( 6, 11, "java");
		System.out.println(str);
	}
}
```

- **capacity()**

This method returns the current capacity of StringBuffer object.

```
class StringBufferDemo7 {
	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer();
		System.out.println( sb.capacity() );		// o/p:   16
	}
}
```

- **ensureCapacity()**

This method is used to ensure minimum capacity of StringBuffer object.

If the argument of the ensureCapacity() method is less than the existing capacity, then there will be no change in existing capacity.

If the argument of the ensureCapacity() method is greater than the existing capacity, then there will be change in the current capacity using following rule:

newCapacity = (oldCapacity*2) + 2.

```
class StringBufferDemo8 {
	public static void main(String[] args) {
		StringBuffer str = new StringBuffer();
		System.out.println( str.capacity()); //output: 16 (since empty
constructor reserves space for 16 characters)
		str.ensureCapacity(19); //greater than the existing capacity
		System.out.println( str.capacity()); //output: 34 (by following the rule -
(oldcapacity*2) + 2.) i.e (16*2)+2 = 34.
	}
}
```

```
public class StringBufferEnsureCapacityDemo9 {
   public static void main(String[] args) {
        StringBuffer sb = new StringBuffer();
        //printing default capacity of string buffer
        System.out.println("default capacity of buffer: " + sb.capacity());
```

```
          StringBuffer sb1 = new StringBuffer("hello");
          System.out.println("string1: " + sb1);

          // returns the current capacity of string buffer 1
          System.out.println("capacity: " + sb1.capacity());
          // ensure capacity is less than old capacity
          sb1.ensureCapacity(20);
          System.out.println("new capacity: " + sb1.capacity());

          StringBuffer sb2 = new StringBuffer("programming");
          System.out.println("string2: " + sb2);

          // returns the current capacity of the string buffer 2
          System.out.println("old capacity: " + sb2.capacity());

          // returns twice oldcapacity*2+2
          sb2.ensureCapacity(28);
          System.out.println("new capacity: "+sb2.capacity());
    }
}
```

class **ConcatStringAndStringBufferDemo10**

```
{
   public static String concatWithString()    {
      String s = "Java";
      for (int i=0; i<10000; i++){
         s = s+ "Programming";
      }
      return s;
   }
   public static String concatWithStringBuffer(){
      StringBuffer sb = new StringBuffer("Java");
      for (int i=0; i<10000; i++){
         sb.append("Programming");
      }
      return sb.toString();
   }
   public static void main(String[] args){
      long startTime = System.currentTimeMillis();
      concatWithString();
```

```
        System.out.println("Time taken by Concating with String: "+(System.currentTimeMillis()-startTime)+"ms");
        startTime = System.currentTimeMillis();
        concatWithStringBuffer();
        System.out.println("Time taken by Concating with StringBuffer: "+(System.currentTimeMillis()-startTime)+"ms");
    }
}
```

Difference between String and StringBuffer

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory