Topic: Mutable strings in Java: The StringBuffer Class.
OOP concepts involved: Classes, Objects, Static Methods.
Programming generic concepts involved: Functions, Variables, Data Types, Arrays, Control Statements, Access Modifiers.

➢ Theoric introduction

MUTABLE STRINGS IN JAVA

Java *StringBuffer* and *StringBuilder* classes are used to create mutable (modifiable) strings. The difference between *StringBuffer* and *StringBuilder* classes compared to the well known String class in Java is that the latter is immutable, and it means that can't be changed.

StringBuffer and *StringBuilder* classes are used when there is a necessity to make a lot of modifications to Strings of characters. Unlike a *String*, objects of type *StringBuffer* and *StringBuilder* can be modified over and over again without leaving behind a lot of new unused objects.

The *StringBuilder* class was introduced as of Java 5 and the main difference between the *StringBuffer* and *StringBuilder* is that *StringBuilders* methods are not thread safe (not synchronized).

It is recommended to use StringBuilder whenever possible because it is faster than StringBuffer. However, if the thread safety is necessary, the best option is StringBuffer objects.

How does the StringBuffer class manage strings?

While the *String* class represents a string as an immutable character sequence with a fixed-length, the *StringBuffer* class represents a string as a growable and writable character sequence. *StringBuffer* may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

CONSTRUCTORS OF THE STRINGBUFFER CLASS

Constructor Header	Description
StringBuffer()	Creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	Creates a string buffer with the specified string.
StringBuffer(int capacity)	Creates an empty string buffer with the specified capacity as length.

IMPORTANT METHODS OF STRINGBUFFER CLASS

Access modifier and Type	Method	Description
public synchronized StringBuffer	*append(String s)* This method is overloaded using one only parameter but different data types as float, boolean, char, etc.	Append the specified string with this string.
public synchronized StringBuffer	*insert(int offset, String s)* This method is overloaded like insert(int, char), insert(int, boolean), insert(int,int),etc.	Insert the specified string with this string at the specified position.
public synchronized StringBuffer	*replace(int startIndex, int endIndex, String str)*	Replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	*delete(int startIndex, int endIndex)*	Delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	*reverse()*	Reverse a string.

public int	*capacity()*	Return de current capacity
public void	*ensureCapacity(int minCapacity)*	Is used to ensure the capacity at least equal to the given minimum.
public char	*charAt(int index)*	Is used to return the character at the specified position.
public int	*length()*	Is used to return the length of the string i.e. the total number of characters.
public String	- *substring(int beginIndex)* - *substring(int beginIndex, int endIndex)*	- Is used to return the substring from the specified beginIndex. - Is used to return the substring from the specified beginIndex and endIndex.

➢ Statement

Using the **StringBuffer** class, make a program that acts like a clock, that count seconds, minutes and hours. If the clock reaches de 24 hours of counting, the clock needs to stop.

➢ Program Code

LowerToUpper.java

```java
public class Clock {
	public static void main(String[] args) {
		StringBuffer clockTime = new StringBuffer(8); // reserving space for 8 characters HH:MM:SS
		int sec = 0, min = 0, hour = 0;

		clockTime.append("00:00:00");

		while (clockTime.substring(0, 8).compareTo("24:00:00") != 1) {
			try {
				System.out.println(clockTime);
```

```
                            Thread.sleep(1000);
                            sec++;
                            clockTime.replace(6, 8, (sec < 10) ? "0" +
String.valueOf(sec) : String.valueOf(sec));

                            if (sec == 60) {
                                  sec = 0;
                                  clockTime.replace(6, 8, (sec < 10) ? "0" +
String.valueOf(sec) : String.valueOf(sec));

                                  min++;
                                  clockTime.replace(3, 5, (min < 10) ? "0" +
String.valueOf(min) : String.valueOf(min));
                                  if (min == 60) {
                                        min = 0;
                                        clockTime.replace(3, 5, (min < 10) ? "0" +
String.valueOf(min) : String.valueOf(min));

                                        hour++;
                                        clockTime.replace(0, 2, (hour < 10) ? "0" +
String.valueOf(hour) : String.valueOf(hour));
                                        if (hour == 24)
                                              hour = 0;
                                  }

                            }

                      } catch (InterruptedException e) {
                            e.printStackTrace();
                      }
                }
          }
}
```

➢ Program execution

In this program, the clock is started with 0 seconds, 0 minutes and 0 hours. The clock works like a normal clock and will count until the output marks the 24 hours.

We use the static ***sleep()*** method from the **Thread** class for generating a delay of 1000 ms (1s), so each time the delay executes, we increment the value of our ***sec*** variable. When our ***sec*** variable reaches the number 60, the value of ***sec*** is set to 0 and the value of our variable ***min*** is incremented by 1. This keeps going on until our ***min*** variable reaches the number 60, at that point, our ***min*** variable is set to 0 and our ***hour*** variable increments by 1 unit.

This whole process keeps going on and on until our ***hour*** variable reaches the number 24, if that is the case, the program exits.

```
00:00:00
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
00:00:07
00:00:08
```

➢ Conclusions

It is convenient to use the classes StringBuffer and StringBuilder when we want to modify and add characters to the same object without needing to create another. This is because these classes are immutable, therefore, unlike the String class, which is immutable, it needs to create a new object every time a change occurs in the string.

- ➔ It is recommended to use String when the value of a string is non-changeable.
- ➔ It is recommended to use StringBuffer when the value of a string is changeable and application is implemented in a multithreaded environment.
- ➔ It is recommended to use StringBuilder when the value of a string is changeable but the application is not using multiple threads.