1.6 STRINGS

- A string is a sequence of characters.
- In Java, objects of the String class are immutable, which means they cannot be changed once created.

In Java, there are two primary methods for creating String objects:

1. Using String Literals:

This is the most common and straightforward way to create a string. A string literal is a sequence of characters enclosed in double quotes. When a string literal is encountered, the Java Virtual Machine (JVM) checks the String Constant Pool (a special area in the heap memory) to see if an identical string already exists.

- If a matching string is found, a reference to that existing string object is returned, promoting efficiency and memory optimization.
- If no matching string is found, a new String object is created in the String Constant Pool, and a reference to it is returned.

Example:

```
String greeting = "Hello World!";
String name = "Alice";
```

2. Using the new Keyword:

This method explicitly creates a new String object in the heap memory, even if an identical string literal exists in the String Constant Pool. This is done by invoking a constructor of the String class.

Example:

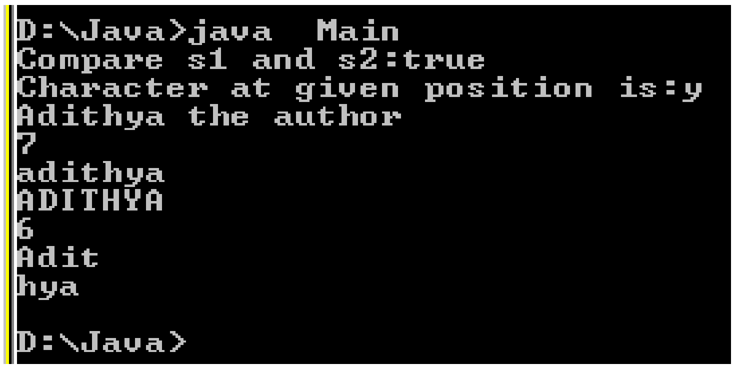
```
String message = new String("Welcome!");
char[] charArray = {'J', 'a', 'v', 'a'};
String fromCharArray = new String(charArray)
```

Methods in String Class:

- **charAt()**:Returns a character at a specified position.
- **equals()**:Compares the two given strings and returns a Boolean, that is, True or False.
- **concat()**: Appends one string to the end of another.
- **length()**: Returns the length of a specified string.
- **toLowerCase()**: Converts the string to lowercase letters.
- **toUpperCase()**: Converts the string to uppercase letters.
- **indexOf()**: Returns the first found position of a character.
- **substring()**: Extracts the substring based on index values, passed as an argument.

```
public class Main
{
```

```
public static void main(String []args)
{
        String s1="Adithya";
        String s2="Adithya";
        String s3="Adi";
        boolean x=s1.equals(s2);
        System.out.println("Compare s1 and s2:"+x);
        System.out.println("Character at given position is:"+ s1.charAt (5));
        System.out.println(s1.concat(" the author"));
        System.out.println(s1.length());
        System.out.println(s1.toLowerCase());
        System.out.println(s1.toUpperCase());
        System.out.println(s1.indexOf('a'));
        System.out.println(s1.substring(0,4));
        System.out.println(s1.substring(4));
    }
}
```



```
D:\Java>java  Main
Compare s1 and s2:true
Character at given position is:y
Adithya the author
7
adithya
ADITHYA
6
Adit
hya

D:\Java>
```

StringBuffer:

- StringBuffer class is used to create mutable (modifiable) string.
- Which means we can change the content of the StringBuffer without creating a new object every time.
- Its operations are synchronized in nature & to be used in multi-threading environment. (Thread Safe)
- Synchronization -> ensures that only one thread can access a critical section (like modifying data) at a time.

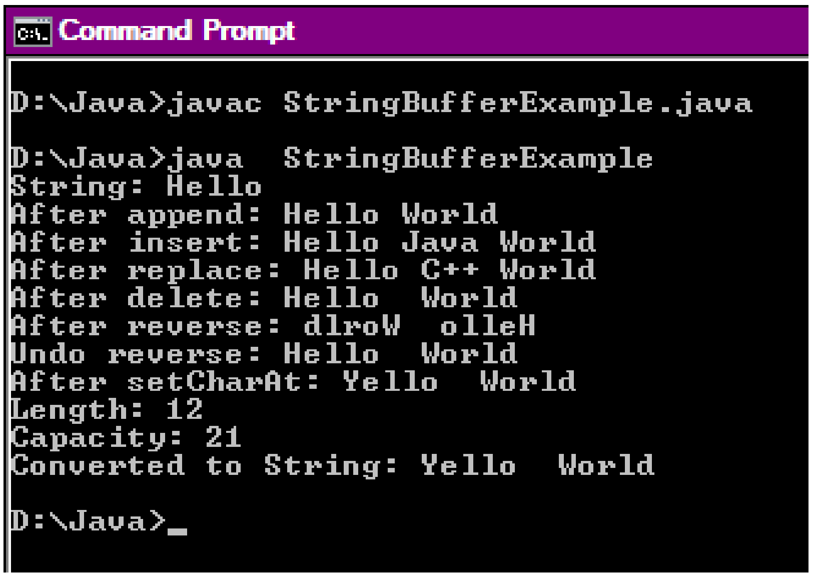
Methods in StringBuffer:

append(String str)	Appends the specified string to the end of the StringBuilder.
insert(int offset, String)	Inserts the specified string at the given position in the StringBuilder.
replace(int start, int end, String)	Replaces characters in a substring with the specified string.
delete(int start, int end)	Removes characters in the specified range
reverse()	Reverses the sequence of characters in the StringBuilder.
capacity()	Returns the current capacity of the StringBuilder.
length()	Returns the number of characters in the StringBuilder.
charAt(int index)	Returns the character at the specified index.
setCharAt(int index, char)	Replaces the character at the specified position with a new character.
substring(int start, int end)	Returns a new String that contains characters from the specified range.
ensureCapacity(int minimum)	Ensures the capacity of the StringBuilder is at least equal to the specified minimum.
deleteCharAt(int index)	Removes the character at the specified position.
indexOf(String str)	Returns the index of the first occurrence of the specified string.
lastIndexOf(String str)	Returns the index of the last occurrence of the specified string.
toString()	Converts the StringBuilder object to a String.

Example Program:

```
public class StringBufferExample
{
    public static void main(String[] args) {
    StringBuffer sb = new StringBuffer("Hello");
    System.out.println("String: " + sb);
    sb.append(" World");
    System.out.println("After append: " + sb);
    sb.insert(6, "Java ");
```

```
        System.out.println("After insert: " + sb);
        sb.replace(6, 10, "C++");
        System.out.println("After replace: " + sb);
        sb.delete(6, 9);
        System.out.println("After delete: " + sb);
        sb.reverse();
        System.out.println("After reverse: " + sb);
        sb.reverse();  // undo reverse
        System.out.println("Undo reverse: " + sb);
        sb.setCharAt(0, 'Y');
        System.out.println("After setCharAt: " + sb);
        System.out.println("Length: " + sb.length());
        System.out.println("Capacity: " + sb.capacity());
        String result = sb.toString();
        System.out.println("Converted to String: " + result);
    }
}
```



```
D:\Java>javac StringBufferExample.java

D:\Java>java  StringBufferExample
String: Hello
After append: Hello World
After insert: Hello Java World
After replace: Hello C++ World
After delete: Hello  World
After reverse: dlroW  olleH
Undo reverse: Hello  World
After setCharAt: Yello  World
Length: 12
Capacity: 21
Converted to String: Yello  World

D:\Java>_
```

StringBuilder:

- StringBuilder class is also used to create mutable (modifiable) string.
- Not thread-safe (not synchronized) ⇒ faster than StringBuffer in single-threaded environments.
- Ideal for string manipulation where thread safety is not a concern.

Features	String	StringBuilder	StringBuffer
Mutability	String are immutable (creates new objects on modification)	StringBuilder are mutable(modifies in place)	StringBuffer are mutable(modifies in place)
Thread-Safe	It is thread-safe	It is not thread-safe	It is thread-safe
Performance	It is slow because it creates an object each time	It is faster (no object creation)	it is slower due to synchronization overhead
Use Case	Fixed, unchanging strings	Single-threaded string manipulation	Multi-threaded string manipulation