**Strings in JAVA**

A String acts the same as an array of characters in JAVA

In JAVA, every character is stored in 16 bits encoding.

Strings are the types of objects which can store characters as elements.

Ways of creating String

1. String Literal
2. Using New Keyword.
3. String Literal

String str = “Vrushali”;

1. Using New Keyword.

String s = new String(“Welcome”);

* It is sequence of characters.
* Objects of Strings are immutable which means a constant and cannot be changed once created.

Interfaces and Classes in Strings in Java

CharBuffer: This Class Implements Charsequence Interface.

CharSequence Interface is used for representing the sequence of Characters in Java.

Classes that are implemented using the CharSequence interface are provides much of functionality like substring, lastoccurence, first occurence, concatenate, toupper, tolower etc.

Strings are not mutable. Immutable means unmodifiable.

Example

class HelloWorld {

public static void main(String[] args) {

String s ="Vrushali";

s.concat(" Haldankar");

System.out.println(s);

}

OUTPUT

Vrushali

Once the String Object is created, we cannot change its data or state, but new String object is created.

Here Vrushali is not changed, but new object “Vrushali Haldankar” is created. That’s why string is known as immutable.

As we can see two objects are created but reference variable still refers to “Vrushali” and not to “Vrushali Haldankar “

But if we explicitly assign to the reference variable, it will refer to “Vrushali Haldankar” object.

class HelloWorld {

public static void main(String[] args) {

String s ="Vrushali";

s= s.concat(" Haldankar");

System.out.println(s);

}

}

OUTPUT

Vrushali Haldankar

Memory Allotment of String

* Whenever a String Object is created as a literal, the object will be created in the String constant pool.

String demoString = "Geeks";

* In case of String are dynamically allocated they are assigned a new memory location in the heap.

String Methods in java

1. length()

String s ="Vrushali";

System.out.println("String Vrushali Length: "+ s.length());

OUTPUT

String Vrushali Length: 8

1. charAt(int i)

System.out.println("Vrushali at 4th location character: "+ s.charAt(4));

OUTPUT

Vrushali at 4th location character: h

1. substring(int 3)

System.out.println("return Vrushali string from 5th location: "+ s.substring(5));

OUTPUT

return Vrushali string from 5th location: ali

1. String substring (int i, int j)

System.out.println("Vrushali Returns the substring from i to j-1 index: "+

s.substring(2,6));

OUTPUT

Vrushali Returns the substring from i to j-1 index: usha

1. String concat( String str)

System.out.println("Concatenates specified string to the end of this string.: "+ s1.concat(s2));

OUTPUT

Concatenates specified string to the end of this string.: VrushaliHaldankar

1. int indexOf (String s)

String v1 = "String are class";

System.out.println

("Returns the index within the string of the first occurrence of the specified

string.: "+ v1.indexOf ("are"));

OUTPUT : 7

1. Int lastIndexOf( String s)

System.out.println(("Returns the index within the string of the first

occurrence of the specified string.: "

+ v1.lastIndexOf ("a"));

OUTPUT : 13

1. Boolean equals(obj)

System.out.println("Returns the boolean if string matches : "

+ s.equals("Vrushali"));

OUTPUT :true

1. Boolean equalsIgnoreCase(String)

System.out.println("Returns the boolean if string matches by ignoring lowercase and uppercase : "+ s.equalsIgnoreCase("VRUSHALI"));

OUTPUT true

1. String toLowerCase()

D1 = uppercase();

System.out.println ("Returns lowercase : " + d1.toLowerCase());

OUTPUT : UPPERCASE

1. String toUpperCase()

System.out.println ("Returns Uppercase : " + s1.toUpperCase());

OUTPUT : VRUSHALI

1. String trim()

String tr = “ remove space “

System.out.println ("Trim removes unwanted spaces : " + tr.trim());

OUtPUT : remove space

1. Boolean contains(String)

Returns true if string contains contains the given string.

println ("Returns true if string contains contains the given string : "

+ s.contains (s1));

OUTPUT :true

StringBuffer Class

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.

* The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
* The append() method is used to add characters, strings, or other objects to the end of the buffer.
* The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
* The delete() method is used to remove characters from the buffer.
* The reverse() method is used to reverse the order of the characters in the buffer.

StringBuffer br = new StringBuffer("Hello");

System.out.println(br.append(" world !")); // output : Hello world !

System.out.println(br.insert(2," world"));// output

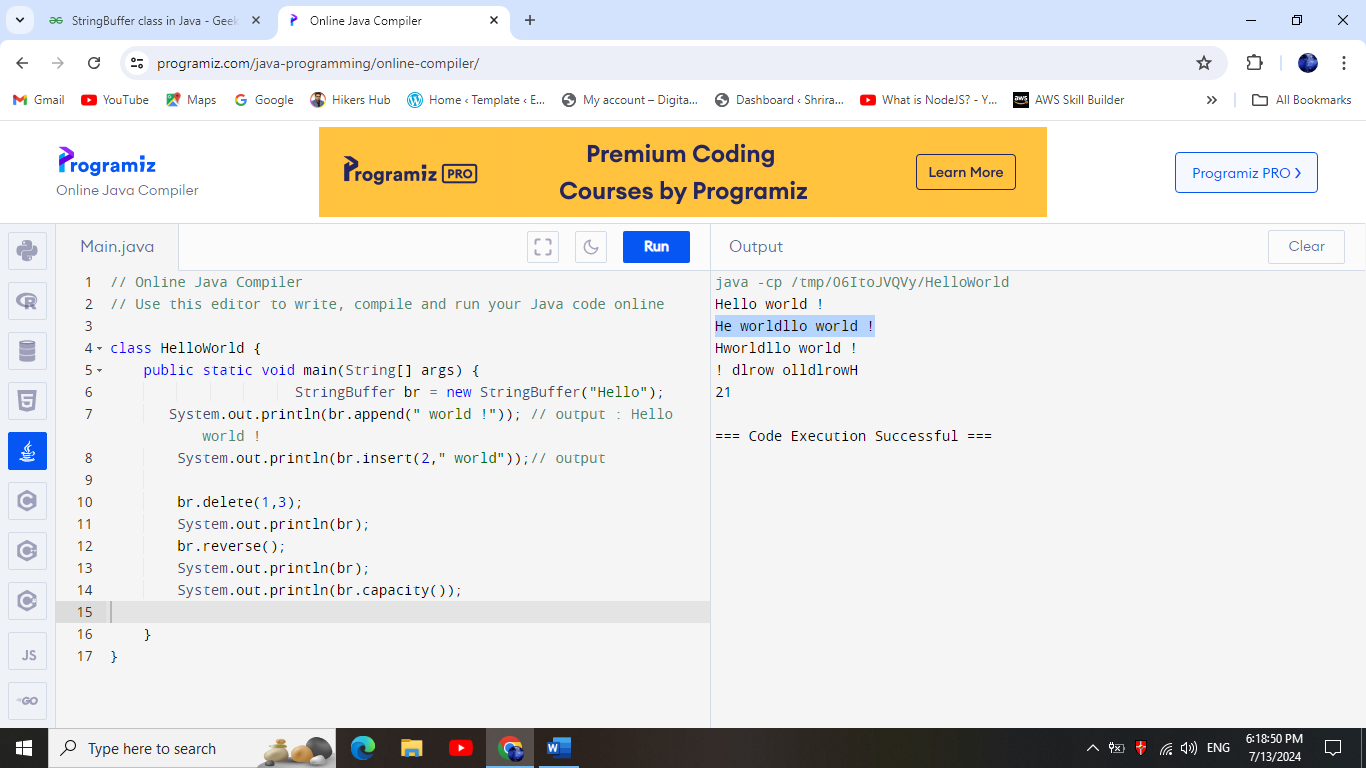
br.delete(1,3);

System.out.println(br);

br.reverse();

System.out.println(br);

System.out.println(br.capacity());



String Builder class

StringBuilder in Java represents a mutable sequence of characters.

The function of StringBuilder is very much similar to the StringBuffer class, as both of them provide an alternative to String Class by making a mutable sequence of characters.

* However, the StringBuilder class differs from the StringBuffer class on the basis of synchronization.
* The StringBuilder class provides no guarantee of synchronization whereas the StringBuffer class does.
* Instances of StringBuilder are not safe for use by multiple threads.
* String Builder is not thread-safe and high in performance compared to String buffer.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | String | StringBuffer | StringBuilder |
| Mutability | Immutable | Mutable | Mutable |
| Thread Safety | Thread Safe | Thread Safe | Not Thread Safe |
| Memory efficiency | High | Low (multi threading) | High |
| Usage | This is used when we want Immutability | This is used whren Thread Safety required | This is used when Thread safety is NOT required |