Java Strings

① Used to storing texts.

```
String a = "Hello";
```

↳ String Length :-

① A string in Java is actually an object, which contains methods that can perform certain operations on strings.

length() → Length of string

```
String b = "ABCD";
System.out.println("The length of b string
                    is: " + b.length());
```

O/P → 4

```
System.out.println( b.toUpperCase());
S.O.P.( b.toLowerCase());
```

↳ Finding a character in a String :-

indexOf() → Returns the index (the position) of the first occurrence of a specified text in a string (including white space).

⊙ Java counts positions from zero.

e.g. String b = "Please indicate wherein!";
S.o.p.(b.indexOf("indicate"));

o/p → 7

↳ String Concatenation

'+' → Used b/w strings to combine them.

e.g. String a = "Palak";
String b = "sharma";
S.o.p.(a+" "+b);

o/p → Palak sharma

S.o.p.(a.concat(b));

o/p → Palak sharma

⊙ concat() → Used to concatenate two strings.

↳ Adding numbers and strings

- In java '+' → used for both addition and concatenation.
- Numbers are added while strings are concatenated.

e.g.
```
int x = 1;
int y = 2;
int z = x+y; // z = 3 (Numbers)
```

e.g.
```
String x = "10";
String y = "20";
String z = x+y; // z = 1020 (String)
```

e.g.
```
String x = "5";
int y = 20;
String z = x+y; // z = 520 (String)
```

DETAILED LECTURE NOTES ①

String in Java

- Generally, string is a sequence of characters.
- But in Java, string is an object that represents a sequence of characters.

e.g. java.lang.string ⇒ Used to create objects.

- An array of character works same as Java string.

e.g.
```
char[] ch = {'j', 'a', 'v', 'a'};
String s = new String(ch);
```
same as:
```
String s = "java";
```

↳ How to create a string object?

i) By string literal
ii) By new keyword

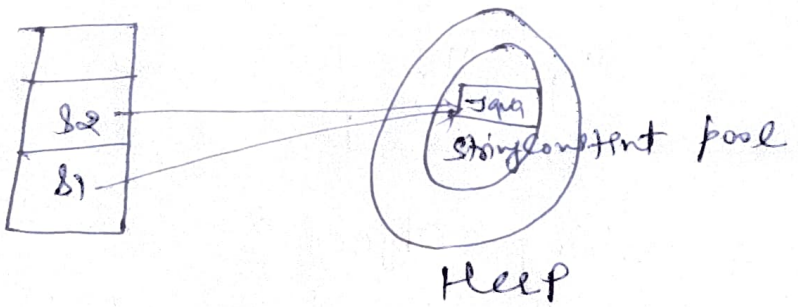
i) String Literal :-

- Java string literal is created by using double quotes.

  e.g. String s = "welcome";

- String objects are stored in a special memory area known as "String Constant pool".
- Whenever any string literal created, JVM checks the "String constant pool" first. If the string already exists in the pool, a reference to the pooled instance is returned.
- If the string doesn't exist in the pool, a new string instance is created and placed in the pool.

e.g. String s1 = "Java";
String s2 = "Java"; // It doesn't create a new instance.



Note:- To make Java more memory efficient (because no new objects are created if exists already in the String constant pool).

2> By new Keyword :-

String s = new String ("Java");

① JVM will create a new String object in normal (non-pool) heap memory, and the literal "Java" will be placed in the string constant pool.

② The variable 's' will refer to the object in a heap(non-pool).

e.g.
```
public class StringExample {
public static void main (String args[]) {
String s1 = "java";
Char ch = {'s', 't', 'r', 'i', 'n', 'g', 's'};
String s2 = new String (ch); // converting
Char array into string
String s3 = new String ("example");
// Creating java string by 'new' keyword
```

```
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

O/P → java
strings
example

↳ Java String Class Methods :-

⊙ The java.lang.String class provides many useful methods to perform operations on sequence of char values.

(1) char charAt(int index) :- It returns char value for the particular index.

(2) int length(); It returns string length.

(3) static String format(String format, Object... args) :- It returns a formatted string.

(4) boolean equals(Object another) :- It checks equality of the string with the given object.

⑤ boolean isEmpty():- It checks if string is empty.

⑥ String Concat(String str):- It concatenes the specified string.

⑦ String toLowerCase():- It returns a string in lower case.

⑧ String toUpperCase():- It returns a string in upper case.

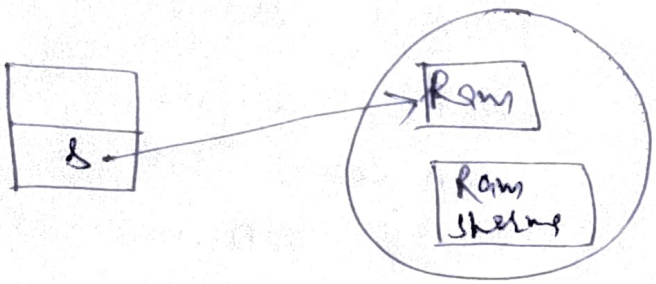
↳ Immutable String in Java:-

⊙ In Java, string objects are immutable. Immutable means unmodifiable or unchangable.

⊙ Once String object is created its data or state can't be changed but a new string object is created.

```
Class TestImmutableString {
pubeic static void main(String args[]) {
  String s = "Ram";
   s.Concat ("sheeru");
    System.out.println(s);
  }
}
```

O/p → Ram



Heap

⊙ Ram is not changed but a new object with Ram sheeru is created; that's why String is Kn as immutable.

↳ If we write

```
String s="Ram";
s= s.concat("sheeru");
System.out.println(s);
```

<u>O/p</u> → Ram sheeru

↳ Following features of String which makes String objects Immutable:-

① Class Loader:- A class loader in Java uses a String object as an argument.
Let if String object → modifiable
then value might be changed & the class that is supposed to be loaded might be different.
⊙ To avoid this kind of misinterpretation, String is immutable.

② Thread Safe:-
⊙ As String object is immutable, we don't have to take care of synchronization i.e. required while sharing an object across multiple threads.

Java String Compare

Three ways to compare String in Java:-

1. By using equals() method
2. By using == operator
3. By compareTo() method

1. **By using equals() method**:- It compares values of string for equality.

String class provides two methods:-

a) **public boolean equals(Object another)**:- Compares string to the specified object.

b) **public boolean equalsIgnoreCase(String another)**:- Compares this string to another string, ignoring case.

e.g.

```
Class Test {
public static void main(String args[]) {
String s1 = "sachin";
String s2 = "Sachin";
```

```
String s3 = new String ("sachin");
String s4 = "saneev";
System.out.println (s1.equals (s2)); // True
System.out.println (s1.equals(s3)); // T
System.out.println (s1.equals (s4)); // f
}
}
```

o/p → true
true
false

eg. (2)

```
Class Test {
public static void main (String args[]){
String s1 = "sachin";
String s2 = "SACHIN";
System.out.println (s1.equals(s2)); // false
System.out.println (s1.equalsIgnoreCase(s2)); // T
}
}
```

o/p → false
true

② By using == operator :- Compares references not values.

```
Class Test {
public static void main (String args[]){
  String s1 = "sachin";
  String s2 = "sachin";
  String s3 = new String ("sachin");
  System.out.println (s1==s2); // T → both refere same instance
  System.out.println (s1==s3); // F → s3 refers to instance created in non pool
}
}
```

③ CompareTo() method :- Demonstrates the use of == operator used for comparing two String objects.

⊙ compares value lexicographically & returns an integer value.

↳ s1 & s2 → objects of two String.

s1 = s2 ⇒ returns 0

s1 > s2 ⇒ (+)ve value

s1 < s2 ⇒ (−)ve value

e.g.
```
Class Test {
public static void main (String args[]) {
    String s1 = "sachin";
    String s2 = "sachin";
    String s3 = "Ratin";
    S.O.P (s1.CompareTo (s2)); //0
    S.O.P (s1.CompareTo (s3)); //1  (s1 > s3)
    S.O.P (s3.CompareTo (s1)); //-1 (s3 < s1)
}
}
```

O/P →
```
0
1
-1
```

↳ String Concatenation in Java

① By + (String concatenation operator)

② By concat() method

↳ + → To add strings

```
Class Test {
public static void main (String args[]) {
String s = "Ram" + "Kumar";
System.out.println(s);
}
}
```

O/P → Ram Kumar

⊙ After a string literal, all the + will be treated as string concatenation operator.

↳ By concat() method → Concatenates the specified string to the end of the current string.

public String concat (String another);

↳
```
Class Test {
  public static void main (String args[]) {
    String s1 = "Ram";
    String s2 = "Kumar";
    String s3 = s1.concat (s2);
    System.out.println (s3);
  }
}
```

O/P → Ram Kumar.

String Buffer Class

① It represents a mutable sequence of characters.

② StringBuffer objects are mutable, meaning that we can change the contents of the buffer without creating a new object.

↳ To concatenate strings:

```
public class StringBufferExample {
public static void main (String args []) {
StringBuffer sb = new StringBuffer ();
 sb. append ("Hello");
 sb. append (" ");
 sb. append ("world!");
  String message = sb. to String ();
  System. out. println (message);
 }
}
```

o/p → Hello world!

append() → Concatenate the given argument with this string.

↳ StringBuffer insert() :-

insert() → Inserts the given String with this String at given position.

e.g. StringBuffer sb = new StringBuffer("Hello");
sb.insert(1, "Java");
S.O.P(sb);

O/P → HJavaello

↳ StringBuffer replace() :-

replace() → Replaces the given String from the specified beginIndex and endIndex.

e.g. StringBuffer sb = new StringBuffer("Hello");
sb.replace(1, 3, "Java");
S.O.P.(sb);

O/P → HJavalo

↳ delete() → delete the String from specified beginIndex to endIndex.

eg. sb.delete(1,3);
S.O.P.(sb);
O/P → Hlo

↳ reverse() → Reverse current String.

e.g. sb.reverse();
S.O.P.(sb);
O/P → olleH