## String Handling

1. Strings, which are widely used in Java programming, are a sequence of characters.
2. In Java programming language, strings are treated as objects.
3. for e.g. “Hello” is a string of 5 characters.
4. In java, string is an immutable object which means it is constant and cannot be changed once it has been created.
5. Strings in Java are Objects that are backed internally by a char array.
6. Since arrays are immutable (cannot grow), Strings are immutable as well. Whenever a change to a String is made, an entirely new String is created.

There are two ways to create String object:

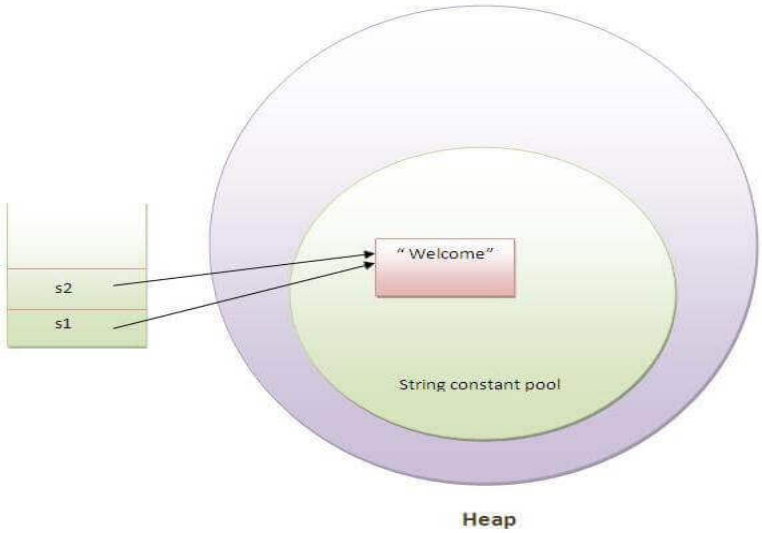
1. By string literal
2. By new keyword

Java **String literal** is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the 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. For example:

1. String s1="Welcome";
2. String s2="Welcome";//It doesn't create a new instance



In the above example, only one object will be created. Firstly, JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create a new object but will return the reference to the same instance. This makes Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

**By new keyword**

String s=**new** String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal (non-pool) heap memory, and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in a heap (non-pool).

**Example :**

```
public class Example{
  public static void main(String args[]){

        //creating a string by java string literal
        String str = " HelloAll";
        char arrch[]={'h','e','l','l','o'};

        //converting char array arrch[] to string str2
        String str2 = new String(arrch);

        //creating another java string str3 by using new keyword
        String str3 = new String("Java String Example");

        //Displaying all the three strings
        System.out.println(str);
        System.out.println(str2);
        System.out.println(str3);
  }
}
```

## Concatenating String

There are 2 methods to concatenate two or more string.

1. Using **concat()** method
2. Using + operator

### 1) Using concat() method

Concat() method is used to add two or more string into a single string object. It is string class method and returns a string object.

```
public class Demo{

   public static void main(String[] args) {
            String s = "Hello";
            String str = "Java";
            String str1 = s.concat(str);
            System.out.println(str1);
   }
}
```

### 2) Using + operator

Java uses "+" operator to concatenate two string objects into single one. It can also concatenate numeric value with string object. See the below example.

```
public class Demo{
   public static void main(String[] args) {
            String s = "Hello";
            String str = "Java";
            String str1 = s+str;
            String str2 = "Java"+11;
            System.out.println(str1);
            System.out.println(str2);
   }
}
```

**String Length**

One method that you can use with strings is the length() method, which returns the number of characters contained in the string object.

```
public class StringDemo {

  public static void main(String args[]) {
    String try1 = "Hello all";
    int len = try1.length();
    System.out.println( "String Length is : " + len );
  }
}
```

**String Comparison**

To compare string objects, Java provides methods and operators both. So we can compare string in following three ways.

1. Using equals() method
2. Using == operator
3. By CompareTo() method

**Using equals() method :** equals() method compares two strings for equality. Its general syntax is, boolean equals (Object str)

**Example**

It compares the content of the strings. It will return **true** if string matches, else returns **false**.

```
public class Demo{
   public static void main(String[] args) {
            String s = "Hell";
            String s1 = "Hello";
            String s2 = "Hello";
            boolean b = s1.equals(s2);    //true
            System.out.println(b);
            b =         s.equals(s1) ;   //false
            System.out.println(b);
   }
}
```

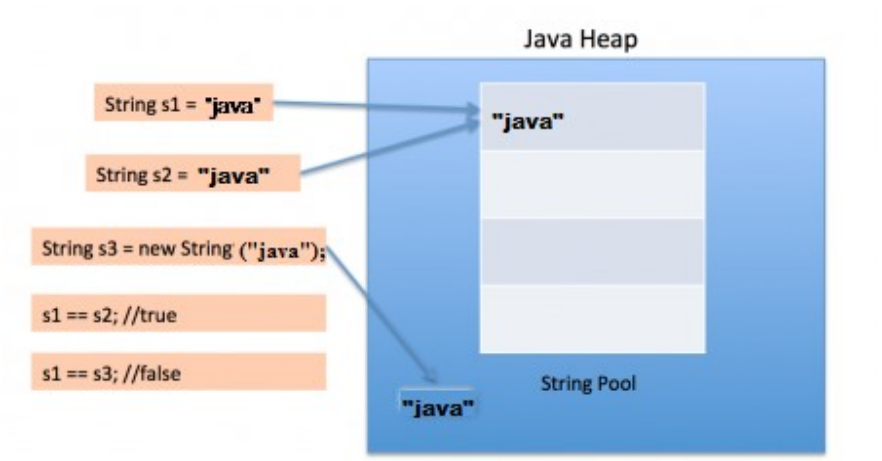
**Using == operator**

The double equal (==) operator compares two object references to check whether they refer to same instance. This also, will return **true** on successful match else returns false.

```
public class Demo{
   public static void main(String[] args) {
            String s1 = "Java";
            String s2 = "Java";
            String s3 = new String ("Java");
            boolean b = (s1 == s2);     //true
            System.out.println(b);
            b =         (s1 == s3);     //false
            System.out.println(b);
   }
}
```

**Explanation**

We are creating a new object using new operator, and thus it gets created in a non-pool memory area of the heap. s1 is pointing to the String in string pool while s3 is pointing to the String in heap and hence, when we compare s1 and s3, the answer is false.




**By compareTo() method**

String compareTo() method compares values and returns an integer value which tells if the string compared is less than, equal to or greater than the other string. It compares the String based on natural ordering i.e alphabetically. Its general syntax is.

Syntax:  int compareTo(String str)

**Example:**

```
public class HelloWorld{
   public static void main(String[] args) {
            String s1 = "Abhi";
            String s2 = "Viraaj";
            String s3 = "Abhi";
            int a = s1.compareTo(s2);     //return -21 because s1 < s2
            System.out.println(a);
            a = s1.compareTo(s3);     //return 0 because s1 == s3
            System.out.println(a);
            a = s2.compareTo(s1);     //return 21 because s2 > s1
            System.out.println(a);
   }
}
```

**Java String Methods**

1. char charAt(int index): It returns the character at the specified index. Specified index value should be between 0 to length() -1 both inclusive. It throws IndexOutOfBoundsException if index<0||>= length of String.

```
public class Demo {
   public static void main(String[] args) {
      String str = "Javaclass";
      System.out.println(str.charAt(2));
   }
}
```

2. boolean equals(Object obj): Compares the string with the specified string and returns true if both matches else false.
3. boolean equalsIgnoreCase(String string): It works same as equals method but it doesn't consider the case while comparing strings. It does a case insensitive comparison.

```
public class Demo {
   public static void main(String[] args) {
      String str = "java";
      System.out.println(str.equalsIgnoreCase("JAVA"));
   }
}
```

4. int compareTo(String string): This method compares the two strings based on the Unicode value of each character in the strings.

Return Value : The value 0 if the argument is a string lexicographically equal to this string; a value less than 0 if the argument is a string lexicographically greater than this string; and a value greater than 0 if the argument is a string lexicographically less than this string.

```
public class Test {

  public static void main(String args[]) {
    String str1 = "Strings are immutable";
    String str2 = new String("Strings are immutable");
    String str3 = new String("Integers are not immutable");

    int result = str1.compareTo( str2 );
    System.out.println(result);

    result = str2.compareTo( str3 );
    System.out.println(result);
  }
}
```

Output

```
0
10
```

5. int compareToIgnoreCase(String string): Same as CompareTo method however it ignores the case during comparison.
6. boolean startsWith(String prefix, int offset): It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.
7. boolean startsWith(String prefix): It tests whether the string is having specified prefix, if yes then it returns true else false.
8. boolean endsWith(String suffix): Checks whether the string ends with the specified suffix.
9. int indexOf(int ch): Returns the index of first occurrence of the specified character ch in the string.

```
import java.io.*;
public class Test {

  public static void main(String args[]) {
    String Str = new String("Welcome to class");
    System.out.print("Found Index :" );
    System.out.println(Str.indexOf( 'o' ));     } }
```

10. int indexOf(int ch, int fromIndex): Same as indexOf method however it starts searching in the string from the specified fromIndex.

```
import java.io.*;
public class Test {

   public static void main(String args[]) {
      String Str = new String("Welcome to the world of java ");
      System.out.print("Found Index :" );
      System.out.println(Str.indexOf( 'o', 5 ));
   }
}
```

11. int lastIndexOf(int ch): It returns the last occurrence of the character ch in the string.

```
import java.io.*;
public class Test {

   public static void main(String args[]) {
      String Str = new String("Welcome to the world of java");
      System.out.print("Found Last Index :" );
      System.out.println(Str.lastIndexOf( 'o' ));
      System.out.print("Found Last Index :" );
      System.out.println(Str.lastIndexOf( 'o', 5 ));
      String SubStr1 = new String("world" );
      System.out.print("Found Last Index :" );
      System.out.println( Str.lastIndexOf( SubStr1 ));


   }
}
```

12. int lastIndexOf(int ch, int fromIndex): Same as lastIndexOf(int ch) method, it starts search from fromIndex.
13. int indexOf(String str): This method returns the index of first occurrence of specified substring str.
14. int lastindexOf(String str): Returns the index of last occurrence of string str.
15. String substring(int beginIndex): It returns the substring of the string. The substring starts with the character at the specified index.
16. String substring(int beginIndex, int endIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.

Example:

```
public class Demo {
   public static void main(String[] args) {
```

```
        String str = "0123456789";
        System.out.println(str.substring(4));
        System.out.println(str.substring(4,7));
    }
}
```

17. String concat(String str): Concatenates the specified string "str" at the end of the string.
18. String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.
19. boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException of 's' is null.
20. String toUpperCase():This method returns string with all lowercase character changed to uppercase.
21. public String intern(): This method searches the specified string in the memory pool and if it is found then it returns the reference of it, else it allocates the memory space to the specified string and assign the reference to it.
22. public boolean isEmpty(): This method returns true if the given string has 0 length. If the length of the specified Java String is non-zero then it returns false.
23. public static String join(): This method joins the given strings using the specified delimiter and returns the concatenated Java String

```
public class Example{
  public static void main(String args[]){
            //The first argument to this method is the delimiter
            String str=String.join("^","You","are","Awesome");
            System.out.println(str);
  }
}
```

Output: You^are^Awesome

24. String replaceFirst(String regex, String replacement): It replaces the first occurrence of substring that fits the given regular expression "regex" with the specified replacement string.
25. String replaceAll(String regex, String replacement): It replaces all the occurrences of substrings that fits the regular expression regex with the replacement string.

26. String[] split(String regex, int limit): It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.
27. String[] split(String regex): Same as split(String regex, int limit) method however it does not have any threshold limit.
28. String toLowerCase(Locale locale): It converts the string to lower case string using the rules defined by given locale.
29. public static String format(): This method returns a formatted java String
30. String toLowerCase(): Equivalent to toLowerCase(Locale. getDefault()).
31. String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.

In the following example we have a string with leading and trailing white spaces, we are using trim() method to get rid of these leading and trailing white spaces but we want to retain the white spaces that are in between the words of the given string str. The trim() method only removes the leading and trailing white spaces and leaves the in between spaces as it is.

```
public class JavaExample{
  public static void main(String args[]){
        String str = new String("    How are you??   ");
        System.out.println("String before trim: "+str);
        System.out.println("String after trim: "+str.trim());
  }
}
```

32. char[] toCharArray(): Converts the string to a character array.
33. static String copyValueOf(char[] data): It returns a string that contains the characters of the specified character array.
34. static String copyValueOf(char[] data, int offset, int count): Same as above method with two extra arguments – initial offset of subarray and length of subarray.
35. void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin): It copies the characters of **src** array to the **dest** array. Only the specified range is being copied(srcBegin to srcEnd) to the dest subarray(starting fromdestBegin).
36. static String valueOf(): This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
37. boolean contentEquals(StringBuffer sb): It compares the string to the specified string buffer.
38. boolean regionMatches(int srcoffset, String dest, int destoffset, int len): It compares the substring of input to the substring of specified string.
39. boolean regionMatches(boolean ignoreCase, int srcoffset, String dest, int destoffset, int len): Another variation of regionMatches method with the extra boolean argument to specify whether the comparison is case sensitive or case insensitive.

**Parameters description:**
ignoreCase– if true, ignore case when comparing characters.
srcoffset – the starting offset of the subregion in this string.
other – the string argument.
destoffset – the starting offset of the subregion in the string argument.
len – the number of characters to compare.

Example:

```
public class RegionMatchesExample{
  public static void main(String args[]){
      String str1 = new String("Hello, How are you");
      String str2 = new String("How");
      String str3 = new String("HOW");

      System.out.print("Result of Test1: " );
      System.out.println(str1.regionMatches(7, str2, 0, 3));

      System.out.print("Result of Test2: " );
      System.out.println(str1.regionMatches(7, str3, 0, 3));

      System.out.print("Result of Test3: " );
      System.out.println(str1.regionMatches(true, 7, str3, 0, 3));
  }
}
```

Output:

```
Result of Test1: true
Result of Test2: false
Result of Test3: true
```

40. int length(): It returns the length of a String.

Example:

```
public class StudyTonight {
   public static void main(String[] args) {
       String str="StudyTonight";
      System.out.println(str.indexOf('u')); // int indexOf(int ch)
      System.out.println(str.indexOf('t', 3));    // int indexOf(int ch, int fromIndex)
```

```
        String subString="Ton";
        System.out.println(str.indexOf(subString)); // int indexOf(String str)
        System.out.println(str.indexOf(subString,7));   // int indexOf(String str, int fromIndex)
    }
}
```

Output

```
2
11
5
-1
```

**Java String split Example**

```
public class SplitExample{
  public static void main(String args[]){
            // This is out input String
            String str = new String("28/12/2013");

            System.out.println("split(String regex):");
            /* Here we are using first variation of java string split method
             * which splits the string into substring based on the regular
             * expression, there is no limit on the substrings
             */
            String array1[]= str.split("/");
            for (String temp: array1){
                System.out.println(temp);
            }

            /* Using second variation of split method here. Since the limit is passed
             * as 2. This method would only produce two substrings.
             */
            System.out.println("split(String regex, int limit) with limit=2:");
            String array2[]= str.split("/", 2);
            for (String temp: array2){
                System.out.println(temp);
            }

            System.out.println("split(String regex, int limit) with limit=0:");
            String array3[]= str.split("/", 0);
            for (String temp: array3){
                System.out.println(temp);
            }
```

```
            /* When we pass limit as negative. The split method works same as the first variation
             * because negative limit says that the method returns substrings with no limit.
             */
            System.out.println("split(String regex, int limit) with limit=-5:");
            String array4[]= str.split("/", -5);
            for (String temp: array4){
                System.out.println(temp);
            }
    }
}
```

**Output:**

```
split(String regex):
28
12
2013
split(String regex, int limit) with limit=2:
28
12/2013
split(String regex, int limit) with limit=0:
28
12
2013
split(String regex, int limit) with limit=-5:
28
12
2013
```