**Java String**

- String is immutable, which means we cannot modify once it is created.
- String is class in Java.
- String represents sequence of characters.

**Example1:**

```
package stringInJava;

public class Example1 {

	public static void main(String[] args) {
		char[] ch = { 'a', 'a', 'b', 'a', 't', 'p',
'k', 'i', 'n', 'x' };
		String s = new String(ch);
		System.out.println(s);
	}

}
```

**Example2:**

String s1="javabybhanu";

Example3:

```java
package stringInJava;

public class Example1 {

	public static void main(String[] args) {
		String s1 = "test";

		// trying to change the string
		s1.concat("5test");
		System.out.println(s1);
		// output "test"
	}
}
```

**How to create Object of String**

There are two ways to create object of string.

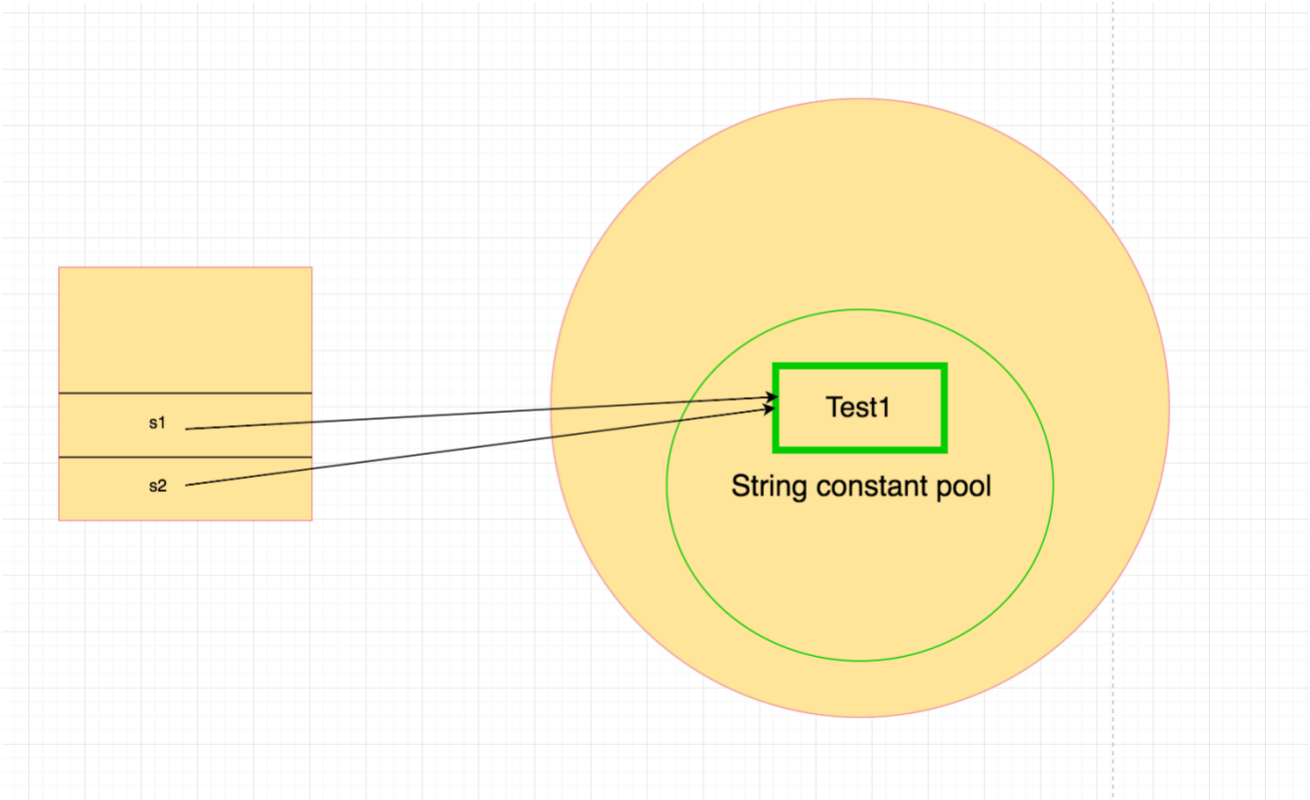
1. By string literal
2. By new keyword

**Java String literal is created by using double quotes.**

Example:

```
String s1 = "Test1";
String s2 = "Test1";
```

Each Time when we create the string object JVM checks the string pool, If the string is already existing in the pool, new object will point to the same object.




**Why java uses concept of string literal?**

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

**String object creation By New Keyword**

String s=**new** String("Hello”);

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

**String Example:**

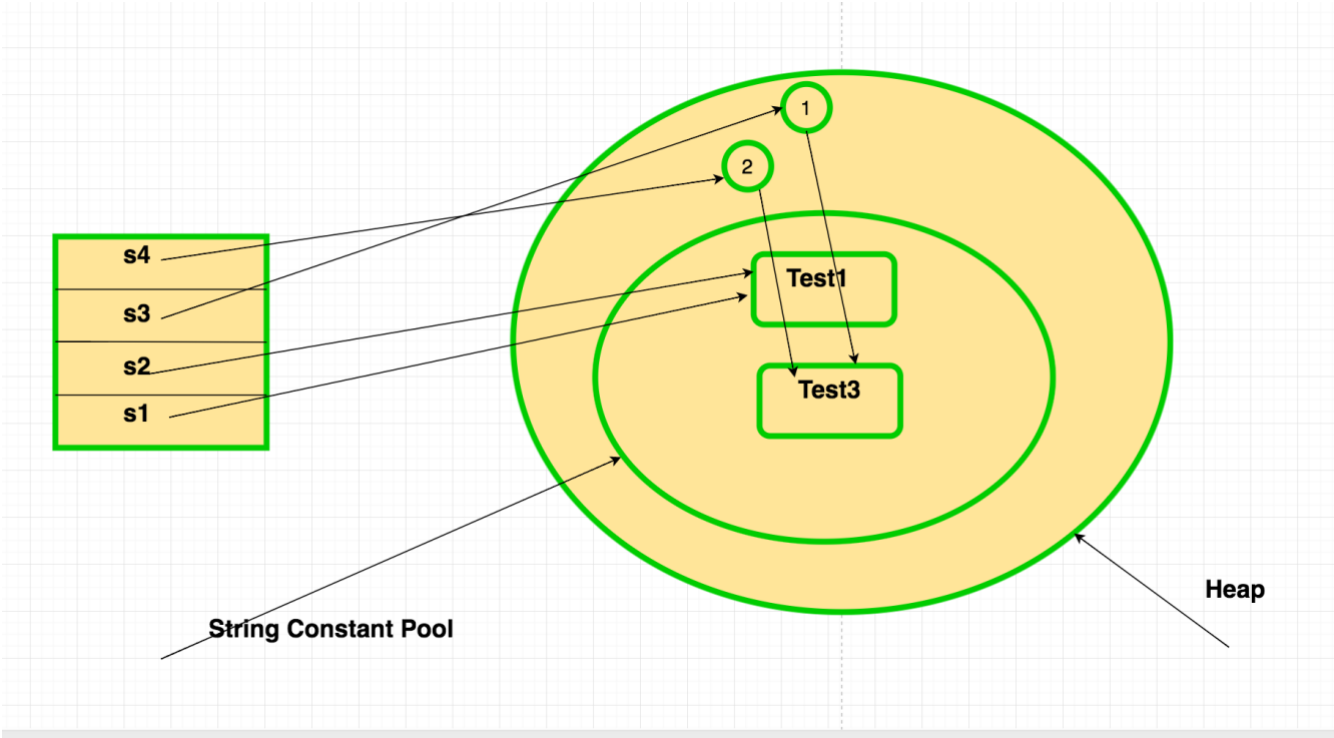
```java
package stringInJava;

public class Example1 {

	public static void main(String[] args) {
		// both s1 and s2 will point to same object in
constant pool
		String s1 = "Test1";
		String s2 = "Test1";

		// both s3 and s4 will point to different
object in non-constant pool
		String s3 = new String("Test3");
		String s4 = new String("Test3");

		System.out.println(s1);
		System.out.println(s2);
		System.out.println(s3);
		System.out.println(s4);
	}
}
```



## Java String compare

We can compare string in java on the basis of content and reference.

## There are three ways to compare string in java:

- By equals() method
- By = = operator
- By compareTo() method

**String compare by equals() method**

The String equals() method compares the original content of the string. It compares values of string for equality. String class provides two methods:

- **public boolean equals(Object another)** compares this string to the specified object.
- **public boolean equalsIgnoreCase(String another)** compares this String to another string, ignoring case.

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu";
		String s2 = "bhanu";
		String s3 = new String("bhanu");
		String s4 = "Singh";
		// equals will check for content
		// true
		System.out.println(s1.equals(s2));
		// true
		System.out.println(s1.equals(s3));
		// false
		System.out.println(s1.equals(s4));
	}
}
```

**Example:**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "Bhanu";
```

```
		String s2 = "BHANU";
		// false equals will check for exact case content
		System.out.println(s1.equals(s2));
		// true equalsIgnoreCase will check for exact content
		// but it is not case sensitive
		System.out.println(s1.equalsIgnoreCase(s2));
	}
}
```

**String compare by == operator**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "Sachin";
		String s2 = "Sachin";
		String s3 = new String("Sachin");
		// == will check for object reference
		// true (because both refer to same instance)
		System.out.println(s1 == s2);
		// false (because s3 refers to instance created in non constant pool)
		System.out.println(s1 == s3);
		// equals will check for content not for object reference
		System.out.println(s1.equals(s2));
	}
}
```

**String Concatenation in Java**

**There are two ways to concat string in java:**

- By + (string concatenation) operator
- By concat() method

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s = "Bhanu" + " Pratap";
		// + will concat the string
		// Bhanu Pratap
		System.out.println(s);
```

```
                String s1 =40+10+"Bhanu"+40+30;
                // here first 2 numbers will get added like summation and the moment String
"Bhanu"
                // get added with number then last 2 numbers will act like string addition
                // output: 50Sachin4030
                System.out.println(s1);

                // by concat Example
                String s3 = "Test1";
                String s4 = "Test2";
                // here in reference s5 we are doing concat of s3 and s4 so
                // s5 becomes "Test1Test2". this concat is happening because we have
                // changed the reference itself, we are not modifying in same reference
                String s5 = s3.concat(s4);
                System.out.println(s5);
        }
}
```

**String Methods:**

| Method Name | Return Type |
|---|---|
| char charAt(int index) | returns char value for the particular index |
| int length() | returns string length |
| static String format(String format, Object... args) | returns formatted string |
| static String format(Locale l, String format, Object... args) | returns formatted string with given locale |
| String substring(int beginIndex) | returns substring for given begin index |
| String substring(int beginIndex, int endIndex) | returns substring for given begin index and end index |
| String intern() | returns interned string |
| int indexOf(int ch) | returns specified char value index |
| int indexOf(int ch, int fromIndex) | returns specified char value index starting with given index |
| int indexOf(String substring) | returns specified substring index |
| int indexOf(String substring, int fromIndex) | returns specified substring index starting with given index |
| String toLowerCase() | returns string in lowercase. |
| String toLowerCase(Locale l) | returns string in lowercase using specified locale. |
| String toUpperCase() | returns string in uppercase. |

| | |
|---|---|
| String toUpperCase(Locale l) | returns string in uppercase using specified locale. |
| String trim() | removes beginning and ending spaces of this string. |
| static String valueOf(int value) | converts given type into string. It is overloaded. |
| boolean contains(CharSequence s) | returns true or false after matching the sequence of char value |
| static String join(CharSequence delimiter, CharSequence... elements) | returns a joined string |
| static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) | returns a joined string |
| boolean equals(Object another) | checks the equality of string with object |
| boolean isEmpty() | checks if string is empty |
| String concat(String str) | concatinates specified string |
| String replace(char old, char new) | replaces all occurrences of specified char value |
| String replace(CharSequence old, CharSequence new) | replaces all occurrences of specified CharSequence |
| static String equalsIgnoreCase(String another) | compares another string. It doesn't check case. |
| String[] split(String regex) | returns splitted string matching regex |
| String[] split(String regex, int limit) | returns splitted string matching regex and limit |

**String Methods**

**charAt()**

The **java string charAt()** method returns *a char value at the given index number*. The index number starts from 0. It returns StringIndexOutOfBoundsException if given index number is greater than this string or negative index number.

Syntax

```
public char charAt(int index)
```

**Parameter**
Index starts from 0;

**Returns:**

Char value at given index.

**Example**:

```
public class Example1 {

	public static void main(String args[]) {
		String name = "bhanuTest";
		// returns the char value at the 4th index
		char ch = name.charAt(4);
		System.out.println(ch);
	}
}
```

**compareTo()**

The **java string compareTo()** method compares the given string with current string lexicographically. It returns positive number, negative number or 0.

Lexical order is nothing but alphabetically order. compareTo methods does a sequential comparison of letters in the string that have the same position.

- if s1 > s2, it returns positive number
- if s1 < s2, it returns negative number
- if s1 == s2, it returns 0

**Syntax:**

**public int** compareTo(String anotherString);

**Parameter**

Second String which need to be compared.

**Returns**

an integer value

**Example:**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu";
		String s2 = "pratap";
		String s3 = "singh";
		String s4 = "bhanu";
```

```
		System.out.println(s1.compareTo(s2));// -14
		System.out.println(s1.compareTo(s3));// -17
		System.out.println(s1.compareTo(s4));// 0
		System.out.println(s3.compareTo(s1));// 17
		String s5 = "a";
		String s6 = "b";
		String s7 = "c";
		String s8 = "d";
		String s9 = "a";
		System.out.println(s5.compareTo(s9));// 0 because both are equals
		System.out.println(s6.compareTo(s7));// -1 because "b" is 1 times lesser than
"c"
		System.out.println(s5.compareTo(s8));// -3 because "a" is 3 times lesser than
"d"
		System.out.println(s8.compareTo(s5));// 3 because "d" is 3 times grater than
"a"
	}
}
```

**concat()**

The **java string concat()** method *combines specified string at the end of this string*. It returns combined string.

**Syntax**

**public** String concat(String anotherString)

**Parameter**

Second String which need to be concat.

**Returns**

combined string

```
package seleniumBasic;

public class Example1 {

	public static void main(String args[]) {
		String s1 = "Bhanu Pratap";
		s1.concat("is immutable");
		// since String is immutable we will get output as "Bhanu Pratap"
		// concat will not change the data
		System.out.println(s1);
		// here we are changing the reference itself. we are reassigning
```

```
		// s1 = s1.concat() and this will change the string
		s1 = s1.concat(" is immutable so assign it explicitly");
		System.out.println(s1);
	}
}
/**
Bhanu Pratap
Bhanu Pratap is immutable so assign it explicitly
*/
```

**contains()**

**contains()** method searches the sequence of characters in this string. It returns *true* if sequence of char values are found in this string otherwise returns *false*.

**Syntax**
**public boolean contains(CharSequence sequence)**
**Parameter**
sequence : specifies the sequence of characters to be searched.

**Returns**
true if sequence of char value exists, otherwise false.

**Throws**
NullPointerException : if sequence is null.

**Example:**

```
package seleniumBasic;

public class Example1 {

	public static void main(String args[]) {
		String name = "bhanu pratap singh";
		System.out.println(name.contains("bhnau"));
		System.out.println(name.contains("about"));
		System.out.println(name.contains("singh"));
	}
}
/**
false
false
true
*/
```

**endsWith()**

**endsWith()** method checks if this string ends with given suffix. It returns true if this string ends with given suffix else returns false.

**Syntax**
**public boolean** endsWith(String s1)

**Parameter**
s1 : Sequence of character

**Returns**
true or false

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu pratap singh";
		System.out.println(s1.endsWith("h")); // true
		System.out.println(s1.endsWith("point")); // false
	}
}
```

**equals()**

**equals()** method compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.

The String equals() method overrides the equals() method of Object class.

**Syntax:**
**public boolean** equals(Object obj);

**Parameter**
obj : another object i.e. compared with this string.

**Returns**
true if characters of both strings are equal otherwise false.

**Overrides**
equals() method of java Object class.

**Example**:

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu";
		String s2 = "bhanu";
		String s3 = "BHANU";
		String s4 = "singh";
		// true because content and case is same
		System.out.println(s1.equals(s2));
		// false because case is not same
		System.out.println(s1.equals(s3));
		// false because content is not same
		System.out.println(s1.equals(s4));
	}
}
```

**equalsIgnoreCase()**

**equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string. It is like equals() method but doesn't check case. If any character is not matched, it returns false otherwise it returns true.

**Syntax:**

**public boolean** equalsIgnoreCase(String s1);

**Parameter**
s1 : another string i.e. compared with this string.

**Returns**
It returns true if characters of both strings are equal ignoring case otherwise false.

**Example:**

```
public class Example1 {
	public static void main(String args[]){
		String s1="bhanu";
		String s2="bhanu";
		String s3="BHANU";
		String s4="singh";
		//true because content and case both are same
		System.out.println(s1.equalsIgnoreCase(s2));
		//true because case is ignored
		System.out.println(s1.equalsIgnoreCase(s3));
		//false because content is not same
		System.out.println(s1.equalsIgnoreCase(s4));
		}
}
```

## format()

**format()** method returns the formatted string by given locale, format and arguments.

If you don't specify the locale in String.format() method, it uses default locale by calling *Locale.getDefault()* method.

**Syntax:**

```
public static String format(Locale loc, String form, Object... args);
        and,
public static String format(String form, Object... args);
```

**Parameter:**

loc– locale value to be applied on the format() method
form– format of the output string
args– It specifies the number of arguments for the format string.It may be zero or more.

**Return:**

This method returns a formatted string.

**Exception:**

NullPointerException -If the format is null.
IllegalFormatException -If the format specified is illegal or there are insufficient arguments.

**Example**:

```
package seleniumBasic;

public class Example1 {
        public static void main(String args[]) {

                String str = "bhanupratap";

                // Concatenation of two strings
                String gfg1 = String.format("My name is %s", str);

                // Output is given upto 8 decimal places
                String str2 = String.format("My salary is %.8f", 47.65734);

                // between "My earning is" and "47.65734000" there are 15 spaces
                String str3 = String.format("My earning is %15.8f", 47.65734);

                System.out.println(gfg1);
                System.out.println(str2);
                System.out.println(str3);
        }
```

```
}
/**
My name is bhanupratap
My salary is 47.65734000
My earning is     47.65734000
*/
```

**Example:**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String str1 = "BHANU";
		String str2 = "PRATAP";

		// %1$ represents first argument, %2$ second argument
		String s1 = String.format("My name" + " is: %1$s, %1$s and %2$s", str1, str2);
		System.out.println(s1);
	}
}
```

**Output**:
My name is: BHANU, BHANU and PRATAP

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		int num = 7044;
		// Output is 3 zero's("000") + "7044",
		// in total 7 digits
		String s1 = String.format("%07d", num);
		System.out.println(s1);
	}
}
/**
Output:
0007044
*/
```

**getBytes()**

**Syntax**

```
public byte[] getBytes()
```

```
public byte[] getBytes(Charset charset)
public byte[] getBytes(String charsetName)throws UnsupportedEncodingException
```

**Example**

```
package seleniumBasic;

import java.io.UnsupportedEncodingException;

public class Example1 {
	public static void main(String args[]) {

		String str = new String("Bhanu");
		byte[] array1 = str.getBytes();
		System.out.print("Default Charset encoding:");
		for (byte b : array1) {
			System.out.print(b);
		}
		System.out.println();
		System.out.print("UTF-16 Charset encoding:");

		try {
			byte[] array2 = str.getBytes("UTF-16");
			for (byte b1 : array2) {
				System.out.print(b1);
			}
			System.out.println();

			byte[] array3 = str.getBytes("UTF-16BE");
			System.out.print("UTF-16BE Charset encoding:");
			for (byte b2 : array3) {
				System.out.print(b2);
			}
			System.out.println();

		} catch (UnsupportedEncodingException ex) {
			System.out.println("Unsupported character set" + ex);
		}
	}
}
/**
Default Charset encoding:6610497110117
UTF-16 Charset encoding:-2-1066010409701100117
UTF-16BE Charset encoding:066010409701100117
*/
```

In the above example we have done encoding using charset UTF -16 and UTF - 16BE, there are many other standard charset like:

| Charset | Description |
|---|---|
| US-ASCII | Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set |
| ISO-8859-1 | ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1 |
| UTF-8 | Eight-bit UCS Transformation Format |
| UTF-16BE | Sixteen-bit UCS Transformation Format, big-endian byte order |
| UTF-16LE | Sixteen-bit UCS Transformation Format, little-endian byte order |
| UTF-16 | Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark |

## getChars()

**Syntax**
**public void** getChars(**int** srcBegin, **int** srcEnd, **char**[] dest, **int** destBegin);

**Parameters description:**
**srcBegin** – index of the first character in the string to copy.
**srcEnd** – index after the last character in the string to copy.
**dest** – Destination array of characters in which the characters from String gets copied.
**destBegin** – The index in Array starting from where the chars will be pushed into the Array.

It throws IndexOutOfBoundsException – If any of the following conditions occurs:
(srcBegin<0) srcBegin is less than zero. (srcBegin>srcEnd) srcBegin is greater than srcEnd.
(srcEnd > length of string) srcEnd is greater than the length of this string.
(destBegin<0) destBegin is negative.
dstBegin+(srcEnd-srcBegin) is larger than dest.length.

**Example**

```
public class Example1 {

	public static void main(String args[]) {
		// creating string object
		String str = new String("This is a String Tutorial");
		// creating character array of size 6
		char[] array = new char[6];

		//str.getChars(srcBegin, srcEnd, dst, dstBegin);
		// this method will push data in array from 10th to 15th index.
		// in str string 10th index data is "S" and 15th index is "g"
```

```
		// when we are coping data into array we need to copy from oth index
		str.getChars(10, 16, array, 0);

		System.out.println("Array Data:");

		// running for each loop to print character array data
		for (char temp : array) {
			// using "print" not "println" method so that we will see data in
sequence
			System.out.print(temp);
		}
		// this println is just for new line
	System.out.println();

	// character array
		char[] array2 = new char[] { 'b', 'b', 'b', 'b', 'b', 'b', 'b', 'b' };

		// this method will push data in array from 10th to 15th index.
		// in str string 10th index data is "S" and 15th index is "g"
		// when we are coping data into array we need to copy from 2nd index.
		// so 10th index will be added after "bb"
		str.getChars(10, 16, array2, 2);

		System.out.println("Second Array Data:");
		for (char temp : array2) {
			// using "print" not "println" method so that we will see data in
sequence
			System.out.print(temp);
		}
	}
}
/**
Array Data:
String
Second Array Data:
bbString
*/
```

**indexOf()**

**indexOf()** method returns index of given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

**Syntax**

**int indexOf(int ch):** It returns the index of the first occurrence of character ch in a String.

**int indexOf(int ch, int fromIndex):** It returns the index of first occurrence if character ch, starting from the specified index “fromIndex”.
**int indexOf(String str):** Returns the index of string str in a particular String.
**int indexOf(String str, int fromIndex):** Returns the index of string str, starting from the specified index “fromIndex”.

**Parameters**

**ch**: char value i.e. a single character e.g. 'a'

**fromIndex**: index position from where index of the char value or substring is retured

**substring**: substring to be searched in this string

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		// creating 4 string object
		String str1 = new String("This is a BhanuString tutorial");
		String str2 = new String("Bhanu");
		String str3 = new String("String");
		String str4 = new String("Strings");

		System.out.println("Index of B in str1: " + str1.indexOf('B'));
		// in str1 we are looking 'B' after 15th index of str1.
		// we will get output as -1, since 'B' is not available after
		// 15th index
		System.out.println("Index of B in str1 after 15th char:" + str1.indexOf('B',
15));
		// in str1 we are looking 'B' after 30th index of str1.
		// we will get output as -1, since 'B' is not available after
		// 15th index
		System.out.println("Index of B in str1 after 30th char:" + str1.indexOf('B',
30));
		System.out.println("Index of string str2 in str1:" + str1.indexOf(str2));
		// in str1 we are looking str2 after 15th index of str1.
		// we will get output as -1, since str2 is "Bhanu" which is not available after
		// 15th index
		System.out.println("Index of str2 after 15th char" + str1.indexOf(str2, 15));
		System.out.println("Index of string str3:" + str1.indexOf(str3));
		System.out.println("Index of string str4" + str1.indexOf(str4));
		System.out.println("Index of harcoded string:" + str1.indexOf("is"));
		// in str1 we are looking 'is' after 4th index of str1.
```

```
                // we will get output as positive number
                System.out.println("Index of hardcoded string after 4th char:" +
str1.indexOf("is", 4));
        }
}
/**
Index of B in str1: 10
Index of B in str1 after 15th char:-1
Index of B in str1 after 30th char:-1
Index of string str2 in str1:10
Index of str2 after 15th char-1
Index of string str3:15
Index of string str4-1
Index of harcoded string:2
Index of hardcoded string after 4th char:5
*/
```

**intern()**

**intern()** method returns the interned string. It returns the canonical representation of string.

It can be used to return string from pool memory, if it is created by new keyword.

**Syntax**

**public** String intern();

**Returns**

interned string

**Example**

```
public class Example1 {
        public static void main(String args[]) {
                // s1 object will created in non-constant pool
                String s1 = new String("bhanu");
                // s2 object will get created in string pool
                String s2 = "bhanu";
                // returns string from pool, now it will be same as s2
                String s3 = s1.intern();
                // false because reference is different
                System.out.println(s1 == s2);
                // true because reference is same
                System.out.println(s2 == s3);
        }
```

}

## isEmpty()

**isEmpty()** method checks if this string is empty. It returns *true*, if length of string is 0 otherwise *false*.

**Syntax**
**public boolean isEmpty();**

**Returns**

true if length is 0 otherwise false.

**Example:**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "";
		String s2 = "Bhanu";
		// true
		System.out.println(s1.isEmpty());
		// false
		System.out.println(s2.isEmpty());
	}

}
```

## join()

The **java string join()** method returns a string joined with given delimiter. In string join method, delimiter is copied for each elements.

In case of null element, "null" is added.

**Syntax**

**public static String join(CharSequence delimiter,CharSequence... elements);**

**Example:**

```
public class Example1 {
	public static void main(String args[]) {
		// The first argument to this method is the delimiter
		String str = String.join("^", "Bhanu", "Pratap", "Singh");
		System.out.println(str);
		//output: Bhanu^Pratap^Singh
```

```
	}

}
```

Example

```
import java.util.Arrays;
import java.util.List;

public class Example1 {
	public static void main(String args[]) {
		// Converting an array of String to the list
		List<String> list = Arrays.asList("bhanu", "pratap", "singh");
		// list data will get join by "|"
		String names = String.join(" | ", list);
		System.out.println(names);
		// output
		// bhanu | pratap | singh
	}

}
```

## lastIndexOf()

**lastIndexOf()** method returns last index of the given character value or substring. If it is not found, it returns -1. The index counter starts from zero.

**It has four methods**

| Method | Description |
|---|---|
| int lastIndexOf(int ch) | returns last index position for the given char value |
| int lastIndexOf(int ch, int fromIndex) | returns last index position for the given char value and from index |
| int lastIndexOf(String substring) | returns last index position for the given substring |
| int lastIndexOf(String substring, int fromIndex) | returns last index position for the given substring and from index |

**Example**

```
package seleniumBasic;
```

```java
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu pratap singh";
		String s2 = "singh";
		int index1 = s1.lastIndexOf('s');
		// will get here positive number
		System.out.println(index1);
		// here we are checking last index of s but before 2nd index
		//s1.lastIndexOf(str, fromIndex)
		int index2 = s1.lastIndexOf('s',2);
		System.out.println(index2);
		int index3 = s1.lastIndexOf(s2);
		System.out.println(index3);
		// here we are checking last index of s2 but before 10th index
		int index4 = s1.lastIndexOf(s2,10);
		System.out.println(index4);
	}
}
/**
13
-1
13
-1

*/
```

**length()**

**length()** method length of the string. It returns count of total number of characters.

**Syntax**

**public int length();**

**Returns**

length of characters

```java
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanupratap";
		String s2 = "singh";
		System.out.println("string length is: " + s1.length());
		System.out.println("string length is: " + s2.length());
	}
```

```
}
/**
string length is: 11
string length is: 5

*/
```

**replace()**

**replace()** method returns a string replacing all the old char or CharSequence to new char or CharSequence.

**Syntax**

**public String replace(char oldChar, char newChar);**

**Returns**

replaced string

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu pratap singh";
		// replaces all occurrences of 'a' to 'e'
		String replaceString = s1.replace('a', 'e');
		System.out.println(replaceString);
	}

}
```

**Output**

bhenu pretep singh

**replaceAll()**

**replaceAll()** method returns a string replacing all the sequence of characters matching regex and replacement string

**Syntax**
**public String replaceAll(String regex, String replacement);**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "I am bhanu pratap singh I am bhanu pratap";
		// replaces all occurrences of "is" to "was"
		String replaceString = s1.replaceAll("pratap", "kumar");
		System.out.println(replaceString);
```

```
	}
}
/**
I am bhanu kumar singh I am bhanu kumar
*/
```

**split()**

**split()** method splits this string against given regular expression and returns a char array.

**Returns**

array of strings

**Example**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu;pratap;singh";
		// splits the string based on semicolon
		String[] data = s1.split(";");
		// using java foreach loop to print elements of string array
		for (String w : data) {
			System.out.println(w);
		}
		System.out.println("---------");
		String s2 = "bhanu pratap singh";
		// splits the string based on whitespace
		String[] data1 = s2.split(" ");
		// using java foreach loop to print elements of string array
		for (String w : data1) {
			System.out.println(w);
		}
	}

}
```

**Output:**

```
bhanu
pratap
singh
---------
bhanu
pratap
```

singh

## startsWith()

**startsWith()** method checks if this string starts with given prefix. It returns true if this string starts with given prefix else returns false.

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu pratap singh";
		// true
		System.out.println(s1.startsWith("bh"));
		// true
		System.out.println(s1.startsWith("bhanu pratap"));
		// false
		System.out.println(s1.startsWith("pratap"));
	}
}
```

## substring()

**substring()** method returns a part of the string.

**String substring(int beginIndex):** Returns the substring starting from the specified index(beginIndex) till the end of the string. For e.g. "bhanu".substring(2) would return "anu". This method throws IndexOutOfBoundsException If the beginIndex is less than zero or greater than the length of String

**Example**

```
package seleniumBasic;

public class Example1 {
	public static void main(String args[]) {
		String str = new String("bhanu pratap singh ");
		System.out.println("Substring starting from index 3:");
		// it will return data from index 3 to last
		System.out.println(str.substring(3));
		System.out.println("Substring starting from index 3 and ending at 15:");
		// it will return data from index 3 to 14th index
		System.out.println(str.substring(3, 15));
	}
}
/**
Substring starting from index 3:
```

```
nu pratap singh
Substring starting from index 3 and ending at 15:
nu pratap si
*/
```

**toCharArray()**

**toCharArray()** method converts this string into character array.

**Syntax**

```
public char[] toCharArray();
```

**Example**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "bhanu";
		char[] ch = s1.toCharArray();
		for (int i = 0; i < ch.length; i++) {
			System.out.println(ch[i]);
		}
	}
}
```

Output

```
b
h
a
n
u
```

**toLowerCase()**

**toLowerCase()** method returns the string in lowercase letter.

**Example**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "BHanu";
		System.out.println(s1.toLowerCase());
	}
}
```

Output:

```
bhanu
```

**toUpperCase()**

**toUpperCase()** method returns the string in uppercase letter

**Example**

```
public class Example1 {
	public static void main(String args[]) {
		String s1 = "BHanu";
		System.out.println(s1.toUpperCase());
	}
}
```

Output:
BHANU

## String Buffer in Java

String Buffer is used to create Mutable String, which means we can modify the String once it is created.

The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.

```
package stringBufferInJava;

public class Example1 {

	public static void main(String[] args) {
		// String object
		String s1 = "Test1";
		// trying to change the string
		s1.concat("htest");
		// since string is  immutable, so we can't change the string
		// here we will get Test1
		System.out.println(s1);

		// creating object of StringBuffer
		StringBuffer sb = new StringBuffer("abc");
		// StringBuffer has append method and it is immutable
		sb.append("test");
		// here we will get output abctest
		System.out.println(sb);
	}
}
```

**Output:**

Test1

abctest

## Constructor of String Buffer

| | |
|---|---|
| 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. |

## Method of String Buffer

| Access Type and return Type | Method Name | Description |
|---|---|---|
| public synchronized StringBuffer | delete(int startIndex, int endIndex) | Is used to delete the string from specified startIndex and endIndex. |
| public synchronized StringBuffer | reverse() | Is used to reverse the string. |
| public synchronized StringBuffer | append(String s) | Is used to append the specified string with thIs string. The append() method Is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc. |
| public synchronized StringBuffer | insert(int offset, String s) | Is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc. |
| public synchronized StringBuffer | replace(int startIndex, int endIndex, String str) | Is used to replace the string from specified startIndex and endIndex. |

| | | |
|---|---|---|
| public int | capacity() | Is used to return the current capacity. |
| public void | ensureCapacity(int minimumCapacity) | 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. total number of characters. |
| public String | substring(int beginIndex) | Is used to return the substring from the specified beginIndex. |
| public String | substring(int beginIndex, int endIndex) | Is used to return the substring from the specified beginIndex and endIndex. |

**Append Method in String Buffer**

The append() method concatenates the given argument with this string.

```
package stringBufferInJava;

public class AppendInJava {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer("Java ");
		sb.append("Test");// now original string is
changed
		System.out.println(sb);// prints Java Test
	}
}
```

**output:** Java Test

**Insert Method in String Buffer**

The insert() method inserts the given string with this string at the given position.

**package** stringBufferInJava;

```
public class InsertMethodInStringBuffer {

	public static void main(String[] args) {
		StringBuffer sb=new StringBuffer("Java ");
		//now original string is changed
		// here at 1st index "Test" will be inserted
		sb.insert(1,"Test");
		//prints JTestava
		System.out.println(sb);
	}
}
```

output: JTestava

**Replace Method in String Buffer**

The replace() method replaces the given string from the specified beginIndex and endIndex.

```
package stringBufferInJava;

public class ReplaceMethodInStringBuffer {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer("Java");
		// here between 1 to 3 th index string will
		// be replaced by Test do "av" will be replaced by "Test"
		sb.replace(1, 3, "Test");
		// prints JTesta
		System.out.println(sb);
	}
}
```

**Delete Method in String Buffer**

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

```
package stringBufferInJava;

public class DeleteMethodInStringBuffer {

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

```
        StringBuffer sb=new StringBuffer("Play");
        // it will delete data from 1 to 3rd index
        sb.delete(1,3);
        //prints Py (so remaining data is P and y)
        System.out.println(sb);
    }
}
```

**Reverse Method in String Buffer**

The reverse() method of StringBuilder class reverses the current string.

```
package stringBufferInJava;

public class ReverseMethodInStringBuffer {

    public static void main(String[] args) {
        StringBuffer sb = new StringBuffer("Java");
        sb.reverse();
        System.out.println(sb);// prints avaJ
    }
}

Output: avaJ
```

**Capacity Method in String Buffer**

The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
package stringBufferInJava;

public class CapacityInStringBuffer {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer();
		// Here Size is 16
		System.out.println(sb.capacity());
		sb.append("Test");
		// Here Size is 16
		System.out.println(sb.capacity());
		sb.append("java support multiple OS");
		// Here Size is 34
		System.out.println(sb.capacity());
	}
}
```

**Ensure Capacity in String Buffer**

The ensureCapacity() method of StringBuffer class ensures that the given capacity is the minimum to the current capacity. If it is greater than the current capacity, it increases the capacity by (oldcapacity*2)+2. For example if your current capacity is 16, it will be (16*2)+2=34.

```
package stringBufferInJava;

public class EnsureCapacityInStringBuffer {

	public static void main(String[] args) {
		StringBuffer sb = new StringBuffer();
		// default size of StringBuffer is 16
		System.out.println(sb.capacity());
		// appending "Test" in StringBuffer
		sb.append("Test");
		// Still capacity is 16, since size of "Test" is 4
		System.out.println(sb.capacity());
		// appending string size more than 16 length
		sb.append("java support multiple OS");
		// now capacity becomes (16*2)+2 = 34 i.e
		// (oldcapacity*2)+2
		System.out.println(sb.capacity());
		// no change will happen in capacity, ensureCapacity will
		// not decrease the capacity
```

```
                sb.ensureCapacity(10);
                // still capacity will be 34
                System.out.println(sb.capacity());
                // ensureCapacity will increase the capacity.
                // since 50 is more than 34 then capacity becomes
                // // now (34*2)+2
                sb.ensureCapacity(50);
                // now capacity is 70
                System.out.println(sb.capacity());
        }

}
/**
16
16
34
34
70
*/
```

**String Builder In Java**

**J**ava StringBuilder class is same as StringBuffer class except that it is non-synchronized. It is available since JDK 1.5.

**Difference Between String and String Buffer.**

| **String** | **StringBuffer** |
|---|---|
| String class is immutable. | StringBuffer class is mutable. |
| String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |

| | |
|---|---|
| String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

**Difference Between String Buffer and String Builder.**

| **StringBuffer** | **StringBuilder** |
|---|---|
| StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |

**Example:**

```
package stringBufferInJava;

public class StringBufferAndBuilderExample {

	public static void main(String[] args) {
		StringBuffer buffer = new
StringBuffer("Java");
		buffer.append("Test");
		System.out.println(buffer);
		System.out.println("------------");
		StringBuilder builder = new
StringBuilder("Java");
		builder.append("Test");
		System.out.println(builder);
	}

}
```

**Output:**

```
JavaTest
------------
JavaTest
```

**Performance Test of String Buffer and String Builder**

```
package stringBufferInJava;

public class PerformanceOfStringBufferAndStringBuilder {

	public static void main(String[] args) {

		long startTime = System.currentTimeMillis();

		StringBuffer sb = new StringBuffer("Java");
		for (int i = 0; i < 100000; i++) {
			sb.append("By Bhanu");
		}

		System.out.println("Time taken by StringBuffer: " +
(System.currentTimeMillis() - startTime) + "ms");

		startTime = System.currentTimeMillis();
```

```
                StringBuilder sb2 = new StringBuilder("Java");

                for (int i = 0; i < 100000; i++) {
                        sb2.append("By Bhanu");
                }

                System.out.println("Time taken by StringBuilder: " +
(System.currentTimeMillis() - startTime) + "ms");
        }
}
/**
Time taken by StringBuffer: 8ms
Time taken by StringBuilder: 5ms
*/
```

**Difference between String and StringBuffer**

| **String** | **StringBuffer** |
|---|---|
| String class is immutable. | StringBuffer class is mutable. |
| String is slow and consumes more memory when you concat too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when you cancat strings. |
| String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |

**Difference between StringBuffer and StringBuilder**

| **StringBuffer** | **StringBuilder** |
| --- | --- |
| StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |