

Unit 6

String Handling

Generally, String is sequence of character but in java it is treated as object of type String. Implementing String as object allow java to provide a feature to handle different operation on string. Treating String as object provides convenient way to handle string operation such as comparing string, changing to upper and lower case, search for a substring etc. When a String object is created then such string cannot be changed i.e. once String object is created we cannot change the character that comprise that String but can perform all type of operation in it. If original string needs to be modified then new version of String is created that contains the modification leaving original String unchanged. For modification of original String, built in class String Buffer and String Builder can be used which will be cover at last of this chapter.

String object can be created in two ways:

1. By using String Literals:

String literals in java can be created using double quote. For Example:

```
String str = "Hello Third Semester";
```

Each time a string is created, JVM checks the string constant pool (heap memory) to check whether such literal or object already exist or not. If exist then reference of the instance from pool is returned. If string does not exist a new string instance is created and placed in pool. At first, no any string object is created so, creating a string literal will create instance and placed on memory. From next time, for each created String literal, string pool (memory) is checked. If match found reference to object is returned otherwise new string object is placed on memory pool. Java uses concept of string literal for memory efficient.

2. By using new Keyword:

String object can be created same as creating object of normal class.

```
String str = new String ();
```

Above statement will create an instance of String with no characters in it. To create a string with initial value, String class provides a variety of constructor to handle it.

To create String object with content in it we can use:

```
String str1 = new String ("hello third semester");
```

This will create a two object and one reference variable.

To create a String initialized by an array of characters following constructor can be used:

```
char chars [] = {'h','e','l','l','o'};
```

```
String s1 = new String (chars);
```

This will initialize s1 with the string "hello".

To specify a subrange of a character array (choosing characters from specified position to another position) following constructor is used

```
String (char chars[], int start_index, int numChars)
```

Here, start_index specifies the index at which the subrange begins, numChars specifies number of characters to use. Index in character starts with 0. For example

```
char chars1[] = {'t', 'h', 'I', 's', 'I', 's', 'j', 'a', 'v', 'a', '1'};
```

```
String s2 = new String (chars1, 2,3);
```

This will initialize s2 with string "isi". In above example, chars is character array, 2 specifies subrange will begins from second position of character array and 3 specifies three character to be count from beginning of subrange.

To create a String object that contains same character sequence as another String object following constructor is used:

```
String (String strObject)
```

Here, strObject is a String Object. For example

```
String s1 = new String ("hello");
```

```
String s2 = new String (s1);
```

String class also provides constructor that initialize a string when given a byte array.

```
String (byte chars[])
```

```
String (byte chars[ ], int startIndex, int numChars)
```

Here, char specifies the array of bytes and second form allows to specify subrange. In both constructor the byte to character conversion is done by using default character encoding of the platform. For example:

```
Byte chars[] = {97, 98, 99, 100, 101};
```

```
String s2 = new String (chars);
```

```
String s3 = new String(chars, 2,4)
```

Different ways to create string object:

```
package com.StringHandling;
```

```
public class StringHandlingDemo {
```

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

```
        //creating String
```

```
        String s1 = new String ("Hello Third Semester");
```

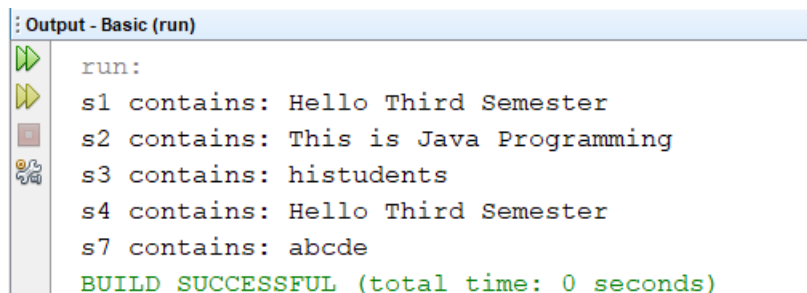
```
        String s2 = "This is Java Programming";
```

```
        char chars[]= {'h','i','s','t','u','d','e','n','t','s'};
```

```
        String s3 = new String(chars);
```

```
        String s4 = new String(s1);
```

```
byte []chr = {97,98,99,100,101};  
String s7 = new String (chr);//covert byte value into ascii value  
System.out.println("s1 contains: "+s1);  
System.out.println("s2 contains: "+s2);  
System.out.println("s3 contains: "+s3);  
System.out.println("s4 contains: "+s4);  
System.out.println("s7 contains "+s7);  
}  
}
```

Output:

```
Output - Basic (run)  
run:  
s1 contains: Hello Third Semester  
s2 contains: This is Java Programming  
s3 contains: histudents  
s4 contains: Hello Third Semester  
s7 contains: abcde  
BUILD SUCCESSFUL (total time: 0 seconds)
```

Finding String Length:

The number of character present in String is the length of string. To obtain length of String, legth() method is used:

Int length()

For example

```
String s1 = new String ("Hello");  
System.out.println("number of character in s1 is"+s1.length());
```

We can also find the length of string literal directly using length() function:

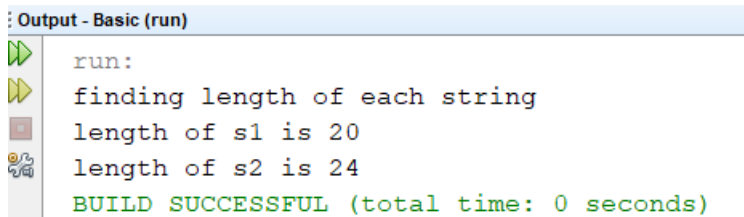
```
String s2 = "BCA";  
System.out.println("BCA".length());
```

Program for finding length of String:

```
package com.StringHandling;  
public class StringHandlingDemo {  
    public static void main(String[] args) {  
        //creating String  
        String s1 = new String ("Hello Third Semester");
```

```
String s2 = "This is Java Programming";  
System.out.println("finding length of each string");  
System.out.println("length of s1 is "+s1.length());  
System.out.println("length of s2 is "+s2.length());  
}  
}
```

Output:



```
Output - Basic (run)  
run:  
finding length of each string  
length of s1 is 20  
length of s2 is 24  
BUILD SUCCESSFUL (total time: 0 seconds)
```

String Concatenation:

Concatenation means joining two elements. Java does not allow operator to be applied to String objects. But “+” operator can be used to join two String, producing a String object as the result. for example:

```
String marks = “50”;  
String result = “he got “+ marks+ ”in Java programming”;  
System.out.println(result);
```

Joining the String is useful when we are creating long String literals. Concatenation will help to break down the string into smaller lines. String can be concatenate with other data type i.e. we can join int or other type in String. For example:

```
int marks = 50;  
String res = “Ram got ”+marks+ “in Java Programming”;  
System.out.println(res);
```

Here marks is int which will be converted into String as other operand are in String form. The compiler will covert an operand to its String equivalent if other operand after + operator is an instance of String. But if other type of operation are mix with String concatenation then result might be not match from what we expect for.

For example:

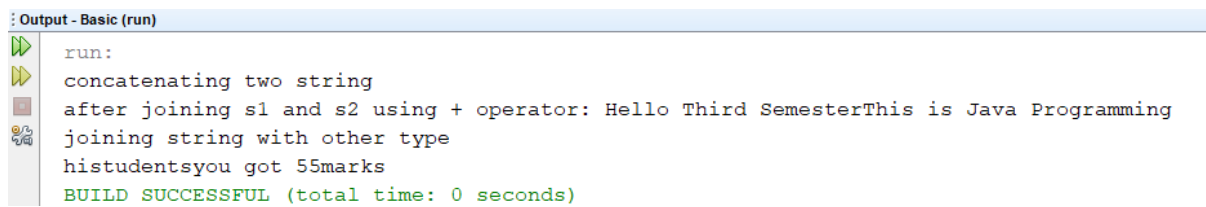
```
String s = “hello you got” + 5 + 5 + “marks”;  
System.out.println(s);
```

Program for joining two String:

```
package com.StringHandling;
```

```
public class StringHandlingDemo {  
    public static void main(String[] args) {  
        //creating String  
        String s1 = new String ("Hello Third Semester");  
        String s2 = "This is Java Programming";  
        System.out.println("concatenating two string ");  
        System.out.println("after joining s1 and s2 using + operator: "+ s1+s2);  
        System.out.println("joining string with other type");  
        System.out.println(s3+"you got "+5+5+"marks");  
    }  
}
```

Output:



```
run:  
concatenating two string  
after joining s1 and s2 using + operator: Hello Third SemesterThis is Java Programming  
joining string with other type  
histudentsyou got 55marks  
BUILD SUCCESSFUL (total time: 0 seconds)
```

String Comparison:

To compare String java provides operator and number of built-in methods. Some of them are

- **Using equals () or equalsIgnoreCase():**

Equals () method compare two strings for equality and return true if String matches otherwise false. This method compares the character inside a String object. Syntax:

boolean equals (Object str);

Here, str is the String object being compared with invoking String, Boolean is the return type of method which will return true if String are matched with one another otherwise return false. The comparison here is case sensitive which means that same letter in upper and lower cased is treated differently.

To perform comparison that ignores case difference, equalsIgnoreCase() method is used. When this method is used, it treats particular letter with upper and lower case as same. It also returns true if character in both String is same in same index otherwise return false. Syntax:

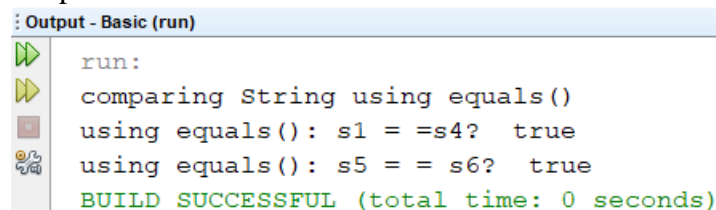
boolean equalsIgnoreCase(Object str)

Here, str is the String object being compared with invoking String, boolean is the return type of method which will return true if String are matched with one another otherwise return false. The comparison here is case in-sensitive which means that same letter in upper and lower cased is treated as same.

Following Program shows an example on using equals() and equalsIgnoreCase():

```
package com.StringHandling;
public class StringHandlingDemo {
    public static void main(String[] args) {
        //creating String
        String s1 = new String ("Hello Third Semester");
        String s4 = new String(s1);
        String s5 = "Hello Third Semester";
        String s6 = "Hello Third Semester";
        System.out.println("comparing String using equals()");
        System.out.println("using equals(): s1 ==s4? "+s1.equals(s4));
        System.out.println("using equals(): s5 == s6? "+s5.equals(s6));
    }
}
```

Output:



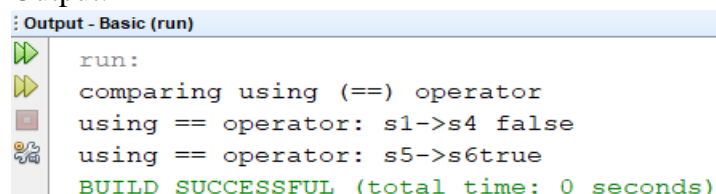
```
Output - Basic (run)
run:
comparing String using equals()
using equals(): s1 ==s4? true
using equals(): s5 == s6? true
BUILD SUCCESSFUL (total time: 0 seconds)
```

- **Using == operator:**

The double (==) equals to operator is also used to compare two string object reference to check whether they refer to same instance or not. If matches occur it returns true otherwise false.

```
package com.StringHandling;
public class StringHandlingDemo {
    public static void main(String[] args) {
        //creating String
        String s1 = new String ("Hello Third Semester");
        String s4 = new String(s1);
        String s5 = "Hello Third Semester";
        String s6 = "Hello Third Semester";
        System.out.println("comparing using (==) operator");
        System.out.println("using == operator: s1->s4 "+(s1==s4));
        System.out.println("using == operator: s5->s6"+(s5==s6));
    }
}
```

Output:



```
Output - Basic (run)
run:
comparing using (==) operator
using == operator: s1->s4 false
using == operator: s5->s6true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

The double equals to (==) operator is used to compare two String reference whether they point to same object or not. Here, s5 and s6 point to same object in String pool as they are created using same literals. Therefore, returns true. But in case of s1 and s4 they point to different object as both are created using new operator. due to this s1 point to its one object whereas s4 point to another object so (==) returns false

- **Compare To () method:**

This method compares value and returns an integer value which informs if the String compared is less than, equal to or greater than the other string. It compares String according to alphabetical order. A string is less than another if it comes before the other in dictionary order, a string is greater than another if it comes after the other in dictionary order, a string is equals to another if characters in both String are matched.

int compareTo(String str);

int compareToIgnoreCase(String str)

Here, str is the String being compared with the invoking String. The second form will ignore the case difference while comparing.

For example:

```
String s1 = "BCA Third";
```

```
String s2 = "BCA Third";
```

```
System.out.println(s1. compareTo(s2));
```

The result of above example will be one of the following:

Less than zero: if invoking String is less than a string being compared.

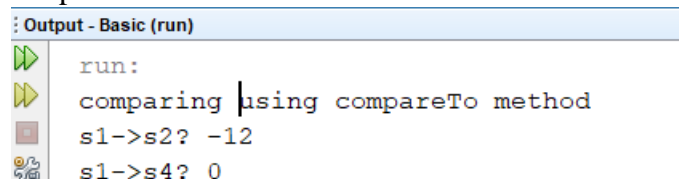
Greater than zero: if invoking String is greater than string being compared

Zero: if two strings are equal.

Following program demonstrate use of compareTo() method:

```
package com.StringHandling;  
public class StringHandlingDemo {  
    public static void main(String[] args) {  
        //creating String  
        String s1 = new String ("Hello Third Semester");  
        String s2 = "This is Java Programming";  
        String s4 = new String(s1);  
        System.out.println("comparing using compareTo method");  
        System.out.println("s1->s2? "+ s1.compareTo(s2));  
        System.out.println("s1->s4? "+s1.compareTo(s5));  
    }  
}
```

Output:



```
Output - Basic (run)  
run:  
comparing using compareTo method  
s1->s2? -12  
s1->s4? 0
```

Explanation:

As, the content of s1 and s4 are equal, compareTo() return zero. But content of s1 and s2 are not equal so whether s1 is larger than s2 or s2 is larger than s1 is checked. Here, s1 is smaller than s2 in term of alphabetical term so, negative value is returned.

- **startsWith() and endsWith():**

the startsWith() method determines whether a given String begins with specified string and endsWith() determines whether a given String ends with a specified string or not.

Syntax:

```
boolean startsWith(String str)
```

```
boolean endsWith(String str)
```

```
boolean startsWith(String str, int index)
```

here, str is a String being tested, index in third form specifies the starting point in invoking String at which search will begin. This method returns true if string match otherwise false. For example.

```
String s = "BCAThird";
```

```
s.startsWith("BCA");
```

```
s.endsWith("third");
```

```
s.startsWith("third", 3); //start searching from position 2 of invoking string
```

Following program demonstrate use of startsWith() and endsWith():

```
package com.StringHandling;
```

```
public class StringHandlingDemo {
```

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

```
        //creating String
```

```
        String s1 = new String ("Hello Third Semester");
```

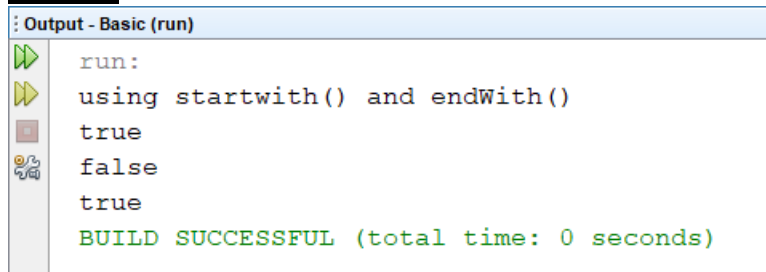
```
        System.out.println("using startwith() and endWith()");
```

```
        System.out.println(s1.startsWith("Hel"));
```

```
        System.out.println(s1.startsWith("Hel",5)); //starts searching from position 5
```

```
        System.out.println(s1.endsWith("ter"));}
```

Output:



```
Output - Basic (run)
run:
using startwith() and endWith()
true
false
true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

s1.startsWith("Hel") statement checks whether content in String object s1 starts with letter "Hel" or not. As s1 contains Hel at first so true is returned. s1.startsWith("Hel",5) statement starts checking letter "Hel" in s1 from fifth position. As, Hel does not exist from 5th position, it returns false. s1.endsWith("ter") statement checks whether content in String object s1 ends with letter "ter" or not. As s1 contains ter at last so true is returned.

Searching a Strings:

To search a specified character or substring from a particular String, java provides two methods:

indexOf():

this method searches for the first occurrence of a character or substring (particular piece of string). Syntax:

```
int indexOf(char ch) or int indexOf(String str)
```

```
int indexOf(char ch, int indexFrom) or int indexOf(String str, int indexForm)
```

here, ch is a character being searched and str is substring being searched. In second form, searching will start from specified index mentioned in indexForm parameter. These both method searches a character or substring from beginning to the end of String and return the index of first occurrence of such character if found otherwise return -1.

Following program demonstrate use of indexOf() method:

```
package com.StringHandling;
```

```
public class StringHandlingDemo {
```

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

```
        //creating String
```

```
        String s1 = new String ("Hello Third Semester");
```

```
        String s2 = "This is Java Programming";
```

```
        System.out.println("searching a string using indexOf() methods");
```

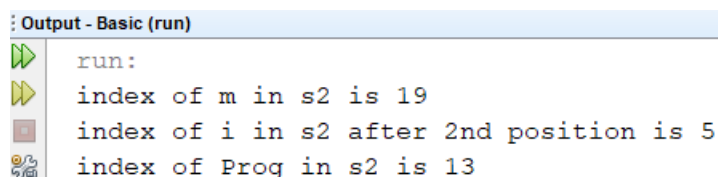
```
        System.out.println("index of J in s2 is "+(s2.indexOf("m")));
```

```
        System.out.println("index of i in s2 after 2nd position is "+(s2.indexOf('i',3)));
```

```
        System.out.println("index of Prog in s2 is "+(s2.indexOf("Prog")));
```

```
    }
```

```
}
```

Output:

```
Output - Basic (run)
run:
index of m in s2 is 19
index of i in s2 after 2nd position is 5
index of Prog in s2 is 13
```

Explanation:

For s2.indexOf("m")) statement, indexof() method start searching letter 'm' from 0th position and it is first found in 19th position so 19 is returned. For s2.indexOf('i',3), it starts searching

letter 'i' in s2 from third position. After third position, letter 'i' is found in position 5 so 5 is returned. For s2.indexOf("Prog"), search is made for substring "Prog" from 0th position and found on 13th position so 13 is returned.

lastIndexOf():

this method searches for the last occurrence of a character or substring i.e. this method starts searching from backward from end of String and return the index of specified character if match occurred. Syntax:

int lastIndexOf(char ch) or int lastIndexOf(String str)

int lastIndexOf(char ch, int indexFrom) or int lastIndexOf(String str, int indexFrom)

Here, ch is a character being searched and str is substring being searched. In second form, searching will start from specified index mentioned in indexFrom parameter. These both method searches a character or substring from end to beginning of String and return the index of first occurrence of such character if found otherwise return -1.

Following program demonstrate use of lastIndexOf():

```
package com.StringHandling;

public class StringHandlingDemo {

    public static void main(String[] args) {

        //creating String

        String s1 = new String ("Hello Third Semester");

        String s2 = "This is Java Programming";

        System.out.println("searching a string using lastIndexOf() methods");

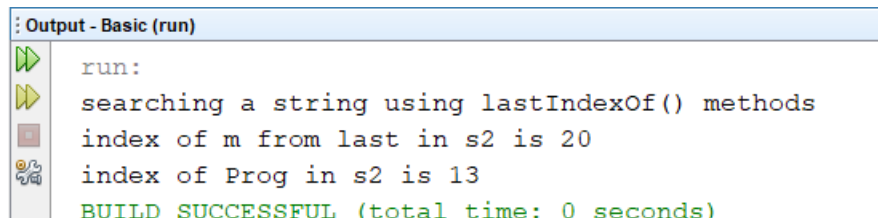
        System.out.println("index of m from last in s2 is" +(s2.lastIndexOf("m")));

        System.out.println("index of Prog in s2 is" +(s2.lastIndexOf("Prog")));

    }

}
```

Output:



```
Output - Basic (run)

run:
searching a string using lastIndexOf() methods
index of m from last in s2 is 20
index of Prog in s2 is 13
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

s2.lastIndexOf("m") statement start searching letter 'm' from last position of s2. As, letter 'm' is found on position 20 from last, 20 is returned.

String Conversion and toString():

To convert different types of value (int, long, Boolean, character, float, character array and object) into string representation during concatenation, java calls its one of the overloaded method `valueOf()` defined in `String` class. `valueOf()` method converts different types of data (primitive and object) into string whether the types are primitive or `Object`. For primitive type `valueOf()` returns a string that contains human readable equivalent of value and for `Object` `valueOf()` method calls `toString()` method.

`toString()` method is implemented by every class as it is defined on `Object`. `toString()` returns a string object that contains the human readable string that describe an object of class. `toString()` can be used in `print()` and `println()` statements and in concatenation expression and it can be override by any class when needed. Syntax:

`String toString();`

Different form of valueOf() method:

Syntax	Uses
<code>String valueOf(boolean b)</code>	To convert boolean value into String representation
<code>String valueOf(char c)</code>	To convert char value into String representation
<code>String valueOf(char [] c)</code>	To convert value of character array into String representation
<code>String valueOf(int a)</code>	To convert integer value into String representation
<code>String valueOf(float f)</code>	To convert floating point value into String representation
<code>String valueOf(double d)</code>	To convert value of double type into String representation
<code>String valueOf(long l)</code>	To convert value of double type into String representation
<code>String valueOf(Object ob)</code>	To convert object type into String representation

Modifying a String:

`String` object are immutable, so to modify a `String` it should be either copy into a `StringBuffer` or use a `String` method that constructs a new copy of the string with modification of original `String`. Some of such methods are:

substring():

This method is used to extract part of the `String` from whole string. It has following form:

`String substring(int startIndex)`

`String substring(int startIndex, int endIndex)`

In first form, `startIndex` specifies the position from which substring will begin. It return part of the string from `startIndex` to the end of invoking `String`.

In second form, startIndex specifies the position from which substring will begin and endIndex specifies the stopping point. It returns all the characters from startIndex up to the endIndex but not including the endIndex i.e. up to endIndex-1.

replace():

replace() method is used to replace some character with another character within the invoking String. It has two form:

String replace(char original, char replacement)

String replace(CharSequence original, CharSequence replacement)

In first form, original specifies the character to be replaced and replacement specified new character to be placed on original character. The resulting string is returned.

The second form, replace character sequence with another character sequence. Original specifies the character sequence to be replaced and replacement specified new character sequence to be placed on original character

trim():

this method removes any leading and trailing whitespace from String and returns the resulting String. Syntax:

String trim().

For example:

```
String s = "    Hello BCA    ".trim();
```

Changing the case of character within a String:

To change all the character of String into lowercase (small letter), toLowerCase() method is used and to change all the character of String into uppercase (capital letter), toUpperCase() is used. Syntax:

String toLowerCase()

String toUpperCase()

Both methods will return the string object that contains the upper- or lower-case equivalent of invoking string.

For example:

```
String s1 = "this is java class";
```

```
String upper = s1.toUpperCase();
```

```
String lower = s1.toLowerCase();
```

Joining String with delimiters:

Join () method is used to concatenate two or more string by separating each string with a delimiter such as comma, space and any other character. Syntax:

```
static String join (CharSequence delim, CharSequence str);
```

Here, delim specifies a delimiter to be used to separate any two string and str specifies a string in which delimiter will be used to separate any string.

For example:

```
String s1 = String.join(",", "Hello", "Java Program", "C program");
```

Some other String Methods:

boolean contains (CharSequence str):

This method returns true if the invoking object contains the string specified by str otherwise return false.

boolean contentEquals(CharSequence str):

This method returns true if the invoking object contains the same string as str otherwise return false.

boolean isEmpty():

This method returns true if the invoking String contains no any characters and has a length of zero

For example:

```
If(Stt1.isEmpty()){  
System.out.println("string is empty")
```

String Buffer:

String is immutable character sequence i.e. cannot be modified once declared. So, to supports a mutable String (which can be modified or changed) java provides StringBuffer class. StringBuffer class is used to create modifiable (mutable) string and also represents growable and writable character sequence. Therefore, by using StringBuffer, it is possible to append characters and substring at middle as well as at end. StringBuffer will automatically make room for such addition.

StringBuffer Constructor:

StringBuffer defines following constructor:

Constructor	Definition
StringBuffer ()	This is default constructor without parameter which reserves room for 16 characters without reallocation. Syntax: StringBuffer s1 = new StringBuffer();
StringBuffer(int size)	This version accepts an integer argument that explicitly sets the size of the buffer. Syntax: StringBuffer s1 = new StringBuffer(50);
StringBuffer(String str)	This version accepts a String as argument that sets the initial contents of the StringBuffer object and reserves room for 16 more character without reallocation. StringBuffer s3 = new StringBuffer("BCA third");
StringBuffer(CharSequence chars)	This version creates an object that contain the character sequence contained in chars and reserves room for 16 more characters.

When no any specific buffer length is specified then StringBuffer allocates room for 16 more character. When more room is allocated for few extra characters then StringBuffer reduces the number of reallocations that take place. Reallocation is costly process in term of time and can fragment more memory.

StringBuffer Methods:

Method Name	Function
length() and capacity()	Length method is used to find out current length (number of character) of StringBuffer. Capacity is used to find out total allocated capacity of buffer. The default capacity is 16 but if the number of character increase beyond current capacity it increases the capacity by (old capacity *2) +2 i.e. (16*2)+2 = 34. Syntax: int length () int capacity ()

ensureCapacity()	This method is used to pre-allocate room for certain number of character (set the size of buffer) after StringBuffer object have been constructed. Syntax Void ensureCapacity(int minCapacity); Here, minCapacity specifies the minimum size of the buffer.
append ()	This method is used to concatenate another string or any other type of data into string representation to the end of existence text in StringBuffer object. It has several forms: StringBuffer append (String str) StringBuffer append (int num) StringBuffer append (Object ob) Result is appended to the current StringBuffer object.
insert ()	This method inserts one string into another. It accepts all the values of the primitive type, String, objects and charSequence. First this method obtains the string representation of other value with which it is called with and then insert such value into invoking StringBuffer object. Syntax: StringBuffer insert (int index, String str) StringBuffer insert (int index, String chr) StringBuffer insert (int index, Object ob) Here index specifies the starting position from which string, character or object will be inserted into the invoking StringBuffer object.
reverse ()	This method is used to reverse the characters contained in StringBuffer object. Syntax: StringBuffer reverse ()
delete () and deleteCharAt()	delete () method of String Buffer class deletes the sequence of character from startIndex to endIndex of the invoking object. deleteCharAt() method deletes the character at the index specified by loc. It returns the resulting StringBuffer object. Syntax: StringBuffer delete (int startIndex, int endIndex) StringBuffer deleteCharAt(int loc)
replace ()	Replace () method is used to replace one set of characters with another set from specified index inside a StringBuffer object. Syntax: StringBuffer replace (int startIndex, int endIndex, String str) Here, startIndex specified the starting point from which replace will start up to the index specified by endIndex. The replacement string is passed on str. After this, StringBuffer object is returned.
char charAt(int index) and setCharAt()	charAt() method is used to obtain value of single character from StringBuffer. setCharAt() method is used to set the value of a character in a specified position within a StringBuffer. Syntax: char charAt(int index) void setCharAt(int index, char ch) for first form (charAt()) index specifies the position from which character is being obtained.

	For second form (void setCharAt ()), index specifies a position in which new character being placed and ch specifies the value of new character.
getChars()	<p>getChars() is used to copy a substring of a StringBuffer into an array. syntax:</p> <pre>void getChars(int sourceStart, int sourceEnd, char target[], int targetStart)</pre> <p>here, sourceStart specifies the index of beginning of substring and sourceEnd specified an index that is one past the end of the desired substring. target [] specifies the array that will receive the characters. The index within target at which the substring will be copied is passed in targetStart.</p>
substring()	<p>substring() is used to obtain portion of a StringBuffer. Syntax:</p> <pre>String substring(int startIndex)</pre> <pre>String substring(int startIndex, int endIndex)</pre> <p>Here, startIndex specifies the position from which part of the string will be extracted and endIndex specifies the position up to which character sequence will be extracted.</p> <p>The first form return the part of the string from startIndex. The second form return the part of the string from startIndex up to endIndex.</p>
indexOf(String str) or indexOf(String str, int StartIndex)	<p>Used for searching a particular string specified by str in invoking StringBuffer object. It has two forms:</p> <pre>int indexOf(String str)</pre> <pre>int indexOf(String str, int StartIndex)</pre> <p>the first form searches a specified str in StringBuffer and return the index of first occurrence of string if found otherwise return false.</p> <p>The second form searches a specified str in StringBuffer from the position specified in startIndex and return the index of first occurrence of String if found otherwise return false.</p>

Program for String Buffer methods:

```
package com.StringHandling;

public class StringBufferDemo {

    public static void main(String[] args) {

        //creating String Buffer

        StringBuffer s1 = new StringBuffer();

        StringBuffer s2 = new StringBuffer(20);//set the size of String buffer to 20

        StringBuffer s3 = new StringBuffer("This is String Handling");

        System.out.println("----example on length() and capacity()---");

        System.out.println("length find out total no. of character presents in StringBuffer");

        System.out.println("capacity find out capacity allocated for StringBuffer");

        System.out.println("length of s3 is "+s3.length()); //dinds out number of character

        System.out.println("length of s1 is "+s1.length());

        System.out.println("capacity of s1 is "+s1.capacity()+" character");//finds allocated capacity

        System.out.println("capacity of s3 is "+s3.capacity()+" character");

        System.out.println("capacity of s2 is "+s2.capacity()+" character");

        System.out.println("\n---example on append() ----");

        System.out.println("append adds specified text at end of StringBuffer");

        s3.append(" now, we are in StringBuffer "); //add specified text at end

        s2.append(s3);//add content of s3 to s2

        System.out.println("now s3 includes "+s3);

        System.out.println("now s2 includes: "+s2);

        System.out.println("\n---example on ensurecapacity() ---");

        System.out.println("after appending any extra text, capacity of s3 is "+s3.capacity());

        System.out.println("\n---example on insert()--");

        System.out.println("insert is use to insert text on particular position");

        s1.insert(0, "Java");//inserting String

        s1.insert(4, "is");// inserting character sequence

        s1.insert(6, true); //inserting boolean value

        s1.insert(10, 's');//inserting characer
```

```
char [] c = {'l','a','n','g','u','a','g','e'};
s1.insert(11, c); //inserting character array
s1.insert(19, s2); //inserting object in s1
System.out.println("now s1 contains: "+s1);
System.out.println("\n--- example on reverse() ---");
StringBuffer s4 = new StringBuffer("OOP in Java");
System.out.println("before reversing s4 it contains: "+s4);
System.out.println("after reversing s4, it contains: "+s4.reverse());
System.out.println("\n--example on delete() and deleteCharAt()---");
System.out.println("both methods are used to delete character from specified position");
StringBuffer s5 = new StringBuffer("Hello Third Semester");
s5.delete(0, 3); //delete from position 0 to (3-1) 2 position
System.out.println("after delete s5 contains "+s5);
s5.deleteCharAt(3); //delete character from specified index
System.out.println("after deleteCharAt(), s5 contains "+s5);
System.out.println("\n---example of replace---");
System.out.println("replace is used to replace some character with another character");
System.out.println("before replace s1 contains "+s1);
s1.replace(0, 4, "Web"); //replace java with web
System.out.println("after replace s1 contains: "+s1);
System.out.println("\n---example on charAt() and setChar()---");
System.out.println("charAt() is used to find specific character in specified position");
System.out.println("setChar() is used to set new character in specified position");
System.out.println("at position 5 s1 contains "+s1.charAt(5));
s1.setCharAt(6, 'a'); //setting character a at position 6 of s1
System.out.println("now s1 contains: "+s1);
System.out.println("--example on getChar()---");
System.out.println("getchar is used to copy certain character from StringBuffer to character array");
StringBuffer s6 = new StringBuffer("this is getchar() method");
char [] ch = new char[16];
```

```
s6.getChars(0, 8, ch, 0);

System.out.println("now array ch contains: ");

for(char s:ch){

    System.out.print(s);

}

System.out.println("\n\n--example on indexOf() and lastIndexOf()---");

System.out.println("these method are used to search particular string in StringBuffer");

System.out.println("position of substring getchar on s6? "+s6.indexOf("getchar"));

System.out.println("position of is after 4th position on s6? "+s6.indexOf("is", 4));

System.out.println("position of substring char from last on s6? "+s6.lastIndexOf("char"));

System.out.println("position of substring is from last on s6? "+s6.lastIndexOf("is", 15));

}

}
```

Output:

```
Output - Basic (run)
run:
---example on length() and capacity()---
length find out total no. of character presents in StringBuffer
capacity find out capacity allocated for StringBuffer
length of s3 is 23
length of s1 is 0
capacity of s1 is 16 character
capacity of s3 is 39 character
capacity of s2 is 20 character

---example on append() ---
append adds specified text at end of StringBuffer
now s3 includes This is String Handling now, we are in StringBuffer
now s2 includes: This is String Handling now, we are in StringBuffer

---example on ensureCapacity() ---
after appending any extra text, capacity of s3 is 80

---example on insert()--
insert is use to insert text on particular position
now s1 contains: JavaistrueslanguageThis is String Handling now, we are in StringBuffer

--- example on reverse() ---
before reversing s4 it contains: OOP in Java
after reversing s4, it contains: avaJ ni POO

--example on delete() and deleteCharAt()---
both method are used to delete character from specified position
after delete s5 contains lo Third Semester
after deleteCharAt(), s5 contains lo hird Semester

---example of replace---
replace is used to replace some character with another character
before replace s1 contains JavaistrueslanguageThis is String Handling now, we are in StringBuffer
after replace s1 contains: WebistrueslanguageThis is String Handling now, we are in StringBuffer
```

```
---example on charAt() and setChar()---
charAt() is used to find specific character in specified position
setChar() is used to set new character in specified position
at position 5 s1 contains t
now s1 contains: WebistaueslanguageThis is String Handling now, we are in StringBuffer
---example on getChar()---
getchar is used to copy certain character from StringBuffer to character array
now array ch contains:
this is

--example on indexOf() and lastIndexOf()---
these methods are used to search particular string in StringBuffer
position of substring getchar on s6? 8
position of is after 4th position on s6? 5
position of substring char from last on s6? 11
position of substring is from last on s6? 5
BUILD SUCCESSFUL (total time: 0 seconds)
```