String Class in Java

A string is a sequence of characters enclosed in double-quotes. Java implements strings as objects of the String class. A String object can be constructed in a number of ways:

1. From String literal
String s = "Computer"; //here's 's' is a variable

2. Using new keyword
String s = new String("Computer"); //here's 's' is an object

3. Using character array
char c[]={'C','o','m','p','u','t','e','r'};
String s=new String(c);

String class contains many methods to use and manipulate strings. Methods of String class are used with the help of String objects according to the given format:

<String object>.methodName(parameter)

How string is stored in the memory?

It is stored in a linear way where each character is allocated one cell with a particular index number. The index number in a string starts with 0 (Zero) and ends at a total no. of characters – 1.

For example, the String s will be like this in the memory:

0	1	2	3	4	5	6	7
C	O	m	p	u	t	e	r

The length of the above string is 8 and hence ending index is 7.

1. String Functions

1.1 length( )

Syntax: public int length()

Description : It returns the length of the String. The length of the String is equal to the number of 16 bit Unicode characters in the String.

For example, String str="Welcome";
int ln=str.length(); //it will return 7 (i.e. total 7 characters)

1.2 charAt( )

Syntax: public char charAt(int index)

Description: It returns the character at the specified index position. An index ranges from `0` to `length() - 1`.

Parameters: index - the index of the desired character

Throws: **StringIndexOutOfBoundsException** , if the index is not in the range `0` to `length()-1`.

For example, String str="Welcome";
char ch=str.charAt(5); //it will return 'm' i.e character at index position 5

1.3 equals( )

public boolean equals(Object anObject)

Compares this String to the specified object. Returns true if the object is equal to this String; that is, has the same length and the same characters in the same sequence. (It checks both the string for equal case too). This function is case-sensitive.

Parameters: anObject - the object to compare this String against
Returns: true if the Strings are equal; false otherwise.

```
String s="RAM", t="ram";
s.equals(t)              //false
s.equals("RAM")          //true
```

1.4 equalsIgnoreCase( )

public boolean equalsIgnoreCase(String anotherString)
Compares this String to another object. Returns true if the object is equal to this String; that is, has the same length and the same characters in the same sequence. Upper case characters are folded to lower case before they are compared. (Non-case sensitive)
Parameters: anotherString - the String to compare this String against
Returns: true if the Strings are equal, ignoring case; false otherwise.

```
String s="RAM", t="ram";
s.equalsIgnoreCase(t) //true
```

1.5 compareTo( )

public int compareTo(String anotherString)
Compares this String to another specified String. Returns an integer that is less than, equal to, or greater than zero. The integer's value depends on whether this String is less than, equal to, or greater than another String. It is a case-sensitive method.
Parameters: anotherString - the String to be compared

ASCII value of characters:
A-Z : 65 TO 90
a-z : 97 to 122
0-9 : 48 to 57
Space: 32
Special chars: 33 to 47

It compares two strings lexicographically (as per the ASCII values of the characters). This function gives the ASCII difference of the first dissimilar character in both the strings.

```
String s="ABC", t="abc";
int a=s.compareTo(t); //(ASCII diff. of 65-97) = -32
int b=t.compareTo(s); //(ASCII diff. of 97-65) = 32
```

i.e. Difference between ASCII ('A') and ASCII ('a')

```
String x="ALPHA", y="ALTO";
int c=x.compareTo(t);          //-4
```

i.e.Difference between ASCII ('P') and ASCII ('T')

If this method returns 0 that means both the strings are equal in all respect
If value>0 that means first string is greater than second string.
e.g. "ZOO" > "ELEPHANT"
If value<0 that means first string is smaller than the second string.
e.g. "AEROPLANE" < "aeroplane"

1.6 startsWith( )

1.6.1 public boolean startsWith(String prefix, int index)

Determines whether this String starts with some prefix from the given index position.
Parameters: prefix - the prefix
index - where to begin looking in the String
Returns: true if the String starts with the specified prefix; false otherwise.

1.6.2 public boolean startsWith(String prefix)

Determines whether this String starts with some prefix.
Parameters: prefix - the prefix
Returns: true if the String starts with the specified prefix; false otherwise.

```
String str=”Today is Monday”;
str.startsWith(“Today”);        //true
str.startsWith(“T”); //true
str.startsWith(‘M’); //false
```

Both the above functions are case-sensitive

1.7 endsWith( )

public boolean endsWith(String suffix)

Determines whether the String ends with some suffix.
Parameters: suffix - the suffix
Returns: true if the String ends with the specified suffix; false otherwise.

```
String str=”Today is Monday”;
str.endsWith(“day”);   //true
str.endsWith(“Y”); //false
str.endsWith(‘y); //true
```

The above function is case-sensitive

1.8 indexOf ( )

1.8.1 public int indexOf(char ch) – returns the index position of the first occurrence of the given character in the string.

Returns the index within this String of the first occurrence of the specified character. This method returns -1 if the index is not found. The function is case-sensitive.
Parameters: ch - the character to search for

```
String s=”WELCOME”;
s.indexOf(‘E’) // 1 (index position of the letter ‘E’, the first occurrence)
s.indexOf(‘e’) // -1 (means not present)
```

1.8.2 public int indexOf(char ch, int fromIndex) – here the search starts from the specified index position

Returns the index within this String of the first occurrence of the specified character, starting the search at fromIndex. This method returns -1 if the index is not found.
Parameters: ch - the character to search for
fromIndex - the index to start the search from

```
String s=”WELCOME”;
s.indexOf(‘E’,2) // 6 (index position of the letter ‘E’, the first occurrence after index 2)
 here WE will not be considered and the search will start from index position 2.
```

1.8.3 public int indexOf(String str)

Returns the index within this String of the first occurrence of the specified substring. This method returns -1 if the index is not found.
Parameters: str - the substring to search for

1.8.4 public int indexOf(String str, int fromIndex)

Returns the index within this String of the first occurrence of the specified substring. The search is started at fromIndex. This method returns -1 if the index is not found.
Parameters: str - the substring to search for
fromIndex - the index to start the search from

1.9 lastIndexOf ( )

1.9.1 public int lastIndexOf(char ch)

Returns the index within this String of the last occurrence of the specified character. The String is searched backwards starting at the last character. This method returns -1 if the index is not found.
Parameters: ch - the character to search for

```
String s="WELCOME";
s.lastIndexOf('E') // 6 (index position of the letter 'E', the last occurrence)
```

1.9.2 public int lastIndexOf(String str)

Returns the index within this String of the last occurrence of the specified substring. The String is searched backwards. This method returns -1 if the index is not found.
Parameters: str - the substring to search for

Note: Both indexOf( ) and lastIndexOf( ) are case sensitive. Also, if there is only one occurrence of the searched letter/word, both the methods will return same index value.

1.10 substring( )

1.10.1 public String substring(int beginIndex)

Returns the substring of this String. The substring is specified by a beginIndex (inclusive) and the end of the string.
Parameters: beginIndex - the beginning index, inclusive

```
String str= "WELCOME";
String res=str.substring(3); //COME
String x=res.substring(2); //ME
```

1.10.2 public String substring(int beginIndex, int endIndex)

Returns the substring of a String. The substring is specified by a beginIndex (inclusive) and an endIndex (exclusive).
Parameters: beginIndex - the beginning index, inclusive
endIndex - the ending index, exclusive
Throws: StringIndexOutOfBoundsException

If the beginIndex or the endIndex is out of range.

```
String str= ”WELCOME”;
String res=str.substring(2,5); //LCO (index position 2 to 4)
String p= “Today is Holiday”; (0-15)
System.out.println(p.substring(5)); (5-15) // “ is Holiday”
System.out.println(p.substring(0,5)); (0-4) // “Toda”
System.out.println(p.substring(5,9)); (5-8) // “ is ”
```

1.11 concat ( )

public String concat(String str)
Concatenates the specified string to the end of this String.
Parameters: str - the String which is concatenated to the end of this String

1.12 replace( )

public String replace(char oldChar, char newChar)
Converts this String by replacing all occurences of oldChar with newChar.
Parameters: oldChar - the old character
newChar - the new character

public String replace(String oldStr, String newStr)
Converts this String by replacing all occurences of oldStr with newStr.
Parameters: oldStr - the old string
newStr - the new string

1.13 toLowerCase

public String toLowerCase()
Converts all of the characters in this String to lower case.
Returns: the String, converted to lowercase.

1.14 toUpperCase

public String toUpperCase()
Converts all of the characters in this String to upper case.
Returns: the String, converted to uppercase.

1.15 trim

public String trim()
Trims leading and trailing white space from this String.
Returns: the String, with white space removed.

1.16 toString

public String to String()
Converts this String to a String.
Returns: the String itself.
Overrides: toString in class Object

1.17 toCharArray

public char[] toCharArray()

Converts this String to a character array. This creates a new array.
Returns: an array of characters.

StringBuffer Class in Java

With an object of String class we can only create strings of fixed length. They cannot be modified as per requirement. However Java provides StringBuffer class that allows us to create modifiable strings. An object of StringBuffer can be created in the following way:

```
StringBuffer str=new StringBuffer("Welcome to java");
```

The above statement creates an object str which is initialized with "Welcome to Java Programming".

Note : We cannot enter values (String constant) to StringBuffer object directly from the console. In such cases the value need to be stored in a String class object or a variable first and then need to be transferred using the code given below:

```
String s = <BufferedReader object>.readLine();
Or
String s = <Scanner object>.nextLine();
StringBuffer obj = new StringBuffer(s);
```

2. StringBuffer Functions

2.1. length

public int length()

This function returns the length of the string stored in StringBuffer object.

```
For example :
int len=str.length();
System.out.println(len);        //15
```

2.2. charAt

public char charAt(int)

This function returns the character at the given index position of the StringBuffer object.

```
For example :
char c=str.charAt(3);
System.out.println(c);          //c
```

2.3. setCharAt

public void setCharAt(int, char)

This function sets the character at the given index position of the StringBuffer object.

```
For example :
str.setCharAt(11, 'J');
System.out.println(str);        //Welcome to Java
```

2.4. reverse

public void reverse()

This function reverse the content of the StringBuffer object.

For example :

```
StringBuffer s=new StringBuffer(“Today”);
str.reverse();
System.out.println(s);          //yadoT
```

2.5. append

public void append(String)

This function adds another string at the end of the current StringBuffer object.

For example :

```
str.append(“ Programming”);
System.out.println(str);        //Welcome to Java Programming
```

2.6. insert

public void insert(String, int)

This function inserts a new string at the given index position of the StringBuffer object.

For example :

```
str.insert(“ the”, 11);
System.out.println(str);        // Welcome to the Java Programming
```

2.7. delete

public void delete(int, int)

This function deletes a set of characters between the two index positions of the StringBuffer object.

For example :

```
StringBuffer s=new StringBuffer(“Today”);
s.delete(0,1);
System.out.println(s);          //day
```

2.8. deleteCharAt

public void deleteCharAt(int)

This function deletes one character at the given index position of the StringBuffer object.

For example :

```
StringBuffer s=new StringBuffer(“Today”);
s.delete(1);
System.out.println(s);          //Tday
```

****