UNIT 3

String Handling

String Constructors:

1. Default constructor or empty constructor

String S = new String ();

2. Create a string object initialized by an array of characters.

String (char ch[]);

```
char ch[ ] = { 'C','S','E'};
String Str = new String (ch);  // Str contains “CSE”
```

3. Specify a sub range of a character array as an initializer.

String (char charArray[], int startIndex, int numChars);

```
char ch [ ] = { 'B', 'C', 'S', 'C', 'S', 'E'};
String Str = new String (ch, 3, 3);   // Str contains “CSE”
```

NOTE: Java uses Unicode character set with 16 bits to represent each character. But the typical format for string on the Internet uses arrays of 8-bit bytes constructed from the ASCII character set. The String class provides constructors that initialize a String when given a byte array.

String (byte asciiChars[ ])

```
String (byte asciiChars[ ], int startIndex, int numChars)
byte AChar[]={83,73,85,'A'};
String S = new String(AChar);
S = new String(AChar,0,2);
```

4. Initialize a string object by using another string object.

String (String Str)

String S1 = new String ("Welcome"); String S2 = new String (S1);

5. String Concatenation: Java does not allow operators to be applied to String objects except + operator.

String S1 = "Welcome To MCET"; String S2 = "By CSE";

String S3 = S1 + S2;

String S3 = S1 + "By CSE";

Note: When you concatenate strings with other data type, the compiler will convert an operand to its string equivalent whenever the other operand of the **+** is an instance of String.

char grade = 'S'; String Str = "He got" + grade + "grade";

Note: String S = "Total is:" + 40 + 50;

System.out.println(S); **Output:** Total is: 4050

S = "Total is:" + (40 + 50);

System.out.println(S); **Output:** Total is: 90

System.out.println(10+10+ ".Rs"); **Output:**20.Rs

6. Find the length of a string.

int length()

String Str = "Welcome To India"; int Len = Str.length ();

7. String conversion:

Convert the value of an object into string.

String toString ( )

Character extraction functions:

1. char charAt(int index); → Extract a single character from a string. Where index (starting from 0) specifies the location of the character being extract.

String S = “WELCOME”; char Ch = S.charAt(3); **Output:** C

char Ch = “JAVA”.charAt (1); **Output:** A

2. void getChars(int start, int end, char target[ ], int targetStart);

Where **start** specifies the index of the beginning of the substring and **end** specifies the index of the end of the desired substring. The substring contains the character from **start** through **end-1**. The array **target[ ]** receive the substring. The index within **target** at which the substring will be copied passed in **targetStart**.

String S = “WELCOME TO INDIA”; int start = 11, end = 16;

char Temp[ ] = new char [end-start];

S.getChars(start, end, Temp,0); // extract start to end-1

System.out.println(Temp); **Output:** INDIA

3. getBytes():

Convert String into bytes. Mostly useful when you are exporting a String value into an environment that does not support 16-bit Unicode characters. Since most Internet protocols and text file formats use 8-bit ASCII for all text interchange.

byte [ ] getBytes ( )

```
byte B[] = new byte[20];

String Str = new String("Welcome");

B=Str.getBytes();

for(int i=0;i<B.length;i++)

    System.out.println((char)B[i]);
```

4. toCharArray():

Convert all the characters in a String object into a character array. [Same as getChars(), but it is convenient one.]

char[ ] toCharArray()

char B[] = new char[20];

String Str = new String("Welcome");

B=Str.toCharArray();

String Comparison

1. Test whether two Strings are equal or not equal.

boolean equals(String S); // With Case Sensitive

boolean equalsIgnoreCase (String S); // Without Case Sensitive

Where, S is the String object being compared with the invoking String object. It returns true if the strings contains the same characters in the same order. Otherwise returns false.

String S1 = new String ("MCET"); String S2 = new String ("Mcet");

boolean b1 = S1.equals(S2); // false

boolean b1 = S1.equalsIgnoreCase(S2); // true

2. regionMatches():

This method compares a specific region inside a String with another specific region in another String.

boolean regionMatches(int startIndex, String Str2, int str2StartIndex, int numChars);

boolean regionMatches(boolean ignoreCase, int startIndex,

String Str2, int str2StartIndex, int numChars);

// if **ignoreCase** is true, then case sensitive is ignored.

Where **startIndex** specifies at which the region begins with the invoking String object. The String being compared is specified by **Str2**. The index at which the comparison will

start within Str2 is specified by **str2StartIndex**. The length of the substring being compared is passed in **numChars**.

boolean B1 = “welcome”**.**regionMatches(0, "wel", 0, 3); // true

boolean B2 = "WELCOME"**.**regionMatches(true, 0,"wel", 0, 3); // true

3. startsWith():

Check whether a given string starts with a specified string or not (with case sensitive).

boolean startsWith(String Str)

boolean startsWith(String Str, int startIndex)

Where startIndex specifies the index into the invoking string at which point the search will began.

boolean B1 = “TestPack”**.**startsWith(“Test”);

boolean B2 = “welcome”**.**startsWith(“come”, 3);

4. endsWith ():

Check whether a given string ends with a specified string or not. (with case sensitive)

boolean endsWith(String Str)

boolean B ="Test.java".endsWith("java");

5. compareTo(): Used in sorting applications.

int compareTo(String Str);

int compareToIgnoreCase(String Str);

If the return Value is < than Zero, then the Invoking string < than the argument String Str.

If the return Value is > than Zero, then the Invoking string > than the argument String Str.

If the return Value is Zero, then the two strings are equal.

Strings Searching Methods:

1. indexOf( ): Search for the 1st occurrence of the character or sub string.

int indexOf(char Ch);

int indexOf(String str);

Here Ch is the character being hunted and str specifies the substring.

```
String str = new String("welcome");
int firstloc =  str.indexOf('e');
int firstloc =  str.indexOf("come");
```

2. lastIndexOf( ): Search for the last occurrence of the character or sub string.

int lastIndexOf(char Ch);

int lastIndexOf(String str);

```
String str = new String("welcome welcome");
int lastloc =  str.lastIndexOf('e');
int lastloc =  str.lastIndexOf("come");
```

Note: You can also specify the starting index for the search.

int indexOf(char Ch, int startIndex); int indexOf(String str, int startIndex);

int lastIndexOf(char Ch, int startIndex); int lastIndexOf(String str, int startIndex);

For indexOf (), the search runs from startIndex to the end of the string. For lastIndexOf() the search runs form startIndex to zero. For all the above cases if the character or substring is not found, then the function returns -1.

Modifying a String:

1. substring(): Extract a substring from another string.

String substring (int startIndex);

// Extract a substring from startIndex to end of the string.

String substring (int startIndex, int endIndex);

// Extract a substring from startIndex to endIndex-1 .

String str = new String("welcome welcome");

String a =str.substring(8); // welcome

String str = new String("welcome welcome");

String a =str.substring(11,15); // come

2. replace(): Replaces all occurrences of one character in the invoking string with another character.

String replace (char original, char replace);

String str = new String("welcome to Java");

String S=str.replace ('a', 'e');

3. concat():

String concat (String S2);

String s1 = new String ("welcome ");

String s2 ="To Dr.MCET";

String s3=s1.concat(s2);

4. trim():

Returns a copy of the invoking string from which any leading and trailing white space has been removed. [Useful when you process user commands]

String trim ();

String str = " \n\t Welcome to Java";

System.out.println(Str.trim());

Changing the case of characters:

1. String toLowerCase(); // Convert Upper Case into Lower Case

2. String toUpperCase(); // Convert Lower Case into Upper Case

Data conversion using valueOf ():

The method **valueOf()** is called when a string representation of some other type of data is needed. You can call this method directly with any data type and get a reasonable String representation.

static String valueOf(double num)

static String valueOf(long num)

static String valueOf(char Chars[ ])

int N = 123;

char ch[]={'G', 'O', 'D',' ', 'i', 's', ' '};

String S = String.valueOf(ch);

System.out.println(S+1); **Output:** GOD is 1

System.out.println(String.valueOf(N)+123); **Output:** 123123

StringBuffer class

String represents fixed lengths. Whereas StringBuffer represents growable and writable character sequences. i.e, when you are adding more characters to the StringBuffer, it will automatically allocate memory for newly added characters.

Constructors:

StringBuffer( ) // Default constructor. It reserves room for 16 characters without reallocation.

StringBuffer(int size) // Explicitly sets the size of the string buffer.

StringBuffer(String Str) // Initialize the StringBuffer with **Str** and reserves room for 16 more characters without reallocation.

int length( )

Return the current length of the string buffer.

int capacity( )

Return the total allocated capacity.

StringBuffer S = new StringBuffer();

System.out.println(S.length()); **OUTPUT: 0**

System.out.println(S.capacity()); **OUTPUT: 16**

StringBuffer S = new StringBuffer("welcome to MCET");

System.out.println(S.length()); **OUTPUT: 15**

System.out.println(S.capacity()); **OUTPUT: 31**

StringBuffer S = new StringBuffer(30);

System.out.println(S.length()); **OUTPUT: 0**

System.out.println(S.capacity()); **OUTPUT: 30**

void ensureCapacity(int capacity)

Pre allocate room for a certain number of characters after a StringBuffer has been constructed. Here, capacity specifies the size of the buffer.

StringBuffer S = new StringBuffer(30);

System.out.println(S.capacity()); **OUTPUT: 30**

S.ensureCapacity(100);

System.out.println(S.capacity()); **OUTPUT: 100**

void setLength(int len)

To sets the length of the buffer. Here **len** specify the length of the buffer (**len** must be non-negative). If the value of **len** less than the current length returned by the **length()**, then the characters stored beyond the new length will be lost.

StringBuffer S = new StringBuffer("Welcome To DrMCET");

S.setLength(7);

System.out.println(S); **OUTPUT:** Welcome

char charAt(int Loc)

Returns a single character specified by the location **Loc**.

void setCharAt(int Loc, char ch)

Set a new character **ch** specified by the location **Loc** within the buffer. **Loc** must be non-negative and must not specify a location beyond the end of the buffer.

StringBuffer S = new StringBuffer("MCCT");

System.out.println(S.charAt(1)); **OUTPUT:** C

S.setCharAt(2,'E');

System.out.println(S); **OUTPUT:** MCET

void getChars(int sourceStart, int sourceEnd, char target[], int targerStart)

The substring contains the characters from **sourceStart** through and **sourceEnd-1**. The array **target** will receive the substring. The index within target at which the substring will be copied is specified by **targetStart**.

char target[] = new char[20];

StringBuffer S = new StringBuffer("Welcome to Dr.MCET");

S.getChars(0, 7, target,3) ;

System.out.println(target); **OUTPUT:** ---Welcome

StringBuffer append(String Str)

StringBuffer append(int Value)

Concatenate the string representation of any other type of data to the end of the invoking StringBuffer object. For the append function the arguments may be StringBuffer, char, char[], int, long, float and double.

StringBuffer S = new StringBuffer("God is ");

S=S.append("sin(").append(90).append(')');

System.out.println(S); **OUTPUT:** God is sin(90)

StringBuffer insert(int Loc, String Str)

StringBuffer insert(int Loc, char ch)

StringBuffer insert(int Loc, double value)

Here Loc specify the location at which point the string will be inserted into the invoking StringBuffer object.

StringBuffer S = new StringBuffer("God is ");

S=S.insert(7,1); System.out.println(S); **OUTPUT:** God is 1

S = S.insert(2, l); System.out.println(S); **OUTPUT:** Gold is 1

StringBuffer reverse()

Reverse the characters within a StringBuffer.

StringBuffer S = new StringBuffer("GST");

S=S.reverse();

System.out.println(S); **OUTPUT:** TSG

StringBuffer delete(int startIndex, int endIndex)

Delete the character sequence form **startIndex** to **endIndex-1**.

StringBuffer deleteCharAt(int Loc)

Delete the character specified by the index **Loc.**

StringBuffer S = new StringBuffer("welcome");

System.out.println(S.delete(4,7)); **OUTPUT:** wel

System.out.println(S.deleteCharAt(2)); **OUTPUT:** we

StringBuffer replace(int startIndex, int endIndex, String str)

Replaces one set of characters with another set inside a StringBuffer object. The substring at **startIndex** to **endIndex-1** is replaced.

StringBuffer S = new StringBuffer("welcome to MCET");

S.replace(12,15,"Java"); System.out.println(S); **O/P:** welcome to Java

Substring

String substring (int startIndex)

Return the substring that starts at **starIndex** and runs to the end of the invoking StringBuffer object.

StringBuffer S = new StringBuffer("welcome");

String Str=S.substring(3);

System.out.println(Str); **OUTPUT:** come

String substring (int startIndex, int endIndex)

Return the substring that starts at **starIndex** and runs through **endIndex-1** of the invoking StringBuffer object.

StringBuffer S = new StringBuffer("welcome");

String Str=S.substring(0,3);

System.out.println(Str); **OUTPUT:** wel

No.	String	StringBuffer
1)	String class is immutable.	StringBuffer class is mutable.
2)	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.
3)	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.

Check the output of the following statements:

1. StringBuffer sb = new StringBuffer(“welcome”);

   What is length & capacity?

   Sb.append(“welcome”): sb.append(“welcometo”);

   What is length & capacity now?

   Sb.append(“welcome”);

   What is length & capacity now?

2. StringBuffer S = new StringBuffer(“welcome”);

   S = S + S;

   S = S + “welcome”;

3. String St = S + “welcome”;

   String St = “welcome” + S;

4. String S=” ”; String S;

   StringBuffer S = “”; StringBuffer S;