GROUP – 8 ASSIGNMENT

- **String ()**

We can declare a string using the **new** keyword. It creates a new string object. The variable defined would point to this string object placed in the non-pool memory, while the literal would be stored in the string pool. The syntax for defining is given below:

String <variable name> = new String (“<string literal name”>);

Example:

```
String st = new String ("Hello World");
System.out.println (st);
```

```
Output:

Hello World!
```

- **String (value: String)**

This is a shorthand notion used for creation of a string, and it helps in more memory efficiency of the program. The syntax of defining is given below:

String <variable name> = “<string literal name>”;

Example:

```
String st = "Hello World!";
System.out.println (st);
```

```
Output:

Hello World!
```

- **String (value: char[])**

Strings in Java are essentially a sequence of characters. Therefore, we can define a String using an array of characters. This is shown below:

Example:

```
char ch[] = {'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '!'};
String st = new String (ch);
System.out.println (st);
```

```
Output:

Hello World!
```

- **charAt(index: int) : char**

This method helps the user obtain the character at a particular index in the string. It is often used in programming using strings. It returns a value of character data type. It is used in the following way:

<string variable name>**.charAt**(<index number>);

Example:

```
String st = "Hello World";
char ch1 = st.charAt(1);
char ch2 = st.charAt(7);
System.out.println ("Character at index 1: " + ch1);
System.out.println ("Character at index 7: " + ch2);
```

```
Output:
Character at index 1: e
Character at index 7: o
```

- **compareTo(anotherString : String): int**

This method does the lexographic comparison between two strings. It compares the two strings character-by-character.

If the first string comes before second string in terms of dictionary order, then the method returns a negative number. If the first string comes after second string in terms of dictionary order, then the method returns a positive number. This number is equal to the difference in the ASCII values of first characters which differ between the two strings. If the two strings are equal, then it returns zero.

Given below is the syntax to use the method:

<first string variable>**.compareTo**(<second string variable>);

Example:

```
String st = "Delhi";
String st1 = "Agra";
String st2 = "Mumbai";
String st3 = "Delhi";
System.out.println (st.compareTo(st1));
System.out.println (st.compareTo(st2));
System.out.println (st.compareTo(st3));
```

```
Output:
3
-9
0
```

Note: **compareTo** is case senstitive. Thus, it compares the difference in ASCII values of characters. ‘a’ and ‘A’ would be considered different when using this method.

- **compareToIgnoreCase (anotherString: String): int**

This method does the same work as **compareTo** method. However, it ignores the case while comparing the two Strings. Therefore, it does not give the difference in ASCII values as output. Rather, it simply gives the difference in the dictionary position of characters as the output.

Given below is the syntax to use the method:

<first string variable>**.compareToIgnoreCase**(<second string variable>);

Example:

```
String st = "Delhi";
String st1 = "agra";
String st2 = "mumbai";
String st3 = "delhi";
System.out.println (st.compareToIgnoreCase(st1));
System.out.println (st.compareToIgnoreCase(st2));
System.out.println (st.compareToIgnoreCase(st3));
```

```
Output:
3
-9
0
```

- **concat(anotherString : String) : String**

This method is used to concatenate (join) two strings. Concatenation of two strings can be done by using the '+' operator as well. Given below is the syntax to do concatenation:

<first string name>.concat (<second string name>); OR

<first string variable> + <second string variable>;

Example:

```
String st1 = "Hello ";
String st2 = "World";
String st3 = st1.concat(st2);
String st4 = st1 + st2;
System.out.println(st3);
System.out.println(st4);
```

```
Output:
Hello World
Hello World
```

- **endsWith (suffix: String): boolean**

This method checks whether a String ends with a given suffix or not. It returns a boolean type value: true if the string ends with the given suffix, and false otherwise. Given below is the syntax:

<string variable name>**.endsWith** (<suffix>);

Example:

```
String st = "BITS Goa";
System.out.println(st.endsWith("a"));
System.out.println(st.endsWith("Goa"));
```

```
System.out.println(st.endsWith("BITS"));
```

```
Output:
true
true
false
```

- **equals (another String: String): boolean**

This method compares the contents of two strings, and returns a boolean type value. If all characters of two strings match, then it returns true, and it returns false otherwise. The comparison is case sensitive. Given below is the syntax for this method:

<first string variable>**.equals**(<seconds string variable>);

Example:

```
String st1 = "Hello";
String st2 = "Hello";
String st3 = "hello";
System.out.println(st1.equals(st2));
System.out.println(st1.equals(st3));
```

```
Output:
true
false
```

- **equalsIgnoreCase (another String: String): boolean**

This methodalso performs comparison between two strings. However it ignores the case of each character during comparison, thus it is not case sensitive. Given below is the syntax for this method:

<first string variable>**.equalsIgnoreCase**(<seconds string variable>);

Example:

```
String st1 = "Hello";
String st2 = "Hello";
String st3 = "hello";
System.out.println(st1.equalsIgnoreCase(st2));
System.out.println(st1.equalsIgnoreCase(st3));
```

```
Output:
true
true
```

- **indexOf(ch: int): int**

This method returns the index of a particular character as it first appears in a given string. If the character is not present, then it returns -1 as output. The syntax is given below:

<string variable>**.indexOf**(‘<character value>’);

Example:

```
String st1 = "Hello Everyone";
System.out.println(st1.indexOf('o'));
System.out.println(st1.indexOf('l'));
System.out.println(st1.indexOf('z'));
```

```
Output:
4
2
-1
```

- **indexOf(ch: int, fromIndex : int): int**

This method returns the index of a character in the string, from a particular index. The syntax is given below:

<string variable>**.indexOf**(‘<character value>’, <fromIndex>);

Example:

```
String st1 = "Hello Everyone";
System.out.println(st1.indexOf('e', 10));
System.out.println(st1.indexOf('o', 6));
```

```
Output:
13
11
```

- **indexOf(str: String): int**

This method returns the index for a particular Substring as it first appears in a given string. The syntax is given below:

<string variable>**.indexOf**(“<Substring>”);

Example:

```
String st1 = "How is this?";
System.out.println(st1.indexOf("is"));
```

```
Output:
4
```

- **indexOf(str: String, fromIndex : int): int**

This method returns the index of a substring, from a particular index. The syntax is given below:

<string variable>**.indexOf**(‘<Substring>’, <fromIndex>);

Example:

```
String st1 = "How is this?";
System.out.println(st1.indexOf("is", 6));
```

```
Output:
9
```

- **intern(): string**

This method returns the *canonical* (shorthand notation) of the string. When this method is executed, it checks the list or pool created by the String class for the exact string. If this string is present then the canonical string is returned from that list, otherwise the string object is added to the list and the reference is returned. The syntax is given below:

<string variable>**.intern()**;

Example:

```
// The use of intern method
String s1 = "Object Oriented Programming"; // The shorthand notation of string.
String s2 = s1.intern();
System.out.println(s2);
// To check whether the same string reference is returned, or new object is created
System.out.println("Is s2 from the same String pool as s1?");
System.out.println(s1 == s2); //If true, implies reference is passed.
```

```
Output:
Object Oriented Programming
Is s2 from the same String pool as s1?
true
```

- **regionMatches(toffset: int, other: String, ooffset: int, len: int): Boolean**

This method is used to check if 2 regions or sub-regions of the strings are matching or not. It returns a boolean type value. The method is case sensitive. There is also a parameter called ignoreCase, when set to true, will make the method case insensitive. The syntax is given below:

<first string variable>**.regionMatches**(<toffset>, <other string variable>, <ooffset>, <len>);

Parameters:

toffset: The starting point of first string region. We will start counting from zero, just like arrays index.

other: The other string, from which we have to compare.

ooffset: The starting point of the second/other string.

len: the length of the substring to be compared.

```
// The use of regionMatches method
String s1 = "Object Oriented Programming"; // The shorthand notation of string.
String s2 = "C Programming"; // *note the whitespace is also counted for determining parameters
// Lets compare the last words for both the strings
System.out.println("Are the last words equal?");
System.out.println(s1.regionMatches(16, s2, 2, 11));
System.out.println("Are the first 10 letters equal?");
System.out.println( s1.regionMatches(0, s2,0, 10));
```

Output:

```
Are the last words equal?
true
Are the first 10 letters equal?
false
```

- **length(): int**

This method returns the number of characters in the string. The syntax is given below:

<string variable>**.length()**;

Example:

```
String st1 = "Object Oriented Programming";
System.out.println(st1.length());
```

Output:

```
27
```

- **replace(oldChar: char, newChar: char): String**

This method returns a string which is derived from previous string, where all the old character values are replaced by new characters entered.

Syntax:

<string variable>**.replace**("<old Char>", "<new Char>");

Example:

```
// The use of replace()
String s1 = "Race";
String s2 = s1.replace("R","P" );
System.out.println(s1 + " " + s2);
```

Output:

```
Race Pace
```

- **startsWith(prefix: String): Boolean**

This method checks whether the given String starts begins with a particular substring or not. It returns a boolean type value.

Syntax:

<string variable>**.startsWith**("<prefix>");

Example:

```
String st1 = "Object Oriented Programming";
System.out.println(st1.startsWith("Object"));
```

Output:

```
true
```

- **substring(beginIndex: int): String**

This method returns the substring from a specified starting index(inclusive), to the end of the string. It returns a string type value.

Syntax:

<string variable>.**subString**(<beginIndex>);

Example:

```
String st1 = "Object Oriented Programming";
System.out.println(st1.substring(16));
```

```
Output:

Programming
```

- **substring(beginIndex: int, endIndex: int): String**

This method returns the substring between two specified index positions of a given string.

Syntax:

<String variable>.**substring**(<beginIndex>, <endIndex>);

Example:

```
String st1 = "Object Oriented Programming";
System.out.println(st1.substring(7, 15));
```

```
Output:

Oriented
```

- **toCharArray(): char[]**

This method returns the given string in form of a character array.

Syntax:

<String variable>.toCharArray();

Example:

```
// The use of toCharArray()
String s1 = "code";
char[] s2 = s1.toCharArray();
for(char element : s2){
    System.out.println(element);
}
```

```
Output:
```

```
c
o
d
e
```

- **toLowerCase(): String**

This method converts all the characters of a given string into lower case, and returns the string with each character in lower case.

Syntax:

<String variable>**.toLowerCase()**

Example:

```
// The use of toLowerCase()
String s1 = "EnGiNeEr";
String s2 = s1.toLowerCase();
System.out.println(s2);
```

Output:

```
engineer
```

- **toString(): String**

This method returns the String representation of the object, in the format, ClassName, then @, then the unsigned hexadecimal representation of the reference. Note, if object is directly printed, the toString() method will be implemented internally.

Syntax:

<Object name>**.toString()**;

Example:

```
class Oop{
    public int noofstudents;
    public int nooflectures;
    void display(){
        System.out.println(noofstudents);
        System.out.println(nooflectures);
    }

}
```

```
Oop obj = new Oop();
System.out.println(obj.toString());
System.out.println(obj); // Same output shows that toString() is implemented internally
```

Output:

```
Oop@4617c264
Oop@4617c264
```