STRING MANIPULATIONS

String Manipulations

In C and C++ applications, to perform String operations, C and C++ programming languages have provided some predefined library in the form of the functions.

In Java, to perform String operations JAVA has provided the following predefined classes.

- java.lang.String
- java.lang.StringBuffer
- java.lang.StringBuilder
- java.util.StringTokenizer

Q) What is the difference between *String* and *StringBuffer?*

- String class objects are immutable objects, where Immutable objects are not allowing modifications over their content, if we are trying to perform modifications over immutable object content then operations are allowed , but, the resultant data will not be stored back in original object, where the resultant data will be stored by creating new object.

- StringBuffer objects are mutable objects, they are able to allow modifications on their content.

Q) What are the differences between *StringBuffer* and *StringBuilder?*

- StringBuffer class was introduced in JDK1.0 version.
StringBuilder class was introduced in JDK5.0 version.

- StringBuffer is synchronized.
StringBuilder is not synchronized.

- Almost all the methods are synchronized in StringBuffer.
No method is synchronized in StringBuilder.

- StringBuffer is able to allow only one thread to access data.
StringBuilder is able to allow more than one thread to access data.

- StringBuffer is following sequential execution.
StringBuilder is following parallel execution.

- StringBuffer will increase execution time.
StringBuilder will reduce execution time.

- StringBuffer will reduce application performance.
StringBuilder will increase application performance.

- StringBuffer is able to give guarantee for Data Consistency.

StringBuilder is unable to give guarantee for data consistency.

- StringBuffer is threadsafe.

StringBuilder is not thread safe.

Q) What is String tokenization and how it is possible to perform String Tokenization in Java Applications?

- The process of dividing a string into no of tokens is called as String Tokenization.
- To perform String Tokenization JAVA has provided a predefined class in the form of java.util.StringTokenizer.

To perform String Tokenization in java applications we have to use the following steps.

Create StringTokenizer Class Object:

To create StringTokenizer class object we have to use the following constructor.

public StringTokenizer(String data)

EX: StringTokenizer st=new StringTokenizer("Durga Software Solutions");

When JVM encounter the above instruction, JVM will take the provided String, JVM will divide the provided String into no of tokens and JVM will store the generated tokens by creating object for StringTokenizer class.

NOTE: When StringTokenizer class object is created with the no of tokens then a pointer or cursor will be created before the first token.

To get no of tokens which are existed in StringTokenizer class object we have to use the following method.
public int countTokens()

Retrieve Tokens from StringTokenizer Object:

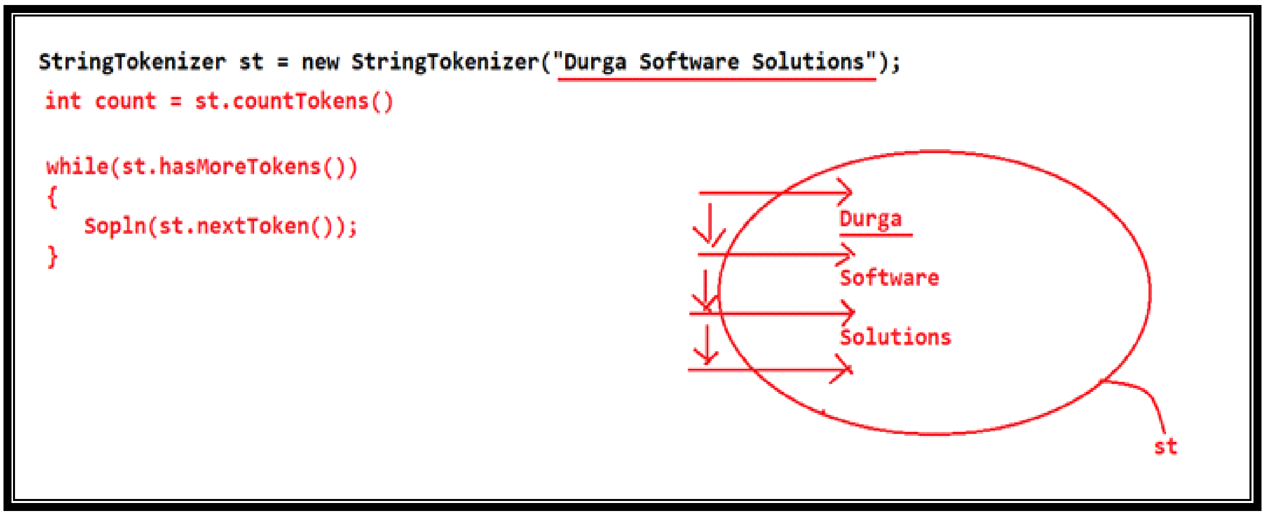
To retrieve tokens from StringTokenizer object we have to use the following steps for each and every token.

- Check whether more tokens are available or not from the current cursor position by using the following method. public boolean hasMoreTokens()
- It will return true value if at least next token is existed.
- It will return false value if no more tokens are existed.

- If atleast next token is available then read next token and move cursor to next position by using the following method.
  public String nextToken()

EX:

```
import java.util.*;
class Test
{
  public static void main(String[] args)
  {
    StringTokenizer st=new StringTokenizer("Durga Software Solutions");
    System.out.println("No Of Tokens:"+st.countTokens());
    while(st.hasMoreTokens())
    {
      System.out.println(st.nextToken());
    }
  }
}
```



```
StringTokenizer st = new StringTokenizer("Durga Software Solutions");
int count = st.countTokens()

while(st.hasMoreTokens())
{
   Sopln(st.nextToken());
}
```

String Class Library:

Constructor:

- **public String()**

It able to create an emptyy String object.

EX: String str = new Stirng();

- **public String(String str)**

This constructor can be used to create String class object with the specified data.

EX: String str = new String("Durga Software Solutions");
System.out.println(str);
Output: Durga Software Solutions

Q) What is the difference between the folloowing 2 Statements

a) String str = new String("Durga Software Solutions");
b) String str = "Durga Software Solutions";

- To create String class object if we use first statement then one String object will be created at heap memory as per "new" keyword and another String object will be created at String Constant Pool Area inside method Area as per "".
- To create String object if we use second statement then String object will be created at String Constant Pool Area only in Method Area.

Note: The objects which are created in String Constant Pool Area are not eligible for Garbage Collection. The objects which are created in Heap memory are eligible for Garbage Collection.

3) public String(byte[] b)

This constructor can be used to create String object with the String equalent of the specified byte[].

EX:
```
byte[] b = {65, 66, 67, 68, 69, 70};
String str = new String(b);
System.out.println(str);
```
Output: ABCDEF

4) public String(byte[] b, int start_index, int no_Of_Chars)

This constructor can be used to create String class object with the String equalent of the specified byte[] starts from the specified start index and up to the specified no of elements.

EX:
```
byte[] b = {65, 66, 67, 68, 69, 70};
String str = new String(b, 2, 3);
System.out.println(str);
```
Output: CDE

5) public String(char[] ch)

This constructor can be used to create String object with the String equalent of the specified char[].

EX:
```
char[] ch = {'A', 'B', 'C', 'D', 'E', 'F'};
String str = new String(ch);
System.out.println(str);
```
Output: ABCDEF

6) public String(char[] ch, int start_Index,int no_Of_Chars)

It will create String object with the String equalent of the specified char[] starts from the specified start index and up to the specified no of element.

EX:
```
char[] ch = {'A','B','C','D','E','F'};
String str = new String(ch,2,3);
System.out.println(str);
```
Output: CDE

Note: Constructors 3 and 4 are used to convert data from byte[] to String and constructors 5 and 6 are used to convert data from char[] to String.

Methods:

- **public int length()**

This method will return an integer value representing the no of characters existed in the String including spaces.

EX:
```
String str = new String("Durga Software Solutions");
System.out.println(str.length());
```
Output: 24

- **public String concat(String str)**

This method will add the specified String to String object content in immutable manner.

EX:
```
String str1 = new String("Durga ");
String str2 = str1.concat("Software ");
String str3 = str2.concat("Solutions");
System.out.println(str1);
System.out.println(str2);
System.out.println(str3);
```

Output:
```
Durga
Durga Software
Durga Software Solutions
```
EX:
```
String str = "Durga".concat("Software ").concat("Solutions");
System.out.println(str);
```

Output: Durga Software Solutions

- **<u>public boolean equals(Object obj)</u>**

This method will check whether the two String objects content is same or not. if two string objects content is same then equals(-) method will return "true" value otherwise equals(-) method will return "false" value.

<u>Q) What is the difference between == *Operator* and *equals(-)* Method?</u>

- **'==' is basically a comparison operator, it will check whether the provided operand values are same or not, where operands may be normal primitive variables or object reference variables.**

- **Initially equals(-) method was defined in java.lang.Object class , it was implemented in such a way that to provide comparison between two object reference values.**

- **String class has overridden Object class equals(-) method in such a way that to provide comparison between two String object contents**

<u>EX:</u>

```
class A
{
}
class Test
{
  public static void main(String[] args)
  {
    A a1=new A();
    A a2=new A();

    int i=10;
    int j=10;

    String str1=new String("abc");
    String str2=new String("abc");

    System.out.println(i == j);// true
    System.out.println(a1 == a2);// false
    System.out.println(str1 == str2);// false
    System.out.println(a1.equals(a2));// false
    System.out.println(str1.equals(str2));//true
  }
}
```

public boolean equalsIgnoreCase(String str)

In String class, equals(-) method will perform case sensitive comparison between two String object contents, but, equalsIgnoreCase(-) method will perform case insensitive comparison between two String objects content.

EX:

```
String str1 = new String("abc");
String str2 = new String("ABC");
System.out.println(str1.equals(str2));
System.out.println(str1.equalsIgnoreCase(str2));
```

Output:

```
false
true
```

- ## public int compareTo(String str)

This method will check whether two string objects contents are existed in dictionary order or not.

str1.compareTo(str2)

a) If str1 come first when compared with str2 in dictionary order then compareTo() method will return -ve value.
b) If str2 come first when compared with str1 in dictionary order then compareTo() method will return +ve value.
c) If str1 and str2 are available at the same position in dictionary order then compareTo() method will return 0 value.

EX:

```
String str1 = new String("abc");
String str2 = new String("def");
String str3 = new String("abc");
System.out.println(str1.compareTo(str2));
System.out.println(str2.compareTo(str3));
System.out.println(str3.compareTo(str1));
```

Output:

```
-3
 3
 0
```

- **public boolean startsWith(String str)**

  It will check whether the String starts with the specified String or not.

- **public boolean endsWith(String str)**

  It will check whether the String ends with the specified String or not.

- **public boolean contains(String str)**

  It will check whether String contains the specified String or not.

EX:
```
String str=new String("Durga Software Solutions");
System.out.println(str.startsWith("Durga"));
System.out.println(str.endsWith("Solutions"));
System.out.println(str.contains("Software"));
OUTPUT:
true
true
true
```

- **public String replace(char old_Char, char new_Char)**

  This method will replace the specified old char with the specified new char.

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

  It will return a character which is existed at the specified index value.

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

  It will return an index value where the first occurrence of the specified String.

- **public int lastIndexOf(String str)**

  It will return an index value where the last occurrence of the specified String.

EX:
```
String str=new String("Durga Software Solutions");
System.out.println(str);
System.out.println(str.replace('S', 's'));
System.out.println(str.charAt(6));
System.out.println(str.indexOf("So"));
System.out.println(str.lastIndexOf("So"));
```

Output:
```
Durga Software Solutions
Durga software solutions
S
6
15
```

- **public String substring(int start_index)**
  This method return a substring starts from the specified index value.

- **public String substring (int start_Index, int end_Index)**
  This method will return a substring starts from the specified start index and up to the specified end index.

EX:
```
String str = new String("Durga Software Solutions");
System.out.println(str.substring(6));
System.out.println(str.substring(6,14));
```

Output:
```
Software Solutions
Software
```

- **public byte[] getBytes()**
  This method will convert data from String to byte[].

- **public char[] toCharArray()**
  This method will convert data from String to char[].

  EX:

```
String str = new String("Durga Software Solutions");
byte[] b = str.getBytes();
for(int i=0; i<b.length; i++)
{
   System.out.print(b[i]+" ");
}
System.out.println();
char[] ch = str.toCharArray();
for(int i=0; i<ch.length; i++)
{
   System.out.print(ch[i]+" ");
}
```

public String[] split(String str)

It will divide the String into no pieces on the basis of the provided String.

EX:

```
String str = new String("Durga Software Solutions");
String[] s = str.split(" ");
for(int i=0; i<s.length; i++)
{
   System.out.println(s[i]);
}
```

- **public String trim()**

  This method will remove pre-spaces and post-spaces of a particular String.

- **public String toLowerCase()**

  It will convert String into lower case letters.

- **public String toUpperCase()**

  It will convert String into Upper case letters.

EX:
```
String str = new String("   Durga Software Solutions  ");
System.out.println(str);
System.out.println(str.trim());
String str1 = new String("Durga Software Solutions");
System.out.println(str1.toLowerCase());
System.out.println(str1.toUpperCase());
```

StringBuffer:

Costructors:

- **public StringBuffer()**

This constructor can be used to create an empty StringBuffer object with 16 elements as initial capacity.

EX:
```
StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());
```
Output: 16

- **public StringBuffer(int capacity)**

This constructor can be used to create an empty StringBuffer object with the specified capacity value.

EX:
```
StringBuffer sb=new StringBuffer(10);
System.out.println(sb.capacity());
```
Output: 10

- **public StringBuffer(String str)**

This constructor can be used to create StringBuffer object with the specified data.

New_capacity = Iitial_Capacity+data_Length;

EX:
```
StringBuffer sb = new StringBuffer("abc");
System.out.println(sb);
System.out.println(sb.capacity());
```
Output:
```
abc
19
```

Methods:

- **public StringBuffer append(String data)**

This method will append the specified data to the StringBuffer object content.

EX:
```
StringBuffer sb1 = new StringBuffer("Durga ");
StringBuffer sb2 = sb1.append("Software ");
StringBuffer sb3 = sb2.append("Solutions");
System.out.println(sb1);
System.out.println(sb2);
System.out.println(sb3);
```

Output:
```
Durga Software Solutions
Durga Software Solutions
Durga Software Solutions
```

- **public void ensureCapacity(int capacity)**

This method can be used to set a particular capacity value to StringBuffer object. If the provided capacity value is less than 16 then StringBuffer object will take 16 as capacity value. If the provided capacity value is greater than 16 and less than 34 then StringBuffer will take 34 as capacity value. If the provided capacity value is greater than 34 then StringBuffer object will take the specified value as capacity value.

EX:
```
StringBuffer sb = new StringBuffer();
sb.ensureCapacity(10);
System.out.println(sb.capacity());
```

Output: 16

- **public StringBuffer reverse()**

This method will reverse the content of StringBuffer object.

EX:
```
StringBuffer sb = new StringBuffer("Durga Software Solutions");
System.out.println(sb);
System.out.println(sb.reverse());
```

public StringBuffer delete(int start_index, int end_index)

This method can be used to delete a string from StringBuffer object content starts from the specified start index and up to the specified end index.

EX:
```
StringBuffer sb = new StringBuffer("Durga Software Solutions");
System.out.println(sb);
System.out.println(sb.delete(6,14));
```

Output:
```
Durga Software Solutions
Durga  Solutions
```

- **public StringBuffer insert(int index, String str)**

This method can be used to insert the specified String at the specified index in StringBuffer object.

EX:
```
StringBuffer sb = new StringBuffer("Durga  Solutions");
System.out.println(sb);
System.out.println(sb.insert(6,"Software"));
```

Output:
```
Durga  Solutions
Durga Software Solutions
```