String Manipulations

Introduction

String operations like calculating length, comparison, tokenization, reversal, and trimming are common in applications.

- **C and C++:** Utilize predefined libraries with functions like `strlen`, `trim`, `toLowerCase`, `toUpperCase`.
- **Java:** Provides predefined classes for string operations:
    1. `java.lang.String`
    2. `java.lang.StringBuffer`
    3. `java.lang.StringBuilder`
    4. `java.util.StringTokenizer`

Q) What is the difference between String and StringBuffer?
—---------- Ans: —--- `String` class objects are **immutable**.

- Immutable objects do not allow modifications to their content directly.
- If an operation modifies the content, the result is stored in a new object of the same class, not in the original object.

`StringBuffer` class objects are **mutable**.

- Mutable objects allow direct modifications to their content.

Q)What are the differences between StringBuffer and StringBuilder?
—---------- Ans: —-- The following table summarizes the differences:

Feature	StringBuffer	StringBuilder
Availability	Java 1.0	Java 1.5
Synchronization	Synchronized	Non-Synchronized
Method Sync	Almost all methods are synchronized	No method is synchronized
Thread Access	Allows only one thread at a time	Allows more than one thread at a time
Execution Model	Sequential execution	Parallel execution
Execution Time	Takes more execution time	Takes less execution time
Performance	Reduces application performance	Improves application performance
Data Consistency	Guaranteed	Not guaranteed
Thread Safety	Thread-safe resource	Not a thread-safe resource

Export to Sheets

Q) What is String Tokenization and how is it possible to perform String Tokenization in Java applications? —---------- Ans: —----
String Tokenization: The process of dividing a string into multiple pieces, where each piece is called a **Token**.

Java provides the `java.util.StringTokenizer` class for this purpose.

Steps for String Tokenization in Java:

1. **Create a `StringTokenizer` class object.**
    - Constructors:
        - `public StringTokenizer(String data)`
            - Performs tokenization based on the default delimiter (space).
            - Example: "Durga Software Solutions" results in tokens: "Durga", "Software", "Solutions".
        - `public StringTokenizer(String data, String delimiter)`
            - Performs tokenization based on the provided delimiter.
            - Example: "10-12-2025" with delimiter "-" results in tokens: "10", "12", "2025".
    - Example Usage:

      Java

      ```
      StringTokenizer st = new StringTokenizer("Durga Software
      Solutions");
      // OR
      StringTokenizer st = new StringTokenizer("10-12-2025", "-");
      ```

    - **JVM Action:** When the object is created, JVM divides the string into tokens based on the delimiter and stores them in the `StringTokenizer` object. A cursor is automatically created before the first token.
2. **Read tokens from the `StringTokenizer` class object.**
    - Methods:
        - `public boolean hasMoreTokens()`: Returns `true` if at least one more token is available from the current cursor position, `false` otherwise.
        - `public String nextToken()`: Reads the next token and moves the cursor to the next position.
        - `public int countTokens()`: Returns the total number of tokens available in the `StringTokenizer` object.

Example Code for String Tokenization:

Java
```
import java.util.StringTokenizer;

public class Main {
  public static void main(String[] args){
      String data1 = "Durga Software Solutions";
```

```
        StringTokenizer st1 =new StringTokenizer(data1);
        int tokesCount1 =st1.countTokens();
        System.out.println("DATA          : "+data1);
        System.out.println("No Of Tokens : "+tokesCount1);
        System.out.println("Tokens        : ");
        while(st1.hasMoreTokens()){
            System.out.println("\t\t"+st1.nextToken()+" ");
        }
        System.out.println();
        System.out.println("-----------------------------------");
        String data2 = "10-12-2024";
        StringTokenizer st2 =new StringTokenizer(data2, "-");
        int tokesCount2 =st2.countTokens();
        System.out.println("DATA          : "+data2);
        System.out.println("No Of Tokens : "+tokesCount2);
        System.out.println("Tokens        : ");
        while(st2.hasMoreTokens()){
            System.out.println("\t\t"+st2.nextToken()+" ");
        }


    }
}
```

Output:

```
DATA          : Durga Software Solutions
No Of Tokens : 3
Tokens        :
                Durga
                Software
                Solutions

-----------------------------------
DATA          : 10-12-2024
No Of Tokens : 3
Tokens        :
                10
                12
                2024
```

`String` Class Library (`java.lang.String`)

- A **String** is a collection of characters, digits, special symbols, etc.
- In Java, data enclosed in double quotations (`"---"`) is treated as String data or a String Literal.
    - Examples:

      Java

      ```
      String name = "Durga";
      String mobile = "91-9988776655";
      String email = "durga123@gmail.com";
      ```

- **Note:** In Java / J2EE, only the `String` data type can represent all types of data (characters, numbers, special symbols).

- `String` is a class in Java (`java.lang.String`), not a primitive data type.

`String` Class Constructors

1. **`public String()`**: Creates an empty String object.
    - Example:

      Java

      ```
      String str = new String();
      System.out.println(str); // Prints an empty line
      ```

2. **`public String(String data)`**: Creates a String object with the provided data.
    - Example:

      Java

      ```
      public class Main {
        public static void main(String[] args){
            String str = new String("Durgasoft");
            System.out.println(str);
        }
      }
      ```

      Output:

      ```
      Durgasoft
      ```

Q) What is the difference between the following two statements?

Java
```
1. String str1 = "abc";
2. String str2 = new String("abc");
```

—---------- Ans: —---

- **`String str1 = "abc";`**:
    - JVM creates a String object with "abc" in the **String constant pool** (inside the Method Area).
    - Objects in the String constant pool are not eligible for Garbage Collection and are destroyed when program execution completes.
    - These objects are **reusable**. JVM first checks if an object with the same data exists in the pool.
        - If it exists, JVM returns the reference to the existing object.
        - If not, JVM creates a new String object in the pool.
- **`String str2 = new String("abc");`**:
    - JVM creates **two** String objects:
        - One in the **String constant pool** (due to the double quotations `"abc"`).
        - Another in the **Heap memory** (due to the `new` keyword).

- The reference variable (`str2`) refers only to the Heap memory object.
- String objects created in Heap memory are **not reusable** and are **eligible for Garbage Collection**.

Example Code:

Java
```
public class Main {
  public static void main(String[] args){
      String str1 = "abc";
      String str2 = "abc";
      System.out.println(str1 == str2);

      String str3 = new String("xyz");
      String str4 = new String("xyz");
      System.out.println(str3 == str4);

  }
}
```

Output:

```
true
false
```

---

3. **`public String(byte[] bytes)`**: Creates a String object from the string equivalent of the provided `byte[]`, where `byte[]` contains ASCII values.
   - Example:

     Java

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

           byte[] bytes = {65, 66, 67, 68, 69, 70, 71, 72, 73,
     74}; // ASCII for A-J
           String str = new String(bytes);
           System.out.println(str); // Output: ABCDEFGHIJ

       }
     }
     ```

4. **`public String(byte[] bytes, int startIndex, int noOfElements)`**: Creates a String object from a sub-array of `byte[]`, starting from `startIndex` and including `noOfElements` characters.
   - Example:

     Java

     ```
     public class Main {
       public static void main(String[] args){
           byte[] bytes = {65, 66, 67, 68, 69, 70, 71, 72, 73, 74};
           String str = new String(bytes, 3, 4); // Starts at index
     3 (D), takes 4 elements
     ```

```
        System.out.println(str);
    }
}
```

Output:

```
DEFG
```

- **Note:** Constructors 3 and 4 are used to convert data from `byte[]` to `String`.

5. **`public String(char[] chars)`**: Creates a String object from the string equivalent of the provided `char[]`.
    - Example:

      Java

```
public class Main {
  public static void main(String[] args){
      char[] chars = {'A','B','C','D','E','F'};
      String str = new String(chars);
      System.out.println(str);
  }
}
```

      Output:

```
ABCDEF
```

6. **`public String(char[] chars, int startIndex, int noOfChars)`**: Creates a String object from a sub-array of `char[]`, starting from `startIndex` and including `noOfChars` characters.
    - Example:

      Java

```
public class Main {
  public static void main(String[] args){
      char[] chars = {'A','B','C','D','E','F'};
      String str = new String(chars, 2,3); // Starts at index 2
(C), takes 3 chars
      System.out.println(str);
  }
}
```

      Output:

```
CDE
```

    - **Note:** Constructors 5 and 6 are used to convert data from `char[]` to `String`.

`String` Class Methods

1. **`public int length()`**: Returns an integer representing the size or length of the String.

Q)What is the difference between 'length' and 'length()'?
—---------- Ans: —---

- **`length`**: A predefined **variable** in all arrays in Java. It represents the length of the array (number of elements).
- **`length()`**: A predefined **method** in the `String` class. It represents the length of the String (number of characters).

Example Code:

Java
```
public class Main {
  public static void main(String[] args){
      int[] ints = {10,20,3,40,50};
      System.out.println(ints.length);

      String str = new String("abcdef");
      System.out.println(str.length());
  }
}
```

Output:

```
5
6
```

---

2. **`public String toString()`**: Returns the content of the String object.
    - When a String object reference is passed to `System.out.println()`, JVM internally calls `toString()`.
    - The `String` class overrides the `Object` class's `toString()` method to return the String's content.
    - Example:

      Java

      ```
      class A{

      }
      public class Main {
        public static void main(String[] args){
            A a = new A();
            System.out.println(a);//A@<hashcode> (e.g., A@30f39991)
            String str =new String("hello");
            System.out.println(str);//hello
         }
      }
      ```

      Output (actual hashcode for 'a' will vary):

```
A@30f39991
hello
```

3. **`public boolean equals(Object obj)`**: Checks if two String objects' data is the same. Returns `true` if data is same, `false` otherwise (case-sensitive).
    - Example:

      Java

      ```
      public class Main {
        public static void main(String[] args){
            String str1 = new String("abc");
            String str2 = new String("xyz");
            String str3 = new String("abc");
            System.out.println(str1.equals(str2));
            System.out.println(str1.equals(str3));
        }
      }
      ```

      Output:

      ```
      false
      true
      ```

Q)What is the difference between == operator and the equals() method?
—---------- Ans: —----

- **== operator**:
    - A comparison operator.
    - Checks if two operand values are the same. Operands can be primitive variables or object reference variables.
    - For objects, == compares if the reference variables point to the **same memory location**.
- **`equals()` method**:
    - Initially defined in `java.lang.Object` class, where it performs reference comparison (like ==).
    - In `java.lang.String` class, `equals()` is **overridden** to perform **content comparison** of the two String objects.

Example Code:

Java
```
class A{

}
public class Main {
  public static void main(String[] args){
      int i = 10;
      int j = 20;

      A a1 = new A();
      A a2 = new A();
```

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

      System.out.println(i == j);// false
      System.out.println(a1 == a2);// false (different objects in memory)
      System.out.println(str1 == str2);// false (different objects in
memory due to 'new')
      System.out.println();

      System.out.println(a1.equals(a2));// false (Object.equals() compares
references)
      System.out.println(str1.equals(str2));// true (String.equals()
compares content)



   }
}
```

Output:

```
false
false
false

false
true
```

---

4. **`public boolean equalsIgnoreCase(String str)`**: Compares two String objects' contents, ignoring case differences.
    - Example:

      Java

      ```
      public class Main {
        public static void main(String[] args){
            String str1 = new String("abc");
            String str2 = new String("ABC");
            System.out.println(str1.equals(str2));// false
            System.out.println(str1.equalsIgnoreCase(str2));// true
         }
      }
      ```

5. **`public boolean startsWith(String data)`**: Checks if a string starts with the provided `data`.
6. **`public boolean endsWith(String data)`**: Checks if a string ends with the provided `data`.
7. **`public boolean contains(String data)`**: Checks if a string contains the provided `data`.
    - Example:

      Java

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

```
        String str = new String("Durga Software Solutions");
        System.out.println(str.startsWith("Durga"));       //
    true
        System.out.println(str.endsWith("Solutions"));   // true
        System.out.println(str.contains("Software"));    // true
        System.out.println(str.startsWith("Solutions")); // false
        System.out.println(str.endsWith("Durga"));       // false
        System.out.println(str.contains("Durgasoft"));   // false
      }
    }
```

8. **`public int compareTo(String str)`**: Compares two String objects' contents based on dictionary (lexicographical) order.
    - `str1.compareTo(str2)`:
        - Returns a **negative** value if `str1` comes before `str2`.
        - Returns a **positive** value if `str1` comes after `str2`.
        - Returns **0** if `str1` and `str2` are identical.
    - Example:

      Java

```
public class Main {
  public static void main(String[] args){
      String str1 = new String("abc");
      String str2 = new String("xyz");
      String str3 = new String("abc");
      System.out.println(str1.compareTo(str2));// -23 (abc vs
xyz)
      System.out.println(str2.compareTo(str3));// 23 (xyz vs
abc)
      System.out.println(str3.compareTo(str1));// 0 (abc vs
abc)
   }
}
```

9. **`public char charAt(int index)`**: Returns the character at the specified `index`.
10. **`public int indexOf(String value)`**: Returns the index of the first occurrence of the specified `value`.
11. **`public int lastIndexOf(String value)`**: Returns the index of the last occurrence of the specified `value`.
12. **`public String replace(char oldChar, char newChar)`**: Replaces all occurrences of `oldChar` with `newChar`.
    - Example:

      Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
      String str = new String("Durga Software Solutions");
      System.out.println(str);
      System.out.println(str.charAt(6));         // S
      System.out.println(str.indexOf("So"));       // 6
(Software)
```

```
        System.out.println(str.lastIndexOf("So"));   // 15
(Solutions)
        System.out.println(str.replace('S','s')); // Durga
software solutions
    }
}
```

13. **`public String substring(int startIndex)`**: Generates a substring starting from `startIndex` to the end of the string.
14. **`public String substring(int startIndex, int endIndex)`**: Generates a substring from `startIndex` up to (but not including) `endIndex`.
    - Example:

      Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
        String str = new String("Durga Software Solutions");
        System.out.println(str);
        System.out.println(str.substring(6));      // Software
Solutions
        System.out.println(str.substring(6, 14));  // Software
(index 14 is ' ')
    }
}
```

15. **`public String concat(String str)`**: Appends the provided `str` to the end of the current string content in an immutable manner (returns a new string).
    - Example 1:

      Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
        String str1 = new String("Durga ");
        String str2 = str1.concat("Software ");
        String str3 = str2.concat("Solutions");

        System.out.println(str1);// Durga
        System.out.println(str2);// Durga Software
        System.out.println(str3);// Durga Software Solutions
    }
}
```

    - Example 2 (Chaining):

      Java

```
// import java.util.Scanner; // Not used in this snippet

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

```
        String str = "Durga ".concat("Software
").concat("Solutions");
        System.out.println(str); // Durga Software Solutions
    }
}
```

- **Note:** The + operator can also be used for string concatenation in Java.

  Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
    public static void main(String[] args){
        String str = "Durga "+"Software "+"Solutions";
        System.out.println(str); // Durga Software Solutions
    }
}
```

16. **`public byte[] getBytes()`**: Converts the String data to a `byte[]` (typically using the platform's default charset).
    - Example:

      Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
    public static void main(String[] args){
      String data = "Durga Software Solutions";
        System.out.println(data);
      byte[] bytes = data.getBytes();
      for (byte b : bytes) {
          System.out.println(b+"----->"+(char)b);
      }
    }
}
```

      Output (ASCII values):

```
Durga Software Solutions
68----->D
117----->u
114----->r
103----->g
97----->a
32----->
83----->S
111----->o
102----->f
116----->t
119----->w
97----->a
114----->r
101----->e
32----->
83----->S
111----->o
```

```
108----->l
117----->u
116----->t
105----->i
111----->o
110----->n
115----->s
```

17. **`public char[] toCharArray()`**: Converts the String data into a `char[]`.
    - Example:

      Java

      ```
      // import java.util.Scanner; // Not used in this snippet

      public class Main {
        public static void main(String[] args){
          String data = "Durga Software Solutions";
          System.out.println(data);
          char[] chars = data.toCharArray();
          for (char c : chars){
              System.out.print(c+"  ");
          }
        }
      }
      ```

      Output:

      ```
      Durga Software Solutions
      D  u  r  g  a     S  o  f  t  w  a  r  e     S  o  l  u  t  i
      o  n  s
      ```

18. **`public String[] split(String delimiter)`**: Splits the string into an array of strings (`String[]`) based on the provided `delimiter`.
    - Example:

      Java

      ```
      // import java.util.Scanner; // Not used in this snippet

      public class Main {
        public static void main(String[] args){
          String data = "Durga Software Solutions";
          System.out.println(data);
          String[] str = data.split(" "); // Split by space
          for(String s : str){
              System.out.println(s);
          }
        }
      }
      ```

      Output:

      ```
      Durga Software Solutions
      Durga
      Software
      Solutions
      ```

19. **`public String toLowerCase()`**: Converts all uppercase letters in the string to lowercase.
20. **`public String toUpperCase()`**: Converts all lowercase letters in the string to uppercase.
    - Example:

      Java

      ```
      // import java.util.Scanner; // Not used in this snippet

      public class Main {
        public static void main(String[] args){
          String data = "Durga Software Solutions";
          System.out.println(data);
          System.out.println(data.toLowerCase());
          System.out.println(data.toUpperCase());
        }
      }
      ```

      Output:

      ```
      Durga Software Solutions
      durga software solutions
      DURGA SOFTWARE SOLUTIONS
      ```

21. **`public String trim()`**: Removes leading and trailing whitespace from the string.
    - Example:

      Java

      ```
      // import java.util.Scanner; // Not used in this snippet

      public class Main {
        public static void main(String[] args){
          String data = "      Durga Software Solutions         ";
          System.out.println(data);
          System.out.println(data.trim());
        }
      }
      ```

      Output:

      ```
            Durga Software Solutions
      Durga Software Solutions
      ```

Note on Immutability and Object Creation: In the `String` class, if operations like `concat`, `trim`, etc., do not result in a change to the string's data, JVM will not create a new object. A new object is created only if the data is actually modified.

- Example:

  Java

  ```
  // import java.util.Scanner; // Not used in this snippet
  ```

```
public class Main {
  public static void main(String[] args){
      String str1 = new String("abc");
      String str2 = str1.concat(" "); // Data changed, new object
      String str3 = str1.concat("");   // Data not changed, same
object
      System.out.println(str1 == str2);// false
      System.out.println(str1 == str3);// true

      String str4 =new String("abc");
      String str5 = str4.trim();      // Data not changed (no
leading/trailing spaces), same object
      System.out.println(str4 == str5); // true

      String str6 = new String(" abc");
      String str7 = str6.trim();      // Data changed, new object
      System.out.println(str6 == str7); // false
  }
}
```

`StringBuffer` Class Library

—----------

Constructors:

1. **`public StringBuffer()`**:
   - Provides an empty `StringBuffer` object with an initial capacity of 16 elements.
   - EX:

     Java

     ```
     // import java.util.Scanner; // Not used in this snippet

     public class Main {
       public static void main(String[] args){
           StringBuffer sb = new StringBuffer();
           System.out.println(sb.capacity());

       }
     }
     ```

     Output:

     ```
     16
     ```

2. **`public StringBuffer(int capacity)`**:
   - Creates an empty `StringBuffer` object with the specified `capacity` value.
   - EX:

     Java

     ```
     // import java.util.Scanner; // Not used in this snippet
     ```

```
public class Main {
  public static void main(String[] args){
      StringBuffer sb = new StringBuffer(20);
      System.out.println(sb.capacity());

  }
}
```

Output:

```
20
```

3. **`public StringBuffer(String data)`**:
   - Creates a `StringBuffer` object with the provided `data`.
   - **Note:** This constructor converts data from `String` to `StringBuffer`. The capacity will be `InitialDefaultCapacity + data.length()` (i.e., `16 + data.length()`).
   - EX:

Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
      StringBuffer sb = new StringBuffer("abc");
      System.out.println(sb);
      System.out.println(sb.capacity());//
InitialCapacity+dataLength

   }
}
```

Output:

```
abc
19
```

Methods:

—---------

1. **`public String toString()`**:
   - Converts the data from `StringBuffer` to `String`.
   - EX:

Java

```
// import java.util.Scanner; // Not used in this snippet

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

```
        StringBuffer sb = new StringBuffer("Durga Software
Solutions");
        System.out.println(sb);
        String str = sb.toString();
        System.out.println(str);

    }
}
```

Output:

```
Durga Software Solutions
Durga Software Solutions
```

2. **`public int capacity()`**:
    - Returns the current capacity of the `StringBuffer` object.
    - EX:

Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
        StringBuffer sb1 = new StringBuffer();
        System.out.println(sb1+" -----> "+sb1.capacity());

        StringBuffer sb2 = new StringBuffer("abc");
        System.out.println(sb2+" -----> "+sb2.capacity());


    }
}
```

Output:

```
 -----> 16
abc -----> 19
```

3. **`public StringBuffer append(String data)`**:
    - Appends the provided `data` to the `StringBuffer` object's content. This is a mutable operation; the original `StringBuffer` object is modified.
    - The method returns a reference to the same `StringBuffer` object.
    - EX:

Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){

        StringBuffer sb1 = new StringBuffer("Durga ");
        System.out.println(sb1);// Durga
        StringBuffer sb2 = sb1.append("Software ");
        System.out.println(sb1);// Durga Software
```

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

        System.out.println(sb1 == sb2);// true
        System.out.println(sb2 == sb3);// true
        System.out.println(sb3 == sb1);// true
        System.out.println();

        System.out.println(sb1);// Durga Software Solutions
        System.out.println(sb2);// Durga Software Solutions
        System.out.println(sb3);// Durga Software Solutions




    }
}
```

Output:

```
Durga
Durga Software
Durga Software Solutions

true
true
true

Durga Software Solutions
Durga Software Solutions
Durga Software Solutions
```

4. **`public StringBuffer reverse()`**:
    - Reverses the content of the `StringBuffer` object. This is a mutable operation.
    - EX:

Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){

        StringBuffer sb1 = new StringBuffer("DURGA SOFTWARE
SOLUTIONS");
        System.out.println(sb1);
        System.out.println(sb1.reverse());

  }
}
```

Output:

```
DURGA SOFTWARE SOLUTIONS
SNOITULOS ERAWTFOS AGRUD
```

- EX (Palindrome Check):

  Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
      String str1 = "LEVEL";
      StringBuffer sb1 = new StringBuffer(str1);
      System.out.println(str1);

      StringBuffer sb2 = sb1.reverse(); // sb1 is also reversed
      String str2 = sb2.toString();
      System.out.println(str2);

      if(str1.equals(str2)){
          System.out.println(str1+" Is Palindrome String");
      }else{
          System.out.println(str1+" Is Not Palindrome String");
      }



  }
}
```

  Output:

```
LEVEL
LEVEL
LEVEL Is Palindrome String
```

EX on Palindrome String:

Java

```
// import java.util.Scanner; // Not used in this snippet

public class Main {
  public static void main(String[] args){
      String str1 = "LEVEL";
      boolean flag = true;
      int length = str1.length();

      for(int index = 0; index < length / 2; index++){ // Optimized loop
condition
          if(str1.charAt(index) != str1.charAt(length- 1 - index)){
              flag = false;
              break;
          }
      }
      if(flag == true){
          System.out.println(str1+" is Palindrome String ");
      }else{
          System.out.println(str1+" is not Palindrome String ");
      }
```

```
  }
}
```

Output:

```
LEVEL is Palindrome String
```

5. **`public StringBuffer insert(int index, String data)`**:
   - Inserts the provided `data` into the `StringBuffer` object at the specified `index`.
   - EX:

     Java

     ```
     // import java.util.Scanner; // Not used in this snippet

     public class Main {
       public static void main(String[] args){
           StringBuffer sb = new StringBuffer("Durga Solutions"); //
     Note: Original had a non-breaking space, using regular space
     here
           System.out.println(sb);
           System.out.println(sb.insert(6,"Software ")); // Added
     space for clarity
       }
     }
     ```

     Output:

     ```
     Durga Solutions
     Durga Software Solutions
     ```

6. **`public StringBuffer delete(int startIndex, int endIndex)`**:
   - Deletes the portion of the string from the `StringBuffer` object starting at `startIndex` up to (but not including) `endIndex`.
   - EX:

     Java

     ```
     // import java.util.Scanner; // Not used in this snippet

     public class Main {
       public static void main(String[] args){
           StringBuffer sb = new StringBuffer("Durga Software
     Solutions");
           System.out.println(sb);
           System.out.println(sb.delete(6,15)); // Deletes "Software
     " (index 15 is exclusive)
       }
     }
     ```

     Output:

     ```
     Durga Software Solutions
     ```

Durga Solutions