StringBuffer class in Java

Difficulty Level : Easy Last Updated : 19 Jan, 2022

StringBuffer is a peer class of **String** that provides much of the functionality of strings. The string represents fixed-length, immutable character sequences while StringBuffer represents growable and writable character sequences. **StringBuffer** may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

Some Interesting Facts about the StringBuffer class

Do keep in the back of mi while applying so which are as follows:

- java.lang.StringBuffer extends (or inherits from) Object class.
- All Implemented Interfaces of StringBuffer class: Serializable, Appendable, CharSequence.
- public final class StringBuffer extends Object implements Serializable, CharSequence, Appendable.
- String buffers are safe for use by multiple threads. The methods can be synchronized wherever necessary so that all the operations on any particular instance behave as if they occur in some serial order.
- Whenever an operation occurs involving a source sequence (such as appending or inserting from a source sequence) this class synchronizes only on the string buffer performing the operation, not on the source.
- It inherits some of the methods from the Object class which such as *clone(), equals(), finalize(), getClass(), hashCode(),* notifies(), *notifyAll().*

Remember: *StringBuilder**,** J2SE 5 adds a new string class to Java's already powerful string handling capabilities. This new class is called StringBuilder. It is identical to StringBuffer e xcept for one important difference: it is not synchronized, which means that it is not thread-safe. The advantage of StringBuilder is faster performance. However, in cases in which you are using multithreading, you must use StringBuffer rather than StringBuilder.*

Constructors of StringBuffer class

1. StringBuffer(): It reserves room for 16 characters without reallocation

```
StringBuffer s = new StringBuffer();
```

2. StringBuffer(int size): It accepts an integer argument that explicitly sets the size of the buffer.

```
StringBuffer s = new StringBuffer(20);
```

3. StringBuffer(String str): It accepts a string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

Methods of StringBuffer class

Methods	Action Performed
append()	Used to add text at the end of the existing text.
length()	The length of a StringBuffer can be found by the length( ) method
capacity()	the total allocated capacity can be found by the capacity( ) method
charAt()	
delete()	Deletes a sequence of characters from the invoking object
deleteCharAt()	Deletes the character at the index specified by *loc*
ensureCapacity()	Ensures capacity is at least equals to the given minimum.
insert()	Inserts text at the specified index position
length()	Returns length of the string
reverse()	Reverse the characters within a StringBuffer object
replace()	Replace one set of characters with another set inside a StringBuffer object

Note: *Besides that, all the methods that are used in the String class can also be used. Tese a uxiliary methods are as follows:*

- **ensureCapacity()**

It is used to increase the capacity of a StringBuffer object. The new capacity will be set to either the value we specify or twice the current capacity plus two (i.e. capacity+2), whichever is larger. Here, capacity specifies the size of the buffer.

Syntax:

```
void ensureCapacity(int capacity)
```

- **appendCodePoint(int codePoint):** This method appends the string representation of the codePoint argument to this sequence.

Syntax:

```
public StringBuffer appendCodePoint(int codePoint)
```

- **charAt(int index)**

This method returns the char value in this sequence at the specified index.

Syntax:

```
public char charAt(int index)
```

- **IntStream chars()**: This method returns a stream of int zero-extending the char values from this sequence.

Syntax:

```
public IntStream chars()
```

- **int codePointAt(int index)**: This method returns the character (Unicode code point) at the specified index.

Syntax:

```
public int codePointAt(int index)
```

- **int codePointBefore(int index)**: This method returns the character (Unicode code point) before the specified index.

Syntax:

```
public int codePointBefore(int index)
```

- **int codePointCount(int beginIndex, int endIndex)**: This method returns the number of Unicode code points in the specified text range of this sequence.

Syntax:

```
public int codePointCount(int beginIndex, int endIndex)
```

- **IntStream codePoints()**: This method returns a stream of code point values from this sequence.

Syntax:

```
public IntStream codePoints()
```

- **void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)**: In this method, the characters are copied from this sequence into the destination character array dst.

Syntax:

```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

- **int indexOf(String str)**: This method returns the index within this string of the first occurrence of the specified substring.

Syntax:

```
public int indexOf(String str)
public int indexOf(String str, int fromIndex)
```

- **int lastIndexOf(String str)**: This method returns the index within this string of the last occurrence of the specified substring.

Syntax:

```
public int lastIndexOf(String str)
public int lastIndexOf(String str, int fromIndex)
```

- **int offsetByCodePoints(int index, int codePointOffset)**: This method returns the index within this sequence that is offset from the given index by codePointOffset code points.

Syntax:

```
public int offsetByCodePoints(int index, int codePointOffset)
```

- **void setCharAt(int index, char ch)**: In this method, the character at the specified index is set to ch.

Syntax:

```
public void setCharAt(int index, char ch)
```

- **void setLength(int newLength)**: This method sets the length of the character sequence.

Syntax:

```
public void setLength(int newLength)
```

- **CharSequence subSequence(int start, int end)**: This method returns a new character sequence that is a subsequence of this sequence.

Syntax:

```
public CharSequence subSequence(int start, int end)
```

- **String substring(int start)**: This method returns a new String that contains a subsequence of characters currently contained in this character sequence.

Syntax:

```
public String substring(int start)
public String substring(int start,int end)
```

- **String toString()**: This method returns a string representing the data in this sequence.

Syntax:

```
public String toString()
```

- **void trimToSize()**: This method attempts to reduce storage used for the character sequence.

Syntax:

```
public void trimToSize()
```

Above we only have discussed the most widely used methods and do keep a tight bound aro und them as they are widel used in programming geeks.

Implementation:

Example 1: *length() and capacity()* Methods

```
// Java Program to Illustrate StringBuffer class
// via length() and capacity() methods

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // main driver method
    public static void main(String[] args)
    {

        // Creating adn storing string by creating object of
        // StringBuffer
        StringBuffer s = new StringBuffer("GeeksforGeeks");

        // Getting the length of the string
        int p = s.length();

        // Getting the capacity of the string
        int q = s.capacity();

        // Printing the length and capacity of
```

```
            // above generated input string on console
            System.out.println("Length of string GeeksforGeeks="
                               + p);
            System.out.println(
                "Capacity of string GeeksforGeeks=" + q);
        }
    }
```

Output

```
Length of string GeeksforGeeks=13
Capacity of string GeeksforGeeks=29
```

Example 2: append()

It is used to add text at the end of the existing text.

Here are a few of its forms:

```
StringBuffer append(String str)
StringBuffer append(int num)
```

```
// Java Program to Illustrate StringBuffer class
// via append() method

// Importing required classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an object of StringBuffer class and
        // passing random string
        StringBuffer s = new StringBuffer("Geeksfor");

        // Usage of append() method
        s.append("Geeks");

        // Returns GeeksforGeeks
        System.out.println(s);
```

```
            s.append(1);
            // Returns GeeksforGeeks1
            System.out.println(s);
        }
    }
```

Output

```
GeeksforGeeks
GeeksforGeeks1
```

Example 3: insert()

It is used to insert text at the specified index position.

Syntax: These are a few of its as follows:

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
```

Here, the *index* specifies the index at which point the string will be inserted into the invoking **StringBuffer** object.

```
// Java Program to Illustrate StringBuffer class
// via insert() method

// Importing required I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an object of StringBuffer class
        StringBuffer s = new StringBuffer("GeeksGeeks");

        // Inserting element and position as an arguments
        s.insert(5, "for");
        // Returns GeeksforGeeks
```

```
            System.out.println(s);

            s.insert(0, 5);
            // Returns 5GeeksforGeeks
            System.out.println(s);

            s.insert(3, true);
            // Returns 5GetrueeksforGeeks
            System.out.println(s);

            s.insert(5, 41.35d);
            // Returns 5Getr41.35ueeksforGeeks
            System.out.println(s);

            s.insert(8, 41.35f);
            // Returns 5Getr41.41.3535ueeksforGeeks
            System.out.println(s);

            // Declaring and initializing character array
            char geeks_arr[] = { 'p', 'a', 'w', 'a', 'n' };

            // Inserting character array at offset 9
            s.insert(2, geeks_arr);
            // Returns 5Gpawanetr41.41.3535ueeksforGeeks
            System.out.println(s);
        }
    }
```

Output:

```
GeeksforGeeks
5GeeksforGeeks
5GetrueeksforGeeks
5Getr41.35ueeksforGeeks
5Getr41.41.3535ueeksforGeeks
5Gpawanetr41.41.3535ueeksforGeeks
```

Example 4: reverse( )

It can reverse the characters within a StringBuffer object using **reverse( ).** This method returns the reversed object on which it was called.

```
// Java Program to Illustrate StringBuffer class
// via reverse() method

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating a string via creating
        // object of StringBuffer class
        StringBuffer s = new StringBuffer("GeeksGeeks");

        // Invoking reverse() method
        s.reverse();

        // Returns "skeeGrofskeeG"
        System.out.println(s);
    }
}
```

Output

```
skeeGskeeG
```

Example 5: delete( ) and deleteCharAt()

It can delete characters within a StringBuffer by using the methods **delete( )** and **deleteCharAt( )**.The **delete( )** method deletes a sequence of characters from the invoking object. Here, the start Index specifies the index of the first character to remove, and the end Index specifies an index one past the last character to remove. Thus, the substring deleted runs from start Index to endIndex–1. The resulting StringBuffer object is returned. The **deleteCharAt( )** method deletes the character at the index specified by *loc.* It returns the resulting StringBuffer object.

Syntax:

```
StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)
```

```
// Java Program to Illustrate StringBuffer class
// via delete() and deleteCharAt() Methods

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        StringBuffer s = new StringBuffer("GeeksforGeeks");

        s.delete(0, 5);
        // Returns forGeeks
        System.out.println(s);

        s.deleteCharAt(7);
        // Returns forGeek
        System.out.println(s);
    }
}
```

Output

```
forGeeks
forGeek
```

Example 6: replace()

It can replace one set of characters with another set inside a StringBuffer object by calling replace( ). The substring being replaced is specified by the indexes start Index and endIndex. Thus, the substring at start Index through endIndex–1 is replaced. The replacement string is passed in str. The resulting StringBuffer object is returned.

Syntax:

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

Example

```
// Java Program to Illustrate StringBuffer class
// via replace() method

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        StringBuffer s = new StringBuffer("GeeksforGeeks");
        s.replace(5, 8, "are");

        // Returns GeeksareGeeks
        System.out.println(s);
    }
}
```

Output

```
GeeksareGeeks
```

This article is contributed by **Lokesh Todwal.** Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.