StringBuffer and String Builder

CHAPTER 10

You have learnt in previous chapter that Strings are immutable and cannot be modified. To overcome this, we got another class 'StringBuffer' which represents strings in such a way that their data can be modified. It means StringBuffer class objects are mutable. And there are methods provided in this class which directly manipulate the data inside the object.

Important Interview Question

What is the difference between String and StringBuffer classes?

String class objects are immutable and hence their contents cannot be modified. StringBuffer class objects are mutable, so they can be modified. Moreover the methods that directly manipulate data of the object are not available in String class. Such methods are available in StringBuffer class.

Are there any other classes whose objects are immutable?

Yes, classes like Character, Byte, Integer, Float, Double, Long... called 'wrapper classes' are created as 'immutable'. Classes like Class, BigInteger, BigDecimal are also immutable.

Creating StringBuffer Objects

There are two ways to create a StringBuffer object, and fill the object with a string:

- We can create a StringBuffer object by using new operator and pass the string to the object, as:

```
StringBuffer sb = new StringBuffer("Hello");
```

Here, we are passing the string "Hello" to the StringBuffer object 'sb'. So, a statement like:

```
System.out.println(sb);
will display "Hello".
```

- Another way of creating a StringBuffer object is to first allotting memory to the StringBuffer object using new operator and later storing the string into it, as:

```
StringBuffer sb = new StringBuffer();
```

Here, we are creating a StringBuffer object as an empty object and not passing any string to it. In this case, a StringBuffer object will be created with a default capacity of 16 characters.

```
StringBuffer sb = new StringBuffer(50);
```

Here, StringBuffer object is created as an empty object with a capacity for storing 50 characters. course, even if we declare the capacity as 50, it is possible to store more than 50 characters ir this StringBuffer. The reason is that StringBuffer is mutable and can expand dynamically memory. To store characters, we can use 'append()' method or insert() methods, as:

- sb.append("Hello"); //add "Hello" to sb.
- sb.insert(0, "Hello"); //insert "Hello" starting from 0th position in sb.

StringBuffer Class Methods

There are some methods in StringBuffer class as given here:

- StringBuffer append(x): x may be boolean, byte, int, long, float, double, char, character arr String or another StringBuffer. It will be added to the StringBuffer object. For example,

```
StringBuffer sb = new StringBuffer("Uni");
sb.append("versity");
```

Preceding two statements produce, "University" as the string "versity" is added at the end of "Uni'

- StringBuffer insert(int i, x): x may be boolean, byte, int, long, float, double, char, chara array, String or another StringBuffer. It will be inserted into the StringBuffer at the posi represented by i. For example,

```
StringBuffer sb = new StringBuffer("Intelligent Person");
sb.insert(11, "young");
```

Here, the StringBuffer object sb contains "Intelligent Person". Counting from 0, we find the str "Intelligent" has 10 characters. So, sb.insert(11, "young") will insert the string, "young" at position, and hence we get "Intelligentyoung Person" in sb.

- StringBuffer delete(int i, int j): This removes the characters from ith position till j-1th posi in the StringBuffer. For example,

```
StringBuffer sb = new StringBuffer("University");
```

sb.delete(0,3) deletes the characters from the beginning to 2nd character ("Uni") and resultant string in 'sb' will be "versity".

- StringBuffer reverse(): This reverses the character sequence in the StringBuffe StringBuffer contains "abc", it becomes "cba".
- String toString(): This converts the StringBuffer object into a String object. This will enabl to use String class methods on StringBuffer object, after its conversion.
- int length(): This returns the number of characters in the StringBuffer object.
- int indexOf(String str): This returns the first occurrence of substring 'str' in StringBuffer object. For example,

```
StringBuffer sb = new StringBuffer("This is a book");
int n = sb.indexOf("is");
```

Observe the substring "is" occurred starting at 2nd position and 5th positions (counting fro indexOf() method returns first occurrence only. So the preceding statement gives 2.

- `int lastIndexOf(String str)`: This returns the last occurrence of substring 'str' in the StringBuffer object. For example,

```
StringBuffer sb = new StringBuffer("This is a book");
int n = sb.lastIndexOf("is");
```

Preceding statement returns 5, as the last occurrence of "is " is at 5th position.

- `StringBuffer replace(int i, int j, String str)`: This replaces the characters from i to j-1, by the string 'str' in the `StringBuffer` object. For example,

```
StringBuffer sb = new StringBuffer("High cost");
sb.replace(0,4, "Low");
```

We are replacing first 4 characters of sb by the 3 characters, "Low". Now sb contains, "Low cost".

- `String substring(int i)`: This retrieves a sub string from the StringBuffer object starting from ith position till the end. For example,

```
StringBuffer sb = new StringBuffer("New Delhi");
String s = sb.substring(4);
```

Preceding statement retrieves characters from 4th position (count from 0) till the end, and hence returns "Delhi" into s.

- `String substring(int i, int j)`: This extracts a sub string from the StringBuffer object starting from ith position to j-1th position. For example,

```
StringBuffer sb = new StringBuffer("New Delhi");
String s = sb.substring(0,3);
```

Here, starting from 0th character to 2nd character is extracted from sb and returned into s. So, s contains "New".

To understand some of the StringBuffer class methods, let us write a Java program. In this, we want to receive the name of a person in three parts as sur name, middle name and last name; and display the full name of the person.

Program 1: Write a program to learn how to use some of the `StringBuffer` class methods.

```
//To compose full name of a person
import java.io.*;
class Full
{
      public static void main(String args[])
      throws IOException
      {
            //create empty string buffer object
            StringBuffer sb = new StringBuffer();

            //to accept data from keyboard
            BufferedReader br = new BufferedReader(new

            InputStreamReader(System.in));

            //accept surname
            System.out.print("Enter surname: ");
            String sur = br.readLine();

            //accept middlename
            System.out.print("Enter midname: ");
            String mid = br.readLine();
```

```
            //accept lastname
            System.out.print("Enter lastname: ");
            String last = br.readLine();

            //append sur to sb
            sb.append(sur);

            //append last to sb
            sb.append(last);

            //display the name till now
            System.out.println("Name= "+ sb);

            //insert mid after sur name in sb
            int n =sur.length();   //n represents no. of chars in sur name
            sb.insert(n,mid);  //insert mid name after nth character

            //display full name
            System.out.println("Full name= "+ sb);

            //reverse and display the name
            System.out.println("In reverse= "+ sb.reverse());
        }
    }
```

Output:

```
C:\> javac Full.java
C:\> java Full
Enter surname: Mallipudi
Enter midname: Vijaya
Enter lastname: Lakshmi
Name= MallipudiLakshmi
Full name= MallipudiVijayaLakshmi
In reverse= imhskaLayajiVidupillaM
```

In Program 1, we are accepting sur name, middle name and last name of a person from the keyboard and storing them into sur, mid and last respectively. First of all, we are appending sur name and last name to StringBuffer object, as:

```
sb.append(sur);
sb.append(last);
```

After that, we are calculating the number of characters in sur name and inserting the middle name after those many characters, as:

```
int n =sur.length();
sb.insert(n,mid);
```

So, we got full name in sb in correct sequence: sur name, middle name and last name joined together one by one. This is depicted in the Figure 10.1.

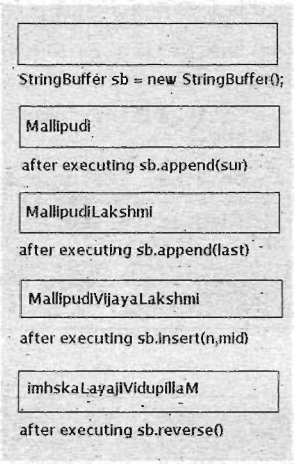

Figure 10.1 Execution sequence of Program 1

In this program, we can also directly add sur name, middle name and last name one by one. But we are first adding sur name and last name and then inserting the middle name after sur. Why? We did so just to demonstrate the functioning of `insert()` method of `StringBuffer()` class.

Let us write another program to check if a given string is a palindrome or not. If a string is palindrome, it gives same string even after reversing it. For example, malayalam, madam, peep, liril etc., are palindromes. The following steps can be followed to check the string:

- Let the string be '`str`'.
- Preserve a copy of the original string 'str' in another string 'temp'.
- Convert the string 'str 'into a `StringBuffer` object 'sb'. Now it is possible to use `StringBuffer` methods on sb.
- Reverse the string in the sb.
- Store the reversed string in 'str' again.
- Now, compare the original string in 'temp' with the reversed string 'str'. If they are equal, then the string is a palindrome, else it is not a palindrome.

The reason why we are converting a string into a `StringBuffer` is to use the `reverse()` method of `StringBuffer` class. Remember it is not possible to use the methods of a class on the objects of another class.

Program 2: Write a program for testing a string whether it is a palindrome or not.

```
//Palindrome or not
import java.io.*;
class Palindrome
{
    public static void main(String args[]) throws IOException
    {
        //accept the string from keyboard
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));

        System.out.print("Enter a string: ");
        String str = br.readLine();

        //store a copy of original string in temp
        String temp = str;

        //convert the string into StringBuffer
```

allows several threads to act on it simultaneously. This may lead to inaccurate results in some cases.

Synchronizing the object is like locking the object. This means, when a thread acts on the object, it is locked and any other thread should wait till the current thread completes and unlocks the object. Thus, synchronization does not allow more than 1 thread to act simultaneously on the object. Implementing the synchronization mechanism (like locking and unlocking of the object) will take some time for the JVM. Hence `StringBuffer` class will take more execution time than the `StringBuilder`, which does not implement this mechanism. We discuss more about threads later in the chapter on 'Threads'.

So, it is recommended that when a programmer wants to use a single thread as is the case generally, he should use `StringBuilder` class instead of `StringBuffer` class, to get faster execution.

Important Interview Question

What is the difference between StringBuffer and StringBuilder classes?

StringBuffer class is synchronized and StringBuilder is not. When the programmer wants to use several threads, he should use StringBuffer as it gives reliable results. If only one thread is used, StringBuilder is preferred, as it improves execution time.

Conclusion

StringBuffer and StringBuilder classes differ from String class in their mutability of objects. These classes provide methods which directly modify the contents of the objects, and hence these objects are called 'mutable'. Some methods like insert(), delete() and reverse() which are not available in String class are very valuable for the programmers.