

The StringBuffer Class

We noted earlier that `String` objects are "immutable" in Java. Once a `String` object is instantiated, it cannot change in size or content. Any change yields a new `String` object and the old one is discarded. Strings created with the `StringBuffer` class, however, are dynamic. Once created, characters can change and new characters can be inserted or deleted. Although these tasks are possible through the creative use of substringing and concatenation with `String` objects, there are performance benefits in using `StringBuffer` objects when such manipulations are frequent.

The `StringBuffer` class is part of the `java.lang` package. The most common methods are summarized in Table 1.

Table 1. Methods of the `StringBuffer` Class

Method	Description
Constructors	
<code>StringBuffer()</code>	constructs a <code>StringBuffer</code> object with no characters in it and an initial capacity of 16 characters; returns a reference to the new object
<code>StringBuffer(int length)</code>	constructs a <code>StringBuffer</code> object with no characters in it and an initial capacity specified by <code>length</code> ; returns a reference to the new object
<code>StringBuffer(String s)</code>	constructs a <code>StringBuffer</code> object so that it represents the same sequence of characters as the string <code>s</code> ; returns a reference to the new object
Instance Methods	
<code>append(char c)</code>	appends the character to the string buffer; returns a reference to this <code>StringBuffer</code>
<code>charAt(int i)</code>	returns the <code>char</code> at the specified index in the string buffer
<code>delete(int i, int j)</code>	deletes characters from positions <code>i</code> to <code>j - 1</code> ; returns a reference to this <code>StringBuffer</code>
<code>deleteCharAt(int i)</code>	deletes the character at position <code>i</code> ; returns a reference to this <code>StringBuffer</code>
<code>insert(int i, char c)</code>	inserts a character at position <code>i</code> ; returns a reference to this <code>StringBuffer</code>
<code>length()</code>	returns an <code>int</code> equal to the length (character count) of the string buffer
<code>replace(int i, int j, String s)</code>	removes characters from position <code>i</code> to <code>j - 1</code> , then inserts string <code>s</code> ; returns a reference to this <code>StringBuffer</code>
<code>reverse()</code>	reverses the characters in the string buffer; returns a reference to this <code>StringBuffer</code>
<code>setCharAt(int i, char c)</code>	sets the character at position <code>i</code> to the character <code>c</code> ; returns a reference to this <code>StringBuffer</code>
<code>substring(int i)</code>	returns a <code>String</code> object equal to a substring of the string buffer from position <code>i</code> to the end of the string buffer
<code>toString()</code>	returns a <code>String</code> object equivalent to this <code>StringBuffer</code>

Note that the `append()` and `insert()` methods are overloaded. They can accept any primitive data type as an argument, as well as character arrays or strings.

The compiler uses methods of the `StringBuffer` class to implement the concatenation operation for `String` objects, but this is transparent to programmers. As an example, the following statement

```
String s = "Y" + 2 + "K";
```

is compiled to the equivalent of

```
String s = new StringBuffer().append("Y").append(2).append("K").toString();
```

which creates a new string buffer (initially empty), and appends in turn the string representation of each operand. Then, the content of the string buffer is converted to a `String` object. Overall, this avoids creating many temporary strings.

Every string buffer has a capacity. As long as the number of characters does not exceed its capacity, all is well. However, if the internal buffer overflows due to an `append()` or `insert()` method, it is automatically made larger. This occurs in the background, and is transparent to the programmer or user.

A demo program using the `StringBuffer` class is presented in the next section.