

Name: _____ () Date: _____

Chapter 8: StringBuilder and StringBuffer Classes (Supplementary Reading)

8.9 The StringBuilder and StringBuffer Classes

The **StringBuilder** and **StringBuffer** classes are similar to the **String** class except that the **String** class is **immutable or unchangeable**. When you write `someString = "Hello"`; and follow it with `someString = "Goodbye"`;, you have neither change the contents in the computer memory at the address represented by `someString` nor eliminated the characters "Hello". Instead, you have store "Goodbye" at a new computer memory location and stored the new address in the `someString` variable. If you want to modify `someString` from "Goodbye" to "Goodbye Everybody", you cannot add a space and "Everybody" to the `someString` that contains "Goodbye". Instead, you must create an entire new `String` "Goodbye Everybody", and assign it to the `someString` address. If you perform many such operations with `Strings`, you end up creating many different `String` objects in memory, which takes time and resources.

To circumvent these limitations, you can use either the `StringBuilder` or `StringBuffer` class. In general, the `StringBuilder` and `StringBuffer` classes can be used wherever a string is used. `StringBuilder` and `StringBuffer` are more flexible than `String`. **You can add, insert, or append new contents into `StringBuilder` and `StringBuffer` objects, whereas the value of a `String` object is fixed once the string is created.** Like the `String` class, these two classes are part of the `java.lang` package and are automatically imported into every program.

The `StringBuilder` class is similar to `StringBuffer` except that the methods for modifying the buffer in `StringBuffer` are synchronized (thread safe), which means that only one task is allowed to execute the methods. Thus, you should use it in applications that run multiple threads of execution which are units of processing that are scheduled by an operation system and that can be used to create multiple paths of control during program execution. Use `StringBuffer` if the class might be accessed by multiple tasks concurrently, because synchronization is needed in this case to prevent corruptions to `StringBuffer`.

Using `StringBuilder` is more efficient if it is accessed by just a single task, because no synchronization is needed in this case. Because most programs that you write in this course contain a single thread, `StringBuilder` will be more than adequate for string manipulation.

The constructors and methods in `StringBuffer` and `StringBuilder` are almost the same. You can replace `StringBuffer` in all occurrences by `StringBuilder`. The program can compile and run without any other changes. The `StringBuilder` class has three constructors and more than 30 methods for managing the builder and modifying strings in the builder. You can create an empty string builder or a string builder from a string using the constructors, as shown in Figure 8.4.

java.lang.StringBuilder	
+StringBuilder() +StringBuilder(capacity: int) +StringBuilder(s: String)	Constructs an empty string builder with capacity 16. Constructs a string builder with the specified capacity. Constructs a string builder with the specified string.

Figure 8.4 The `StringBuilder` class constructors

8.9.1 Modifying Strings in the StringBuilder

java.lang.StringBuilder	
<pre> +append(data: char[]): StringBuilder +append(data: char[], offset: int, len: int): StringBuilder +append(v: aPrimitiveType): StringBuilder +append(s: String): StringBuilder +delete(startIndex: int, endIndex: int): StringBuilder +deleteCharAt(index: int): StringBuilder +insert(index: int, data: char[], offset: int, len: int): StringBuilder +insert(offset: int, data: char[]): StringBuilder +insert(offset: int, b: aPrimitiveType): StringBuilder +insert(offset: int, s: String): StringBuilder +replace(startIndex: int, endIndex: int, s: String): StringBuilder +reverse(): StringBuilder +setCharAt(index: int, ch: char): void </pre>	<p>Appends a char array into this string builder. Appends a subarray in data into this string builder.</p> <p>Appends a primitive type value as a string to this builder. Appends a string to this string builder.</p> <p>Deletes characters from <code>startIndex</code> to <code>endIndex-1</code>. Deletes a character at the specified index.</p> <p>Inserts a subarray of the data in the array into the builder at the specified index. Inserts data into this builder at the position offset.</p> <p>Inserts a value converted to a string into this builder. Inserts a string into this builder at the position offset.</p> <p>Replaces the characters in this builder from <code>startIndex</code> to <code>endIndex-1</code> with the specified string. Reverses the characters in the builder. Sets a new character at the specified index in this builder.</p>

Figure 8.5 The StringBuilder class contains the methods for modifying string builders.

You can append new contents at the end of a string builder, insert new contents at a specified position in a string builder, and delete or replace characters in a string builder, using the methods listed in Figure 8.5. The StringBuilder class provides several overloaded methods to append boolean, char, char[], double, float, int, long, and String into a string builder. For example, the following code appends strings and characters into stringBuilder to form a new string, `Welcome to Java`.

```

StringBuilder stringBuilder = new StringBuilder();
stringBuilder.append("Welcome");
stringBuilder.append(' ');
stringBuilder.append("to");
stringBuilder.append(' ');
stringBuilder.append("Java");

```

The StringBuilder class also contains overloaded methods to insert boolean, char, char array, double, float, int, long, and String into a string builder. Consider the following code:

```
stringBuilder.insert(11, "HTML and ");
```

Suppose stringBuilder contains `Welcome to Java` before the insert method is applied. This code inserts `"HTML and "` at position 11 in stringBuilder (just before the J). The new stringBuilder is `Welcome to HTML and Java`. You can also delete characters from a string in the builder using the two delete methods, reverse the string using the reverse method, replace characters using the replace method, or set a new character in a string using the `setCharAt()` method. For example, suppose stringBuilder contains `Welcome to Java` before each of the following methods is applied:

```

stringBuilder.delete(8, 11) changes the builder to Welcome Java.
stringBuilder.deleteCharAt(8) changes the builder to Welcome o Java.
stringBuilder.reverse() changes the builder to avaJ ot emocleW.
stringBuilder.replace(11, 15, "HTML") changes the builder to Welcome to HTML.
stringBuilder.setCharAt(0, 'w') sets the builder to welcome to Java.

```

All these modification methods except `setCharAt()` do two things:

- Change the contents of the string builder
- Return the reference of the string builder

For example, the following statement

```
StringBuilder stringBuilder1 = stringBuilder.reverse();
```

reverses the string in the builder and assigns the builder's reference to `stringBuilder1`. Thus, `stringBuilder` and `stringBuilder1` both point to the same `StringBuilder` object. Recall that a value-returning method can be invoked as a statement, if you are not interested in the return value of the method. In this case, the return value is simply ignored. For example, in the following statement `stringBuilder.reverse();` the return value is ignored.

8.9.2 The `toString`, `capacity`, `length`, `setLength`, and `charAt` Methods

The `StringBuilder` class provides the additional methods for manipulating a string builder and obtaining its properties, as shown in Figure 8.6.

java.lang.StringBuilder	
+ <code>toString(): String</code>	Returns a string object from the string builder.
+ <code>capacity(): int</code>	Returns the capacity of this string builder.
+ <code>charAt(index: int): char</code>	Returns the character at the specified index.
+ <code>length(): int</code>	Returns the number of characters in this builder.
+ <code>setLength(newLength: int): void</code>	Sets a new length in this builder.
+ <code>substring(startIndex: int): String</code>	Returns a substring starting at <code>startIndex</code> .
+ <code>substring(startIndex: int, endIndex: int): String</code>	Returns a substring from <code>startIndex</code> to <code>endIndex-1</code> .
+ <code>trimToSize(): void</code>	Reduces the storage size used for the string builder.

Figure 8.6 The `StringBuilder` class contains the methods for modifying string builders.

The `capacity()` method returns the current capacity of the string builder. The capacity is the number of characters the string builder is able to store without having to increase its size. The `length()` method returns the number of characters actually stored in the string builder. The `setLength(newLength)` method sets the length of the string builder. If the `newLength` argument is less than the current length of the string builder, the string builder is truncated to contain exactly the number of characters given by the `newLength` argument. If the `newLength` argument is greater than or equal to the current length, sufficient null characters (`\u0000`) are appended to the string builder so that length becomes the `newLength` argument. The `newLength` argument must be greater than or equal to 0.

The `charAt(index)` method returns the character at a specific index in the string builder. The index is 0 based. The first character of a string builder is at index 0, the next at index 1, and so on. The index argument must be greater than or equal to 0, and less than the length of the string builder. The length of the string is always less than or equal to the capacity of the builder. The length is the actual size of the string stored in the builder, and the capacity is the current size of the builder. The builder's capacity is automatically increased if more characters are added to exceed its capacity. Internally, a string builder is an array of characters, so the builder's capacity is the size of the array. If the builder's capacity is exceeded, the array is replaced by a new array. The new array size is $2 * (\text{the previous array size} + 1)$.

PROGRAM 8-7, considered all the characters in a string to check whether it is a palindrome. The program that ignores nonalphanumeric characters in checking whether a string is a palindrome.

1. Filter the string by removing the nonalphanumeric characters. This can be done by creating an empty string builder, adding each alphanumeric character in the string to a string builder, and returning the string from the string builder. You can use the `isLetterOrDigit(ch)` method in the `Character` class to check whether character `ch` is a letter or a digit.
2. Obtain a new string that is the reversal of the filtered string. Compare the reversed string with the filtered string using the `equals` method.

The `filter(String s)` method (lines 20–29) examines each character in string `s` and copies it to a string builder if the character is a letter or a numeric character. The `filter()` method returns the string in the builder. The `reverse(String s)` method (lines 31–35) creates a new string that reverses the specified string `s`. The `filter` and `reverse` methods both return a new string. The original string is not changed. PROGRAM 8-7 checks whether a string is a palindrome by comparing pairs of characters from both ends of the string. The program uses the `reverse` method in the `StringBuilder` class to reverse the string, then compares whether the two strings are equal to determine whether the original string is a palindrome.

<pre> 1 import java.util.Scanner; 2 public class PalindromeIgnoreNonAlphanumeric { 3 public static void main(String[] args) { 4 Scanner input = new Scanner(System.in); 5 System.out.print("Enter a string: "); 6 String s = input.nextLine(); 7 System.out.println("Ignoring nonalphanumeric characters, \nis " 8 + s + " a palindrome? " + isPalindrome(s)); 9 } 10 public static boolean isPalindrome(String s) { 11 // Create a new string by eliminating nonalphanumeric chars 12 String s1 = filter(s); 13 // Create a new string that is the reversal of s1 14 String s2 = reverse(s1); 15 // Check if the reversal is the same as the original string 16 return s2.equals(s1); 17 } 18 /** Create a new string by eliminating nonalphanumeric chars */ 19 public static String filter(String s) { 20 StringBuilder stringBuilder = new StringBuilder(); 21 for (int i = 0; i < s.length(); i++) { // Skip alphanumeric 22 if (Character.isLetterOrDigit(s.charAt(i))) { 23 stringBuilder.append(s.charAt(i)); 24 } 25 } 26 return stringBuilder.toString(); // Return a new filtered string 27 } 28 public static String reverse(String s){ // reverse string 29 StringBuilder stringBuilder = new StringBuilder(s); 30 stringBuilder.reverse(); // Invoke reverse in StringBuilder 31 return stringBuilder.toString(); 32 } 33 } </pre>	PROGRAM 8-9
--	--------------------

PROGRAM OUTPUT

```

Enter a string: ab<c>cb?a
Ignoring nonalphanumeric characters,
is ab<c>cb?a a palindrome? true

```

```

Enter a string: abcc><?cab
Ignoring nonalphanumeric characters,
is abcc><?cab a palindrome? false

```