

## 14.4 Class `StringBuilder`

We now discuss the features of class `StringBuilder` for creating and manipulating *dynamic* string information—that is, *modifiable* strings. Every `StringBuilder` is capable of storing a number of characters specified by its *capacity*. If a `StringBuilder`'s capacity is exceeded, the capacity expands to accommodate the additional characters.



### Performance Tip 14.4

*Java can perform certain optimizations involving `String` objects (such as referring to one `String` object from multiple variables) because it knows these objects will not change. `Strings` (not `StringBuilders`) should be used if the data will not change.*



### Performance Tip 14.5

*In programs that frequently perform string concatenation, or other string modifications, it's often more efficient to implement the modifications with class `StringBuilder`.*



# Software Engineering

## Observation 14.1

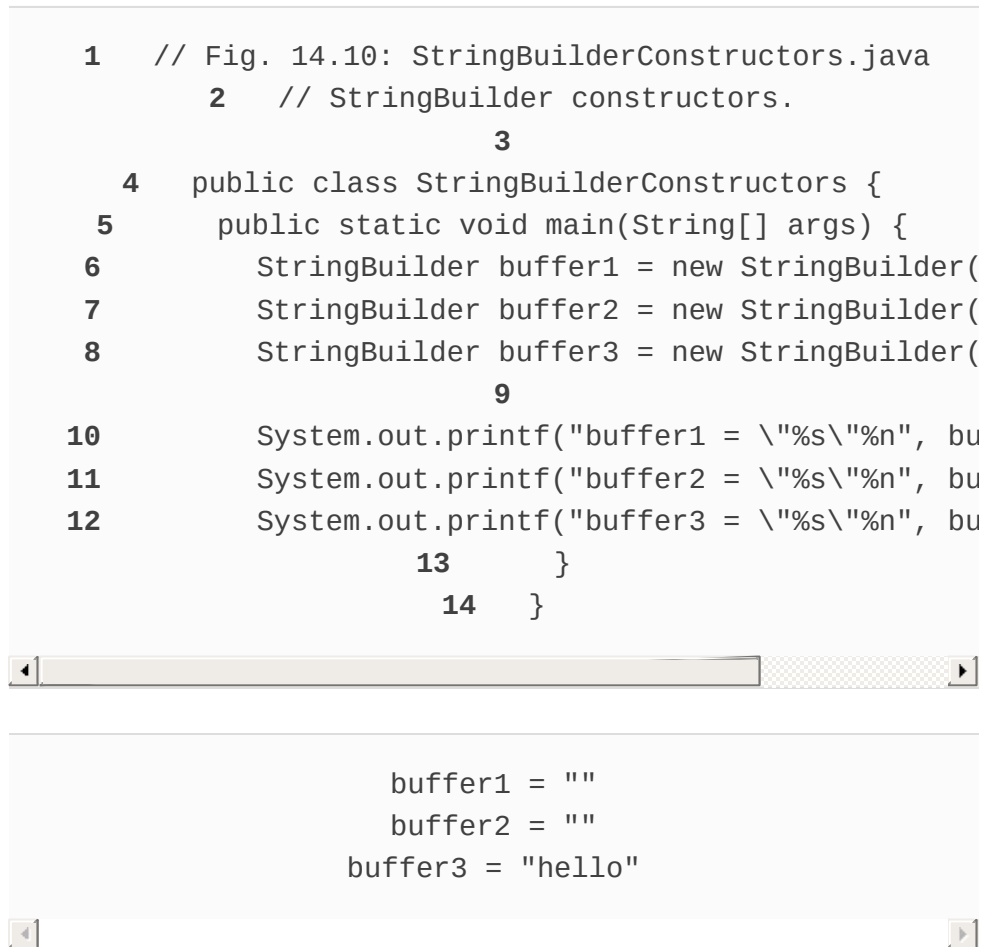
*StringBuilders are not thread safe. If multiple threads require access to the same dynamic string information, use class `StringBuffer` in your code. Classes `StringBuilder` and `StringBuffer` provide identical capabilities, but class `StringBuffer` is thread safe. For more details on threading, see [Chapter 23](#).*

### 14.4.1 `StringBuilder` Constructors

Class `StringBuilder` provides four constructors. We demonstrate three of these in [Fig. 14.10](#). Line 6 uses the no-argument `StringBuilder` constructor to create a `StringBuilder` with no characters in it and an initial capacity of 16 characters (the default for a `StringBuilder`). Line 7 uses the `StringBuilder` constructor that takes an integer argument to create a `StringBuilder` with no characters in it and the initial capacity specified by the integer argument (i.e., 10). Line 8 uses the `StringBuilder` constructor that takes a `String` argument to create a `StringBuilder` containing the characters in the `String` argument. The initial capacity is the number of characters in the `String` argument plus 16. Lines

10–12 implicitly use the method `toString` of class `StringBuilder` to output the `StringBuilders` with the `printf` method. In [Section 14.4.4](#), we discuss how Java uses `StringBuilder` objects to implement the `+` and `+=` operators for string concatenation.

```
1  // Fig. 14.10: StringBuilderConstructors.java
2  // StringBuilder constructors.
3
4  public class StringBuilderConstructors {
5      public static void main(String[] args) {
6          StringBuilder buffer1 = new StringBuilder(
7          StringBuilder buffer2 = new StringBuilder(
8          StringBuilder buffer3 = new StringBuilder(
9
10         System.out.printf("buffer1 = \"%s\\\"%n", bu
11         System.out.printf("buffer2 = \"%s\\\"%n", bu
12         System.out.printf("buffer3 = \"%s\\\"%n", bu
13     }
14 }
```



```
buffer1 = ""
buffer2 = ""
buffer3 = "hello"
```

**Fig. 14.10**

`StringBuilder` constructors.

## 14.4.2 StringBuilder Methods length, capacity, setLength and ensureCapacity

Class `StringBuilder`'s `length` and `capacity` method return the number of characters currently in a `StringBuilder` and the number of characters that can be stored without allocating more memory, respectively. Method `ensureCapacity` guarantees that a `StringBuilder` has at least the specified capacity. Method `setLength` increases or decreases the length of a `StringBuilder`. [Figure 14.11](#) demonstrates these methods.

```
1 // Fig. 14.11: StringBuilderCapLen.java
2 // StringBuilder length, setLength, capacity and
3
4 public class StringBuilderCapLen {
5     public static void main(String[] args) {
6         StringBuilder buffer = new StringBuilder("
7
8         System.out.printf("buffer = %s\nlength = %
9         buffer.toString(), buffer.length(), buf
10
11         buffer.ensureCapacity(75);
12         System.out.printf("New capacity = %d\n%n",
13
14         buffer.setLength(10));
15         System.out.printf("New length = %d\nbuffer
16         buffer.length(), buffer.toString());
17     }
```

```
18    }  
  
buffer = Hello, how are you?  
    length = 19  
    capacity = 35  
  
    New capacity = 75  
  
    New length = 10  
    buffer = Hello, how
```

Fig. 14.11

`StringBuilder` `length`, `setLength`, `capacity` and `ensureCapacity` methods.

The application contains one `StringBuilder` called `buffer`. Line 6 uses the `StringBuilder` constructor that takes a `String` argument to initialize the `StringBuilder` with "Hello, how are you?". Lines 8–9 print the contents, length and capacity of the `StringBuilder`. Note in the output window that the capacity of the `StringBuilder` is initially 35. Recall that the `StringBuilder` constructor that takes a `String` argument initializes the capacity to the length of the string passed as an argument plus 16.

Line 11 uses method `ensureCapacity` to expand the

capacity of the `StringBuilder` to a minimum of 75 characters. Actually, if the original capacity is less than the argument, the method ensures a capacity that's the greater of the number specified as an argument and twice the original capacity plus 2. The `StringBuilder`'s current capacity remains unchanged if it's more than the specified capacity.



## Performance Tip 14.6

*Dynamically increasing the capacity of a `StringBuilder` can take a relatively long time. Executing a large number of these operations can degrade the performance of an application. If a `StringBuilder` is going to increase greatly in size, possibly multiple times, setting its capacity high at the beginning will increase performance.*

Line 14 uses method `setLength` to set the length of the `StringBuilder` to 10. If the specified length is less than the current number of characters in the `StringBuilder`, its contents are truncated to the specified length (i.e., the characters in the `StringBuilder` after the specified length are discarded). If the specified length is greater than the number of characters currently in the `StringBuilder`, null characters (characters with the numeric representation 0) are appended until the total number of characters in the `StringBuilder` is equal to the specified length.

## 14.4.3 StringBuilder Methods charAt, setCharAt, getChars and reverse

Class `StringBuilder` provides methods `charAt`, `setCharAt`, `getChars` and `reverse` to manipulate the characters in a `StringBuilder` (Fig. 14.12). Method `charAt` (line 10) takes an integer argument and returns the character in the `StringBuilder` at that index. Method `getChars` (line 13) copies characters from a `StringBuilder` into the character array passed as an argument. This method takes four arguments—the starting index from which characters should be copied in the `StringBuilder`, the index one past the last character to be copied from the `StringBuilder`, the character array into which the characters are to be copied and the starting location in the character array where the first character should be placed. Method `setCharAt` (lines 20 and 21) takes an integer and a character argument and sets the character at the specified position in the `StringBuilder` to the character argument. Method `reverse` (line 24) reverses the contents of the `StringBuilder`. Attempting to access a character that's outside the bounds of a `StringBuilder` results in a `StringIndexOutOfBoundsException`.

```

2 // StringBuilder methods charAt, setCharAt, getCharAt
3
4 public class StringBuilderChars {
5     public static void main(String[] args) {
6         StringBuilder buffer = new StringBuilder("
7
8         System.out.printf("buffer = %s\n", buffer.
9         System.out.printf("Character at 0: %s\nCha
10        buffer.charAt(0), buffer.charAt(4));
11
12        char[] charArray = new char[buffer.length(
13        buffer.getChars(0, buffer.length(), charAr
14        System.out.print("The characters are: ");
15
16        for (char character : charArray) {
17            System.out.print(character);
18        }
19
20        buffer.setCharAt(0, 'H');
21        buffer.setCharAt(6, 'T');
22        System.out.printf("%n\nbuffer = %s", buffer.
23
24        buffer.reverse();
25        System.out.printf("%n\nbuffer = %s\n", buffer.
26    }
27 }

```

```

buffer = hello there
Character at 0: h
Character at 4: o

```

The characters are: hello there

```
buffer = Hello There
```

```
buffer = erehT olleH
```



## Fig. 14.12

`StringBuilder` methods `charAt`, `setCharAt`,  
`getChars` and `reverse`.

### 14.4.4 `StringBuilder` append Methods

Class `StringBuilder` provides *overloaded* append methods (Fig. 14.13) to allow values of various types to be appended to the end of a `StringBuilder`. Versions are provided for each of the primitive types and for character arrays, `Strings`, `Objects`, and more. (Remember that method `toString` produces a string representation of any `Object`.) Each method takes its argument, converts it to a string and appends it to the `StringBuilder`. The call `System.getProperty("line.separator")` returns a platform-independent newline.

```
1  // Fig. 14.13: StringBuilderAppend.java
2  // StringBuilder append methods.
3
4  public class StringBuilderAppend
5  {
6      public static void main(String[] args)
7      {
8          Object objectRef = "hello";
9          String string = "goodbye";
10         char[] charArray = {'a', 'b', 'c', 'd', 'e'};
11         boolean booleanValue = true;
```

```
12         char characterValue = 'Z';
13         int integerValue = 7;
14         long longValue = 100000000000L;
15         float floatValue = 2.5f;
16         double doubleValue = 33.333;
17
18     StringBuilder lastBuffer = new StringBuild
19     StringBuilder buffer = new StringBuilder()
20
21         buffer.append(objectRef)
22         .append(System.getProperty("line.sep
23         .append(string)
24         .append(System.getProperty("line.sep
25         .append(charArray)
26         .append(System.getProperty("line.sep
27         .append(charArray, 0, 3)
28         .append(System.getProperty("line.sep
29         .append(booleanValue)
30         .append(System.getProperty("line.sep
31         .append(characterValue);
32         .append(System.getProperty("line.sep
33         .append(integerValue)
34         .append(System.getProperty("line.sep
35         .append(longValue)
36         .append(System.getProperty("line.sep
37         .append(floatValue)
38         .append(System.getProperty("line.sep
39         .append(doubleValue)
40         .append(System.getProperty("line.sep
41         .append(lastBuffer);
42
43     System.out.printf("buffer contains%n%s%n",
44         }
45     }
```

buffer contains  
hello  
goodbye

```
abcdef
abc
true
Z
7
100000000000
2.5
33.333 last buffer
```

Fig. 14.13

`StringBuilder` append methods.

The compiler can use `StringBuilder` and the append methods to implement the `+` and `+=` `String` concatenation operators. For example, assuming the declarations

```
String string1 = "hello";
String string2 = "BC";
int value = 22;
```

the statement

```
String s = string1 + string2 + value;
```

concatenates "hello", "BC" and 22. The concatenation can be performed as follows:

```
String s = new StringBuilder().append("hello").append(
    append(22).toString());
```

First, the preceding statement creates an *empty* `StringBuilder`, then appends to it the strings "hello" and "BC" and the integer 22. Next, `StringBuilder`'s `toString` method converts the `StringBuilder` to a `String` object to be assigned to `String s`. The statement

```
s += "!";
```

can be performed as follows (this may differ by compiler):

```
s = new StringBuilder().append(s).append("!").toString();
```

This creates an empty `StringBuilder`, then appends to it the current contents of `s` followed by "!". Next, `StringBuilder`'s method `toString` (which must be called *explicitly* here) returns the `StringBuilder`'s contents as a `String`, and the result is assigned to `s`.

## 14.4.5 `StringBuilder` Insertion and Deletion Methods

`StringBuilder` provides overloaded `insert` methods to insert values of various types at any position in a `StringBuilder`. Versions are provided for the primitive types and for character arrays, `Strings`, `Objects` and `CharSequences`. Each method takes its second argument and inserts it at the index specified by the first argument. If the first argument is less than 0 or greater than the `StringBuilder`'s length, a `StringIndexOutOfBoundsException` occurs. Class `StringBuilder` also provides methods `delete` and `deleteCharAt` to delete characters at any position in a `StringBuilder`. Method `delete` takes two arguments—the starting index and the index one past the end of the characters to delete. All characters beginning at the starting index up to but *not* including the ending index are deleted. Method `deleteCharAt` takes one argument—the index of the character to delete. Invalid indices cause both methods to throw a `StringIndexOutOfBoundsException`. [Figure 14.14](#) demonstrates methods `insert`, `delete` and `deleteCharAt`.

```
1 // Fig. 14.14: StringBuilderInsertDelete.java
2 // StringBuilder methods insert, delete and deleteCharAt
3
4 public class StringBuilderInsertDelete {
5     public static void main(String[] args) {
6         Object objectRef = "hello";
7         String string = "goodbye";
8         char[] charArray = {'a', 'b', 'c', 'd', 'e'};
9         boolean booleanValue = true;
10        char characterValue = 'K';
11        int integerValue = 7;
```

```

12         long longValue = 100000000;
13     float floatValue = 2.5f; // f suffix indic
14         double doubleValue = 33.333;
15
16     StringBuilder buffer = new StringBuilder()
17
18         buffer.insert(0, objectRef);
19     buffer.insert(0, " "); // each of these c
20         buffer.insert(0, string);
21         buffer.insert(0, " ");
22         buffer.insert(0, charArray);
23         buffer.insert(0, " ");
24     buffer.insert(0, charArray, 3, 3);
25         buffer.insert(0, " ");
26         buffer.insert(0, booleanValue);
27         buffer.insert(0, " ");
28     buffer.insert(0, characterValue);
29         buffer.insert(0, " ");
30     buffer.insert(0, integerValue);
31         buffer.insert(0, " ");
32     buffer.insert(0, longValue);
33         buffer.insert(0, " ");
34     buffer.insert(0, floatValue);
35         buffer.insert(0, " ");
36     buffer.insert(0, doubleValue);
37
38     System.out.printf(
39     "buffer after inserts:%n%s%n", buffer.
40
41     buffer.deleteCharAt(10); // delete 5 in 2.
42     buffer.delete(2, 6); // delete .333 in 33.
43
44     System.out.printf(
45     "buffer after deletes:%n%s%n", buffer.t
46     }
47     }

```

buffer after inserts:

```
33.333 2.5 10000000 7 K true def abcdef goodb  
buffer after deletes:  
33 2. 10000000 7 K true def abcdef goodbye h
```



Fig. 14.14

StringBuilder methods insert, delete and  
deleteCharAt.