```
1  // Fig. 10.1: StringConstructors.java
2  // This program demonstrates the String class constructors.
3  import javax.swing.*;
4
5  public class StringConstructors {
6     public static void main( String args[] )
7     {
8        char charArray[] = { 'b', 'i', 'r', 't', 'h', ' ',
9                             'd', 'a', 'y' };
10       byte byteArray[] = { (byte) 'n', (byte) 'e', (byte) 'w',
11                            (byte) ' ', (byte) 'y', (byte) 'e',
12                            (byte) 'a', (byte) 'r' };
13       StringBuffer buffer;
14       String s, s1, s2, s3, s4, s5, s6, s7, output;
15
16       s = new String( "hello" );
17       buffer =
18          new StringBuffer( "Welcome to Java Programming!" );
19
20       // use the String constructors
21       s1 = new String();
22       s2 = new String( s );
23       s3 = new String( charArray );
24       s4 = new String( charArray, 6, 3 );
25       s5 = new String( byteArray, 4, 4 );
26       s6 = new String( byteArray );
27       s7 = new String( buffer );
28
29       output = "s1 = " + s1 +
30                "\ns2 = " + s2 +
31                "\ns3 = " + s3 +
32                "\ns4 = " + s4 +
33                "\ns5 = " + s5 +
34                "\ns6 = " + s6 +
35                "\ns7 = " + s7;
36
37       JOptionPane.showMessageDialog( null, output,
38          "Demonstrating String Class Constructors",
39          JOptionPane.INFORMATION_MESSAGE );
40
41       System.exit( 0 );
42    }
43 }
```

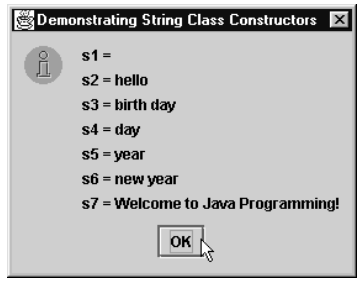

Fig. 10.1 Demonstrating the `String` class constructors.

```
1  // Fig. 10.2: StringMisc.java
2  // This program demonstrates the length, charAt and getChars
3  // methods of the String class.
4  //
5  // Note: Method getChars requires a starting point
6  // and ending point in the String. The starting point is the
7  // actual subscript from which copying starts. The ending point
8  // is one past the subscript at which the copying ends.
9  import javax.swing.*;
10
11 public class StringMisc {
12    public static void main( String args[] )
13    {
14       String s1, output;
15       char charArray[];
16
17       s1 = new String( "hello there" );
18       charArray = new char[ 5 ];
19
20       // output the string
21       output = "s1: " + s1;
22
23       // test the length method
24       output += "\nLength of s1: " + s1.length();
25
26       // loop through the characters in s1 and display reversed
27       output += "\nThe string reversed is: ";
28
29       for ( int i = s1.length() - 1; i >= 0; i-- )
30          output += s1.charAt( i ) + " ";
31
32       // copy characters from string into char array
33       s1.getChars( 0, 5, charArray, 0 );
34       output += "\nThe character array is: ";
35
36       for ( int i = 0; i < charArray.length; i++ )
37          output += charArray[ i ];
38
39       JOptionPane.showMessageDialog( null, output,
40          "Demonstrating String Class Constructors",
41          JOptionPane.INFORMATION_MESSAGE );
42
43       System.exit( 0 );
44    }
45 }
```

Fig. 10.2 The `String` class character manipulation methods.

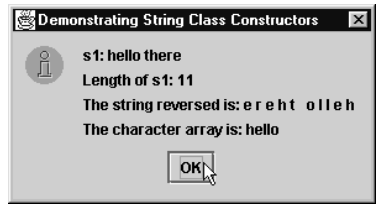

Fig. 10.2 The `String` class character manipulation methods.

```
1  // Fig. 10.3: StringCompare
2  // This program demonstrates the methods equals,
3  // equalsIgnoreCase, compareTo, and regionMatches
4  // of the String class.
5  import javax.swing.JOptionPane;
6
7  public class StringCompare {
8     public static void main( String args[] )
9     {
10       String s1, s2, s3, s4, output;
11
12       s1 = new String( "hello" );
13       s2 = new String( "good bye" );
14       s3 = new String( "Happy Birthday" );
15       s4 = new String( "happy birthday" );
16
17       output = "s1 = " + s1 + "\ns2 = " + s2 +
18                "\ns3 = " + s3 + "\ns4 = " + s4 + "\n\n";
19
20       // test for equality
21       if ( s1.equals( "hello" ) )
22          output += "s1 equals \"hello\"\n";
23       else
24          output += "s1 does not equal \"hello\"\n";
25
26       // test for equality with ==
27       if ( s1 == "hello" )
28          output += "s1 equals \"hello\"\n";
29       else
30          output += "s1 does not equal \"hello\"\n";
31
32       // test for equality--ignore case
33       if ( s3.equalsIgnoreCase( s4 ) )
34          output += "s3 equals s4\n";
35       else
36          output += "s3 does not equal s4\n";
37
38       // test compareTo
39       output +=
40          "\ns1.compareTo( s2 ) is " + s1.compareTo( s2 ) +
41          "\ns2.compareTo( s1 ) is " + s2.compareTo( s1 ) +
42          "\ns1.compareTo( s1 ) is " + s1.compareTo( s1 ) +
43          "\ns3.compareTo( s4 ) is " + s3.compareTo( s4 ) +
44          "\ns4.compareTo( s3 ) is " + s4.compareTo( s3 ) +
45          "\n\n";
46
47       // test regionMatches (case sensitive)
48       if ( s3.regionMatches( 0, s4, 0, 5 ) )
49          output += "First 5 characters of s3 and s4 match\n";
50       else
51          output +=
52             "First 5 characters of s3 and s4 do not match\n";
53
```

```
54          // test regionMatches (ignore case)
55          if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
56             output += "First 5 characters of s3 and s4 match";
57          else
58             output +=
59                "First 5 characters of s3 and s4 do not match";
60
61          JOptionPane.showMessageDialog( null, output,
62             "Demonstrating String Class Constructors",
63             JOptionPane.INFORMATION_MESSAGE );
64
65          System.exit( 0 );
66       }
67    }
```

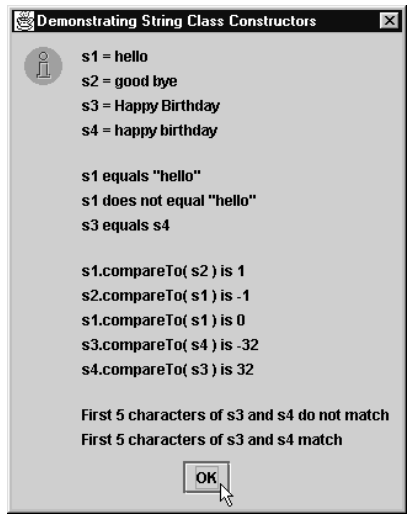

Fig. 10.3 Demonstrating `String` comparisons.

```
1  // Fig. 10.4: StringStartEnd.java
2  // This program demonstrates the methods startsWith and
3  // endsWith of the String class.
4  import javax.swing.*;
5
6  public class StringStartEnd {
7     public static void main( String args[] )
8     {
9        String strings[] = { "started", "starting",
10                            "ended", "ending" };
11       String output = "";
12
13       // Test method startsWith
14       for ( int i = 0; i < strings.length; i++ )
15          if ( strings[ i ].startsWith( "st" ) )
16             output += "\"" + strings[ i ] +
17                       "\" starts with \"st\"\n";
18
19       output += "\n";
20
21       // Test method startsWith starting from position
22       // 2 of the string
23       for ( int i = 0; i < strings.length; i++ )
24          if ( strings[ i ].startsWith( "art", 2 ) )
25             output +=
26                "\"" + strings[ i ] +
27                "\" starts with \"art\" at position 2\n";
28
29       output += "\n";
30
31       // Test method endsWith
32       for ( int i = 0; i < strings.length; i++ )
33          if ( strings[ i ].endsWith( "ed" ) )
34             output += "\"" + strings[ i ] +
35                       "\" ends with \"ed\"\n";
36
37       JOptionPane.showMessageDialog( null, output,
38          "Demonstrating String Class Comparisons",
39          JOptionPane.INFORMATION_MESSAGE );
40
41       System.exit( 0 );
42    }
43 }
```

Fig. 10.4 `String` class `startsWith` and `endsWith` methods (part 1 of 2).

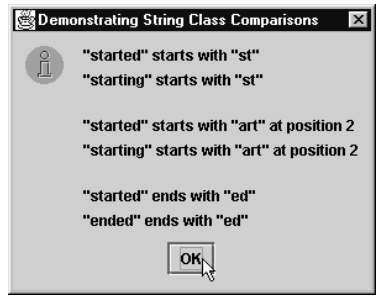

Fig. 10.4 `String` class `startsWith` and `endsWith` methods (part 2 of 2).

```
1  // Fig. 10.5: StringHashCode.java1
2  // This program demonstrates the method
3  // hashCode of the String class.
4  import javax.swing.*;
5
6  public class StringHashCode {
7     public static void main( String args[] )
8     {
9        String s1 = "hello",
10              s2 = "Hello";
11
12       String output =
13          "The hash code for \"" + s1 + "\" is " +
14          s1.hashCode() +
15          "\nThe hash code for \"" + s2 + "\" is " +
16          s2.hashCode();
17
18       JOptionPane.showMessageDialog( null, output,
19          "Demonstrating String Method hashCode",
20          JOptionPane.INFORMATION_MESSAGE );
21
22       System.exit( 0 );
23    }
24 }
```

Fig. 10.5 The `String` class `hashCode` method.

```
1  // Fig. 10.6: StringIndexMethods.java
2  // This program demonstrates the String
3  // class index methods.
4  import javax.swing.*;
5
6  public class StringIndexMethods {
7     public static void main( String args[] )
8     {
9        String letters = "abcdefghijklmabcdefghijklm";
10       String output;
11
12       // test indexOf to locate a character in a string
13       output = "'c' is located at index " +
14                letters.indexOf( 'c' );
15
16       output += "\n'a' is located at index " +
17                 letters.indexOf( 'a', 1 );
18
19       output += "\n'$' is located at index " +
20                 letters.indexOf( '$' );
21
22       // test lastIndexOf to find a character in a string
23       output += "\n\nLast 'c' is located at index " +
24                 letters.lastIndexOf( 'c' );
25
26       output += "\nLast 'a' is located at index " +
27                 letters.lastIndexOf( 'a', 25 );
28
29       output += "\nLast '$' is located at index " +
30                 letters.lastIndexOf( '$' );
31
32       // test indexOf to locate a substring in a string
33       output += "\n\n\"def\" is located at index " +
34                 letters.indexOf( "def" );
35
36       output += "\n\"def\" is located at index " +
37                 letters.indexOf( "def", 7 );
38
39       output += "\n\"hello\" is located at index " +
40                 letters.indexOf( "hello" );
41
42       // test lastIndexOf to find a substring in a string
43       output += "\n\nLast \"def\" is located at index " +
44                 letters.lastIndexOf( "def" );
45
46       output += "\nLast \"def\" is located at index " +
47                 letters.lastIndexOf( "def", 25 );
48
49       output += "\nLast \"hello\" is located at index " +
50                 letters.lastIndexOf( "hello" );
51
52       JOptionPane.showMessageDialog( null, output,
53          "Demonstrating String Class \"index\" Methods",
```

```
54             JOptionPane.INFORMATION_MESSAGE );
55
56          System.exit( 0 );
57       }
58    }
```

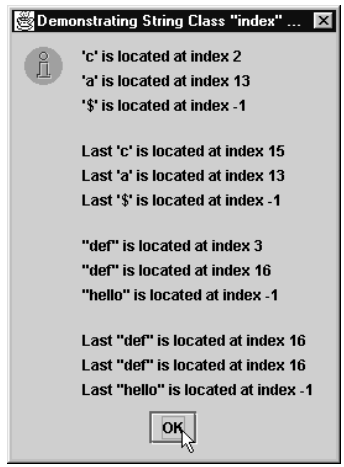

Fig. 10.6 The `String` class searching methods.

```
1  // Fig. 10.7: SubString.java
2  // This program demonstrates the
3  // String class substring methods.
4  import javax.swing.*;
5
6  public class SubString {
7     public static void main( String args[] )
8     {
9        String letters = "abcdefghijklmabcdefghijklm";
10       String output;
11
12       // test substring methods
13       output = "Substring from index 20 to end is " +
14                "\"" + letters.substring( 20 ) + "\"\n";
15
16       output += "Substring from index 0 up to 6 is " +
17                 "\"" + letters.substring( 0, 6 ) + "\"";
18
19       JOptionPane.showMessageDialog( null, output,
20          "Demonstrating String Class Substring Methods",
21          JOptionPane.INFORMATION_MESSAGE );
22
23       System.exit( 0 );
24    }
25 }
```

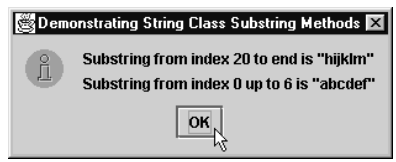

Fig. 10.7 The `String` class `substring` methods.

```
1  // Fig. 10.8: StringConcat.java
2  // This program demonstrates the String class concat method.
3  // Note that the concat method returns a new String object. It
4  // does not modify the object that invoked the concat method.
5  import javax.swing.*;
6
7  public class StringConcat {
8     public static void main( String args[] )
9     {
10       String s1 = new String( "Happy " ),
11              s2 = new String( "Birthday" ),
12              output;
13
14       output = "s1 = " + s1 +
15                "\ns2 = " + s2;
16
17       output += "\n\nResult of s1.concat( s2 ) = " +
18                 s1.concat( s2 );
19
20       output += "\ns1 after concatenation = " + s1;
21
22       JOptionPane.showMessageDialog( null, output,
23          "Demonstrating String Method concat",
24          JOptionPane.INFORMATION_MESSAGE );
25
26       System.exit( 0 );
27    }
28 }
```

Fig. 10.8 The `String` method `concat` (part 1 of 2).

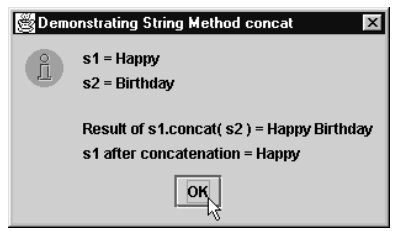

Fig. 10.8 The `String` method `concat` (part 2 of 2).

```
1  // Fig. 10.9: StringMisc2.java
2  // This program demonstrates the String methods replace,
3  // toLowerCase, toUpperCase, trim, toString and toCharArray
4  import javax.swing.*;
5
6  public class StringMisc2 {
7     public static void main( String args[] )
8     {
9        String s1 = new String( "hello" ),
10              s2 = new String( "GOOD BYE" ),
11              s3 = new String( "   spaces   " ),
12              output;
13
14       output = "s1 = " + s1 +
15                "\ns2 = " + s2 +
16                "\ns3 = " + s3;
17
18       // test method replace
19       output += "\n\nReplace 'l' with 'L' in s1: " +
20                 s1.replace( 'l', 'L' );
21
22       // test toLowerCase and toUpperCase
23       output +=
24          "\n\ns1.toUpperCase() = " + s1.toUpperCase() +
25          "\ns2.toLowerCase() = " + s2.toLowerCase();
26
27       // test trim method
28       output += "\n\ns3 after trim = \"" + s3.trim() + "\"";
29
30       // test toString method
31       output += "\n\ns1 = " + s1.toString();
32
33       // test toCharArray method
34       char charArray[] = s1.toCharArray();
35       output += "\n\ns1 as a character array = ";
36
37       for ( int i = 0; i < charArray.length; ++i )
38          output += charArray[ i ];
39
40       JOptionPane.showMessageDialog( null, output,
41          "Demonstrating Miscellaneous String Methods",
42          JOptionPane.INFORMATION_MESSAGE );
43
44       System.exit( 0 );
45    }
46 }
```

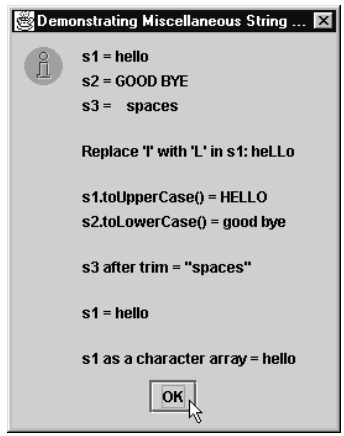

Fig. 10.9 Miscellaneous **`String`** methods.

```
1  // Fig. 10.10: StringValueOf.java
2  // This program demonstrates the String class valueOf methods.
3  import javax.swing.*;
4
5  public class StringValueOf {
6     public static void main( String args[] )
7     {
8        char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
9        boolean b = true;
10       char c = 'Z';
11       int i = 7;
12       long l = 10000000;
13       float f = 2.5f;
14       double d = 33.333;
15       Object o = "hello";  // Assign to an Object reference
16       String output;
17
18       output = "char array = " + String.valueOf( charArray ) +
19                "\npart of char array = " +
20                String.valueOf( charArray, 3, 3 ) +
21                "\nboolean = " + String.valueOf( b ) +
22                "\nchar = " + String.valueOf( c ) +
23                "\nint = " + String.valueOf( i ) +
24                "\nlong = " + String.valueOf( l ) +
25                "\nfloat = " + String.valueOf( f ) +
26                "\ndouble = " + String.valueOf( d ) +
27                "\nObject = " + String.valueOf( o );
28
29       JOptionPane.showMessageDialog( null, output,
30          "Demonstrating String Class valueOf Methods",
31          JOptionPane.INFORMATION_MESSAGE );
32
33       System.exit( 0 );
34    }
35 }
```

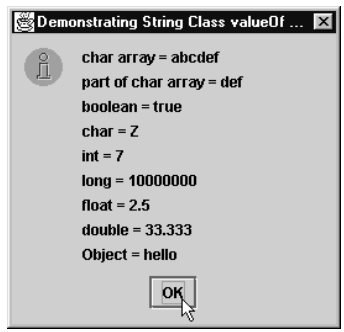

Fig. 10.10 The `String` class `valueOf` methods.

```
1  // Fig. 10.11: StringIntern.java
2  // This program demonstrates the intern method
3  // of the String class.
4  import javax.swing.*;
5
6  public class StringIntern {
7     public static void main( String args[] )
8     {
9        String s1, s2, s3, s4, output;
10
11       s1 = new String( "hello" );
12       s2 = new String( "hello" );
13
14       // Test strings to determine if they are the same
15       // String object in memory.
16       if ( s1 == s2 )
17          output =
18             "s1 and s2 are the same object in memory";
19       else
20          output =
21             "s1 and s2 are not the same object in memory";
22
23       // Test strings for equality of contents
24       if ( s1.equals( s2 ) )
25          output += "\ns1 and s2 are equal";
26       else
27          output += "\ns1 and s2 are not equal";
28
29       // Use String intern method to get a unique copy of
30       // "hello" referred to by both s3 and s4.
31       s3 = s1.intern();
32       s4 = s2.intern();
33
34       // Test strings to determine if they are the same
35       // String object in memory.
36       if ( s3 == s4 )
37          output +=
38             "\ns3 and s4 are the same object in memory";
39       else
40          output +=
41             "\ns3 and s4 are not the same object in memory";
42
43       // Determine if s1 and s3 refer to the same object
44       if ( s1 == s3 )
45          output +=
46             "\ns1 and s3 are the same object in memory";
47       else
48          output +=
49             "\ns1 and s3 are not the same object in memory";
50
51       // Determine if s2 and s4 refer to the same object
52       if ( s2 == s4 )
53          output +=
54             "\ns2 and s4 are the same object in memory";
```

```
55          else
56             output +=
57                "\ns2 and s4 are not the same object in memory";
58
59          // Determine if s1 and s4 refer to the same object
60          if ( s1 == s4 )
61             output +=
62                "\ns1 and s4 are the same object in memory";
63          else
64             output +=
65                "\ns1 and s4 are not the same object in memory";
66
67          JOptionPane.showMessageDialog( null, output,
68             "Demonstrating String Method intern",
69             JOptionPane.INFORMATION_MESSAGE );
70
71          System.exit( 0 );
72       }
73    }
```

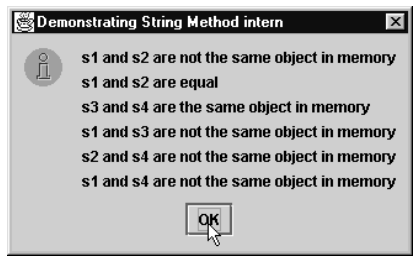

Fig. 10.11 The `String` class `intern` method.

```
1  // Fig. 10.12: StringBufferConstructors.java
2  // This program demonstrates the StringBuffer constructors.
3  import javax.swing.*;
4
5  public class StringBufferConstructors {
6     public static void main( String args[] )
7     {
8        StringBuffer buf1, buf2, buf3;
9
10       buf1 = new StringBuffer();
11       buf2 = new StringBuffer( 10 );
12       buf3 = new StringBuffer( "hello" );
13
14       String output =
15          "buf1 = " + "\"" + buf1.toString() + "\"" +
16          "\nbuf2 = " + "\"" + buf2.toString() + "\"" +
17          "\nbuf3 = " + "\"" + buf3.toString() + "\"";
18
19       JOptionPane.showMessageDialog( null, output,
20          "Demonstrating StringBuffer Class Constructors",
21          JOptionPane.INFORMATION_MESSAGE );
22
23       System.exit( 0 );
24    }
25 }
```

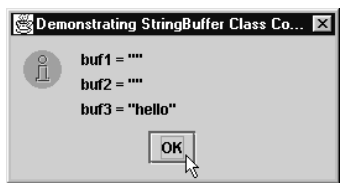

Fig. 10.12 The **`StringBuffer`** class constructors.

```
1  // Fig. 10.13: StringBufferCapLen.java
2  // This program demonstrates the length and
3  // capacity methods of the StringBuffer class.
4  import javax.swing.*;
5
6  public class StringBufferCapLen {
7     public static void main( String args[] )
8     {
9        StringBuffer buf =
10          new StringBuffer( "Hello, how are you?" );
11
12       String output = "buf = " + buf.toString() +
13                       "\nlength = " + buf.length() +
14                       "\ncapacity = " + buf.capacity();
15
16       buf.ensureCapacity( 75 );
17       output += "\n\nNew capacity = " + buf.capacity();
18
19       buf.setLength( 10 );
20       output += "\n\nNew length = " + buf.length() +
21                 "\nbuf = " + buf.toString();
22
23       JOptionPane.showMessageDialog( null, output,
24          "StringBuffer length and capacity Methods",
25          JOptionPane.INFORMATION_MESSAGE );
26
27       System.exit( 0 );
28    }
29 }
```

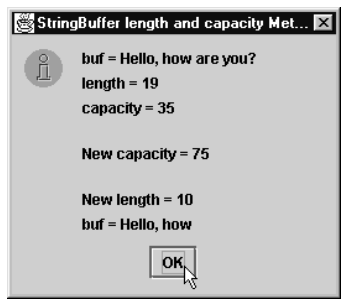

Fig. 10.13 `StringBuffer` `length` and `capacity` methods.

```
// Fig. 10.14: StringBufferChars.java
// The charAt, setCharAt, getChars, and reverse methods
// of class StringBuffer.
import javax.swing.*;

public class StringBufferChars {
   public static void main( String args[] )
   {
      StringBuffer buf = new StringBuffer( "hello there" );

      String output = "buf = " + buf.toString() +
                      "\nCharacter at 0: " + buf.charAt( 0 ) +
                      "\nCharacter at 4: " + buf.charAt( 4 );

      char charArray[] = new char[ buf.length() ];
      buf.getChars( 0, buf.length(), charArray, 0 );
      output += "\n\nThe characters are: ";

      for ( int i = 0; i < charArray.length; ++i )
         output += charArray[ i ];

      buf.setCharAt( 0, 'H' );
      buf.setCharAt( 6, 'T' );
      output += "\n\nbuf = " + buf.toString();

      buf.reverse();
      output += "\n\nbuf = " + buf.toString();

      JOptionPane.showMessageDialog( null, output,
         "Demonstrating StringBuffer Character Methods",
         JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

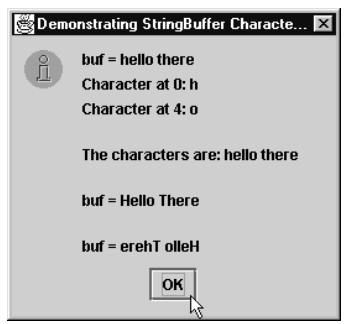

Fig. 10.14 `StringBuffer` class character manipulation methods.

```
1  // Fig. 10.15: StringBufferAppend.java
2  // This program demonstrates the append
3  // methods of the StringBuffer class.
4  import javax.swing.*;
5
6  public class StringBufferAppend {
7     public static void main( String args[] )
8     {
9        Object o = "hello";
10       String s = "good bye";
11       char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
12       boolean b = true;
13       char c = 'Z';
14       int i = 7;
15       long l = 10000000;
16       float f = 2.5f;
17       double d = 33.333;
18       StringBuffer buf = new StringBuffer();
19
20       buf.append( o );
21       buf.append( "  " );
22       buf.append( s );
23       buf.append( "  " );
24       buf.append( charArray );
25       buf.append( "  " );
26       buf.append( charArray, 0, 3 );
27       buf.append( "  " );
28       buf.append( b );
29       buf.append( "  " );
30       buf.append( c );
31       buf.append( "  " );
32       buf.append( i );
33       buf.append( "  " );
34       buf.append( l );
35       buf.append( "  " );
36       buf.append( f );
37       buf.append( "  " );
38       buf.append( d );
39
40       JOptionPane.showMessageDialog( null,
41          "buf = " + buf.toString(),
42          "Demonstrating StringBuffer append Methods",
43          JOptionPane.INFORMATION_MESSAGE );
44
45       System.exit( 0 );
46    }
47 }
```

Fig. 10.15 The `StringBuffer` class `append` methods.

Fig. 10.15 The **`StringBuffer`** class **`append`** methods

```
1  // Fig. 10.16: StringBufferInsert.java
2  // This program demonstrates the insert and delete
3  // methods of class StringBuffer.
4  import javax.swing.*;
5
6  public class StringBufferInsert {
7     public static void main( String args[] )
8     {
9        Object o = "hello";
10       String s = "good bye";
11       char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
12       boolean b = true;
13       char c = 'K';
14       int i = 7;
15       long l = 10000000;
16       float f = 2.5f;
17       double d = 33.333;
18       StringBuffer buf = new StringBuffer();
19
20       buf.insert( 0, o );
21       buf.insert( 0, "  " );
22       buf.insert( 0, s );
23       buf.insert( 0, "  " );
24       buf.insert( 0, charArray );
25       buf.insert( 0, "  " );
26       buf.insert( 0, b );
27       buf.insert( 0, "  " );
28       buf.insert( 0, c );
29       buf.insert( 0, "  " );
30       buf.insert( 0, i );
31       buf.insert( 0, "  " );
32       buf.insert( 0, l );
33       buf.insert( 0, "  " );
34       buf.insert( 0, f );
35       buf.insert( 0, "  " );
36       buf.insert( 0, d );
37
38       String output = "buf after inserts:\n" + buf.toString();
39
40       buf.deleteCharAt( 10 );      // delete 5 in 2.5
41       buf.delete( 2, 6 );          // delete .333 in 33.333
42
43       output += "\n\nbuf after deletes:\n" + buf.toString();
44
45       JOptionPane.showMessageDialog( null, output,
46          "Demonstrating StringBufferer Inserts and Deletes",
47          JOptionPane.INFORMATION_MESSAGE );
48
49       System.exit( 0 );
50    }
51 }
```

Fig. 10.16 The **`StringBuffer`** class **`insert`** methods

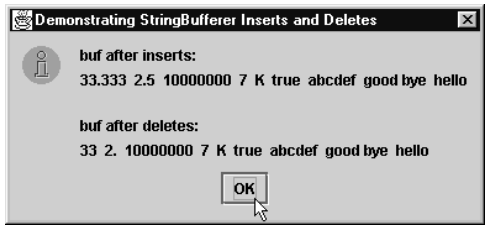

Fig. 10.16 The **`StringBuffer`** class **`insert`** methods.

```
1  // Fig. 10.17: StaticCharMethods.java
2  // Demonstrates the static character testing methods
3  // and case conversion methods of class Character
4  // from the java.lang package.
5  import javax.swing.*;
6  import java.awt.*;
7  import java.awt.event.*;
8
9  public class StaticCharMethods extends JFrame {
10    private char c;
11    private JLabel prompt;
12    private JTextField input;
13    private JTextArea outputArea;
14
15    public StaticCharMethods()
16    {
17       super( "Static Character Methods" );
18
19       Container container = getContentPane();
20       container.setLayout( new FlowLayout() );
21
22       prompt =
23          new JLabel( "Enter a character and press Enter" );
24       container.add( prompt );
25
26       input = new JTextField( 5 );
27       input.addActionListener(
28          new ActionListener() {
29             public void actionPerformed( ActionEvent e )
30             {
31                String s = e.getActionCommand();
32                c = s.charAt( 0 );
33                buildOutput();
34             }
35          }
36       );
37       container.add( input );
38
39       outputArea = new JTextArea( 10, 20 );
40       container.add( outputArea );
41
42       setSize( 300, 250 );  // set the window size
43       show();               // show the window
44    }
45
46    public void buildOutput()
47    {
48       outputArea.setText(
49           "is defined: " + Character.isDefined( c ) +
50           "\nis digit: " + Character.isDigit( c ) +
51           "\nis Java letter: " +
52           Character.isJavaIdentifierStart( c ) +
53           "\nis Java letter or digit: " +
```

```
            Character.isJavaIdentifierPart( c ) +
            "\nis letter: " + Character.isLetter( c ) +
            "\nis letter or digit: " +
            Character.isLetterOrDigit( c ) +
            "\nis lower case: " + Character.isLowerCase( c ) +
            "\nis upper case: " + Character.isUpperCase( c ) +
            "\nto upper case: " + Character.toUpperCase( c ) +
            "\nto lower case: " + Character.toLowerCase( c ) );
   }

   public static void main( String args[] )
   {
      StaticCharMethods application = new StaticCharMethods();

      application.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}
```

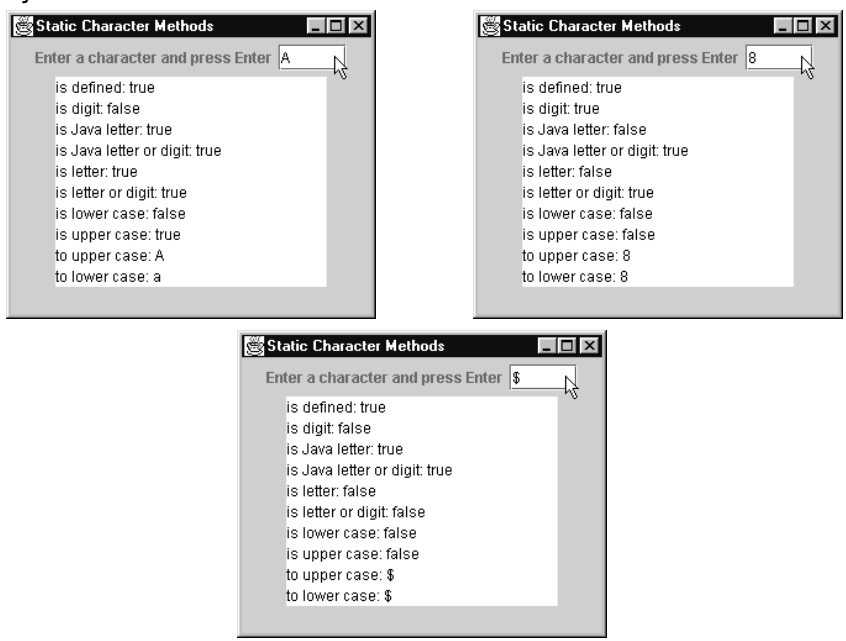

Fig. 10.17 `static` character testing methods and case conversion methods of class `Character`

```
1  // Fig. 10.18: StaticCharMethods2.java
2  // Demonstrates the static character conversion methods
3  // of class Character from the java.lang package.
4  import javax.swing.*;
5  import java.awt.*;
6  import java.awt.event.*;
7
8  public class StaticCharMethods2 extends JFrame {
9     private char c;
10    private int digit, radix;
11    private JLabel prompt1, prompt2;
12    private JTextField input, radixField;
13    private JButton toChar, toInt;
14
15    public StaticCharMethods2()
16    {
17       super( "Character Conversion Methods" );
18
19       Container con = getContentPane();
20       con.setLayout( new FlowLayout() );
21
22       prompt1 = new JLabel( "Enter a digit or character " );
23       input = new JTextField( 5 );
24       con.add( prompt1 );
25       con.add( input );
26
27       prompt2 = new JLabel( "Enter a radix " );
28       radixField = new JTextField( 5 );
29       con.add( prompt2 );
30       con.add( radixField );
31
32       toChar = new JButton( "Convert digit to character" );
33       toChar.addActionListener(
34          new ActionListener() {
35             public void actionPerformed( ActionEvent e )
36             {
37                digit = Integer.parseInt( input.getText() );
38                radix =
39                   Integer.parseInt( radixField.getText() );
40                JOptionPane.showMessageDialog( null,
41                   "Convert digit to character: " +
42                   Character.forDigit( digit, radix ) );
43             }
44          }
45       );
46       con.add( toChar );
47
48       toInt = new JButton( "Convert character to digit" );
49       toInt.addActionListener(
50          new ActionListener() {
51             public void actionPerformed( ActionEvent e )
52             {
53                String s = input.getText();
```

```
54                c = s.charAt( 0 );
55                radix =
56                   Integer.parseInt( radixField.getText() );
57                JOptionPane.showMessageDialog( null,
58                   "Convert character to digit: " +
59                   Character.digit( c, radix ) );
60             }
61          }
62       );
63       con.add( toInt );
64
65       setSize( 275, 150 );  // set the window size
66       show();               // show the window
67    }
68
69    public static void main( String args[] )
70    {
71       StaticCharMethods2 app = new StaticCharMethods2();
72
73       app.addWindowListener(
74          new WindowAdapter() {
75             public void windowClosing( WindowEvent e )
76             {
77                System.exit( 0 );
78             }
79          }
80       );
81    }
82 }
```

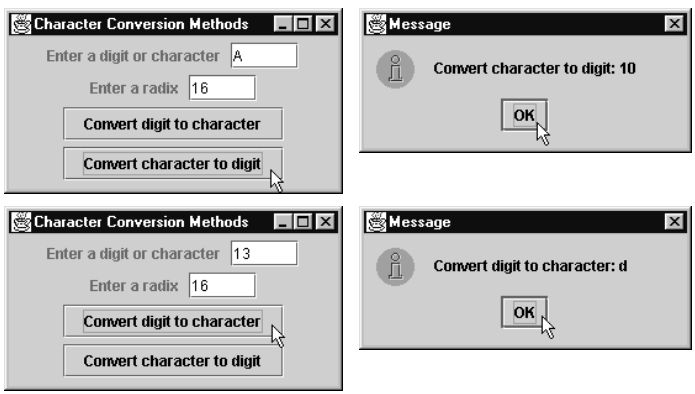

Fig. 10.18 `Character` class `static` conversion methods.

```
1  // Fig. 10.19: OtherCharMethods.java
2  // Demonstrate the non-static methods of class
3  // Character from the java.lang package.
4  import javax.swing.*;
5
6  public class OtherCharMethods {
7     public static void main( String args[] )
8     {
9        Character c1, c2;
10
11       c1 = new Character( 'A' );
12       c2 = new Character( 'a' );
13
14       String output =
15          "c1 = " + c1.charValue() +
16          "\nc2 = " + c2.toString() +
17          "\n\nhash code for c1 = " + c1.hashCode() +
18          "\nhash code for c2 = " + c2.hashCode();
19
20       if (  c1.equals( c2 ) )
21          output += "\n\nc1 and c2 are equal";
22       else
23          output += "\n\nc1 and c2 are not equal";
24
25       JOptionPane.showMessageDialog( null, output,
26          "Demonstrating Non-Static Character Methods",
27          JOptionPane.INFORMATION_MESSAGE );
28
29       System.exit( 0 );
30    }
31 }
```

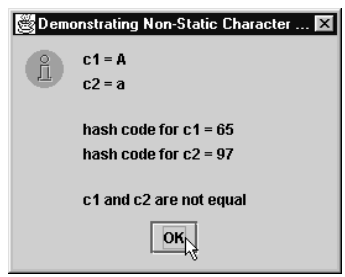

Fig. 10.19 Non-`static` methods of class `Character`.

```
1  // Fig. 10.20: TokenTest.java
2  // Testing the StringTokenizer class of the java.util package
3  import javax.swing.*;
4  import java.util.*;
5  import java.awt.*;
6  import java.awt.event.*;
7
8  public class TokenTest extends JFrame {
9     private JLabel prompt;
10    private JTextField input;
11    private JTextArea output;
12
13    public TokenTest()
14    {
15       super( "Testing Class StringTokenizer" );
16
17       Container c = getContentPane();
18       c.setLayout( new FlowLayout() );
19
20       prompt =
21          new JLabel( "Enter a sentence and press Enter" );
22       c.add( prompt );
23
24       input = new JTextField( 20 );
25       input.addActionListener(
26          new ActionListener() {
27             public void actionPerformed( ActionEvent e )
28             {
29                String stringToTokenize = e.getActionCommand();
30                StringTokenizer tokens =
31                   new StringTokenizer( stringToTokenize );
32
33                output.setText( "Number of elements: " +
34                   tokens.countTokens() +
35                   "\nThe tokens are:\n" );
36
37                while ( tokens.hasMoreTokens() )
38                   output.append( tokens.nextToken() + "\n" );
39             }
40          }
41       );
42       c.add( input );
43
44       output = new JTextArea( 10, 20 );
45       output.setEditable( false );
46       c.add( new JScrollPane( output ) );
47
48       setSize( 275, 260 );  // set the window size
49       show();               // show the window
50    }
51
52    public static void main( String args[] )
53    {
54       TokenTest app = new TokenTest();
55
```

```
56          app.addWindowListener(
57             new WindowAdapter() {
58                public void windowClosing( WindowEvent e )
59                {
60                   System.exit( 0 );
61                }
62             }
63          );
64       }
65    }
```

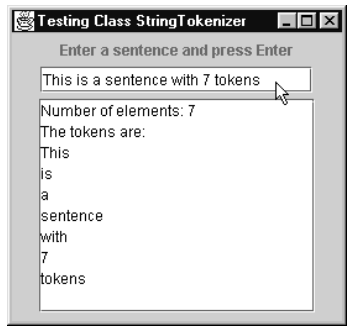

Fig. 10.20 Tokenizing strings with a **`StringTokenizer`** object.

```
1  // Fig. 10.21: DeckOfCards.java
2  // Card shuffling and dealing program
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class DeckOfCards extends JFrame {
8     private Card deck[];
9     private int currentCard;
10    private JButton dealButton, shuffleButton;
11    private JTextField displayCard;
12    private JLabel status;
13
14    public DeckOfCards()
15    {
16       super( "Card Dealing Program" );
17
18       String faces[] = { "Ace", "Deuce", "Three", "Four",
19                          "Five", "Six", "Seven", "Eight",
20                          "Nine", "Ten", "Jack", "Queen",
21                          "King" };
22       String suits[] = { "Hearts", "Diamonds",
23                          "Clubs", "Spades" };
24
25       deck = new Card[ 52 ];
26       currentCard = -1;
27
28       for ( int i = 0; i < deck.length; i++ )
29          deck[ i ] = new Card( faces[ i % 13 ],
30                               suits[ i / 13 ] );
31
32       Container c = getContentPane();
33       c.setLayout( new FlowLayout() );
34
35       dealButton = new JButton( "Deal card" );
36       dealButton.addActionListener(
37          new ActionListener() {
38             public void actionPerformed( ActionEvent e )
39             {
40                Card dealt = dealCard();
41
42                if ( dealt != null ) {
43                   displayCard.setText( dealt.toString() );
44                   status.setText( "Card #: " + currentCard );
45                }
46                else {
47                   displayCard.setText(
48                      "NO MORE CARDS TO DEAL" );
49                   status.setText(
50                      "Shuffle cards to continue" );
51                }
52             }
53          }
54       );
55       c.add( dealButton );
```

```
56
57          shuffleButton = new JButton( "Shuffle cards" );
58          shuffleButton.addActionListener(
59             new ActionListener() {
60                public void actionPerformed( ActionEvent e )
61                {
62                   displayCard.setText( "SHUFFLING ..." );
63                   shuffle();
64                   displayCard.setText( "DECK IS SHUFFLED" );
65                }
66             }
67          );
68          c.add( shuffleButton );
69
70          displayCard = new JTextField( 20 );
71          displayCard.setEditable( false );
72          c.add( displayCard );
73
74          status = new JLabel();
75          c.add( status );
76
77          setSize( 275, 120 );  // set the window size
78          show();               // show the window
79       }
80
81       public void shuffle()
82       {
83          currentCard = -1;
84
85          for ( int i = 0; i < deck.length; i++ ) {
86             int j =  ( int ) ( Math.random() * 52 );
87             Card temp = deck[ i ];   // swap
88             deck[ i ] = deck[ j ];   // the
89             deck[ j ] = temp;        // cards
90          }
91
92          dealButton.setEnabled( true );
93       }
94
95       public Card dealCard()
96       {
97          if ( ++currentCard < deck.length )
98             return deck[ currentCard ];
99          else {
100            dealButton.setEnabled( false );
101            return null;
102         }
103      }
104
105      public static void main( String args[] )
106      {
107         DeckOfCards app = new DeckOfCards();
108
109         app.addWindowListener(
110            new WindowAdapter() {
```

```
111             public void windowClosing( WindowEvent e )
112             {
113                System.exit( 0 );
114             }
115          }
116       );
117    }
118 }
119
120 class Card {
121    private String face;
122    private String suit;
123
124    public Card( String f, String s )
125    {
126       face = f;
127       suit = s;
128    }
129
130    public String toString() { return face + " of " + suit; }
131 }
```

Fig. 10.21 Card dealing program.