Chapter 10. Strings and Characters

- String and character processing
  - Class **`java.lang.String`**
  - Class **`java.lang.StringBuffer`**
  - Class **`java.lang.Character`**
  - Class **`java.util.StringTokenizer`**

10.2 Fundamentals of Characters and Strings

- Characters
  - "Building blocks" of Java source programs
- String
  - Series of characters treated as single unit
  - May include letters, digits, etc.
  - Object of class **`String`**

Chapter 10. Strings and Characters

- Class **`String`**
  - Provides 9 constructors
- String is an immutable object ➔不可變更的物件
  - + & += 連接字串的功能是透過StringBuffer物件來執行
- String Method
  - Indexof(string, fromindex) ➔ 後有提及
  - Substring(beginindex, endindex)
    - Extract substrings➔ return sub-string of original string
  - String1.Equals(string2) ➔ 後有提及
  - toUpperCase(), toLowerCase()
  - Replace(char1, char2)

```
1   // Fig. 10.1: StringConstructors.java
2   // This program demonstrates the String class constructors.
3
4   // Java extension packages
5   import javax.swing.*;
6
7   public class StringConstructors {
8
9      // test String constructors
10     public static void main( String args[] )
11     {
12        char charArray[] = { 'b', 'i', 'r', 't', 'h', ' ',
13                             'd', 'a', 'y' };
14        byte byteArray[] = { ( byte ) 'n', ( byte ) 'e',
15           ( byte ) 'w', ( byte ) ' ', ( byte ) 'y',
16           ( byte ) 'e', ( byte ) 'a', ( byte ) 'r' };
17
18        StringBuffer buffer;
19        String s, s1, s2, s3, s4, s5, s6, s7, output;
20
21        s = new String( "hello" );
22        buffer = new StringBuffer( "Welcome to Java Programming!" );
23
24        // use String constructors
25        s1 = new String();
26        s2 = new String( s );
27        s3 = new String( charArray );
28        s4 = new String( charArray, 6, 3 );
29        s5 = new String( byteArray, 4, 4 );
30        s6 = new String( byteArray );
31        s7 = new String( buffer );
32
```

Outline

StringConstructors.java

Line 25

Line 26

Line 27

Line 28

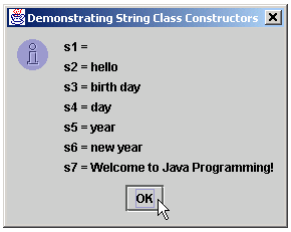
Line 29

Line 30

Line 31

3

```
33        // append Strings to output
34        output = "s1 = " + s1 + "\ns2 = " + s2 + "\ns3 = " + s3 +
35           "\ns4 = " + s4 + "\ns5 = " + s5 + "\ns6 = " + s6 +
36           "\ns7 = " + s7;
37
38        JOptionPane.showMessageDialog( null, output,
39           "Demonstrating String Class Constructors",
40           JOptionPane.INFORMATION_MESSAGE );
41
42        System.exit( 0 );
43     }
44
45  }  // end class StringConstructors
```



Outline

StringConstructors.java

4

10.4 `String` Methods `length`, `charAt` and `getChars`

- Method **`length()`** → **`XX.length()`**
  - Determine **`String`** length
    - Like arrays, **`String`**s always "know" their size
    - Unlike array, **`String`**s do not have length instance variable
    - Return an integer
- Method **`charAt()`** → **`XX.charAt()`**
  - Get character at specific location in **`String`**
  - Return a character
- Method **`getChars()`**→ **`XX.getChars(s,c,d,s)`**
  - Get entire set of characters in **`String`**
  - No return → parameter should have a destination char array

5

```
1  // Fig. 10.2: StringMiscellaneous.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
10 // Java extension packages
11 import javax.swing.*;
12
13 public class StringMiscellaneous {
14
15    // test miscellaneous String methods
16    public static void main( String args[] )
17    {
18       String s1, output;
19       char charArray[];
20
21       s1 = new String( "hello there" );
22       charArray = new char[ 5 ];
23
24       // output the string
25       output = "s1: " + s1;
26
27       // test length method
28       output += "\nLength of s1: " + s1.length();
29
30       // loop through characters in s1 and display reversed
31       output += "\nThe string reversed is: ";
32
33       for ( int count = s1.length() - 1; count >= 0; count-- )
34          output += s1.charAt( count ) + " ";
35
```

Outline

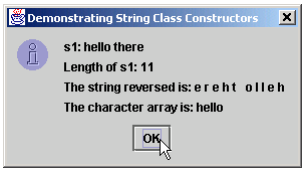
StringMiscellaneous.java

Line 28

Line 33

6

```
36         // copy characters from string into char array
37         s1.getChars( 0, 5, charArray, 0 );
38         output += "\nThe character array is: ";
39
40         for ( int count = 0; count < charArray.length; count++ )
41            output += charArray[ count ];
42
43         JOptionPane.showMessageDialog( null, output,
44            "Demonstrating String Class Constructors",
45            JOptionPane.INFORMATION_MESSAGE );
46
47         System.exit( 0 );
48      }
49
50   }  // end class StringMiscellaneous
```



10.5 Comparing Strings

- Comparing **String** objects
  - Method **equals()**
  - Method **equalsIgnoreCase()**
  - Method **compareTo(0, 1, -1, or first ASCII 之差)**
  - Method **regionMatches(case sensitive or not)**
  - Method **startsWith(return boolean→ check start substring)**
  - Method **endsWith(return boolean→ check end substring)**

```
1    // Fig. 10.3: StringCompare.java
2    // This program demonstrates the methods equals,
3    // equalsIgnoreCase, compareTo, and regionMatches
4    // of the String class.
5
6    // Java extension packages
7    import javax.swing.JOptionPane;
8
9    public class StringCompare {
10
11      // test String class comparison methods
12      public static void main( String args[] )
13      {
14         String s1, s2, s3, s4, output;
15
16         s1 = new String( "hello" );
17         s2 = new String( "good bye" );
18         s3 = new String( "Happy Birthday" );
19         s4 = new String( "happy birthday" );
20
21         output = "s1 = " + s1 + "\ns2 = " + s2 +
22            "\ns3 = " + s3 + "\ns4 = " + s4 + "\n\n";
23
24         // test for equality
25         if ( s1.equals( "hello" ) )
26            output += "s1 equals \"hello\"\n";
27         else
28            output += "s1 does not equal \"hello\"\n";
29
30         // test for equality with ==
31         if ( s1 == "hello" )
32            output += "s1 equals \"hello\"\n";
33         else
34            output += "s1 does not equal \"hello\"\n";
35
```

Outline

StringCompare.java

Line 25

Line 31

9

```
36         // test for equality (ignore case)
37         if ( s3.equalsIgnoreCase( s4 ) )
38            output += "s3 equals s4\n";
39         else
40            output += "s3 does not equal s4\n";
41
42         // test compareTo
43         output +=
44            "\ns1.compareTo( s2 ) is " + s1.compareTo( s2 ) +
45            "\ns2.compareTo( s1 ) is " + s2.compareTo( s1 ) +
46            "\ns1.compareTo( s1 ) is " + s1.compareTo( s1 ) +
47            "\ns3.compareTo( s4 ) is " + s3.compareTo( s4 ) +
48            "\ns4.compareTo( s3 ) is " + s4.compareTo( s3 ) +
49            "\n\n";
50
51         // test regionMatches (case sensitive)
52         if ( s3.regionMatches( 0, s4, 0, 5 ) )
53            output += "First 5 characters of s3 and s4 match\n";
54         else
55            output +=
56               "First 5 characters of s3 and s4 do not match\n";
57
58         // test regionMatches (ignore case)
59         if ( s3.regionMatches( true, 0, s4, 0, 5 ) )
60            output += "First 5 characters of s3 and s4 match";
61         else
62            output +=
63               "First 5 characters of s3 and s4 do not match";
64
```

Outline

StringCompare.java

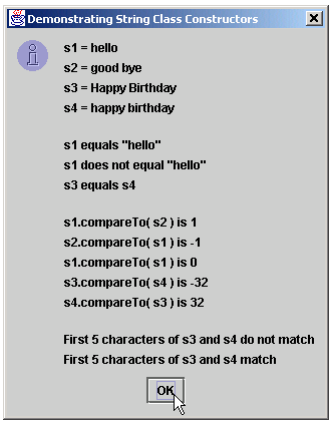
Line 37

Lines 44-48

Line 52 and 59

10

```
65          JOptionPane.showMessageDialog( null, output,
66             "Demonstrating String Class Constructors",
67             JOptionPane.INFORMATION_MESSAGE );
68
69          System.exit( 0 );
70       }
71
72    }  // end class StringCompare
```



Outline

StringCompare.java

11

```
1    // Fig. 10.4: StringStartEnd.java
2    // This program demonstrates the methods startsWith and
3    // endsWith of the String class.
4
5    // Java extension packages
6    import javax.swing.*;
7
8    public class StringStartEnd {
9
10      // test String comparison methods for beginning and end
11      // of a String
12      public static void main( String args[] )
13      {
14         String strings[] =
15            { "started", "starting", "ended", "ending" };
16         String output = "";
17
18         // test method startsWith
19         for ( int count = 0; count < strings.length; count++ )
20
21            if ( strings[ count ].startsWith( "st" ) )
22               output += "\"" + strings[ count ] +
23                  "\" starts with \"st\"\n";
24
25         output += "\n";
26
27         // test method startsWith starting from position
28         // 2 of the string
29         for ( int count = 0; count < strings.length; count++ )
30
31            if ( strings[ count ].startsWith( "art", 2 ) )
32               output += "\"" + strings[ count ] +
33                  "\" starts with \"art\" at position 2\n";
34
35         output += "\n";
```

Outline

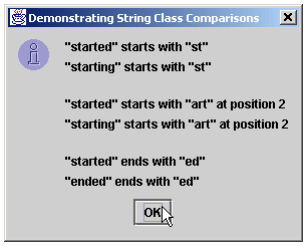
StringStartEnd.java

Line 21

Line 31

12

```
36
37          // test method endsWith
38          for ( int count = 0; count < strings.length; count++ )
39
40             if ( strings[ count ].endsWith( "ed" ) )
41                output += "\"" + strings[ count ] +
42                   "\" ends with \"ed\"\n";
43
44          JOptionPane.showMessageDialog( null, output,
45             "Demonstrating String Class Comparisons",
46             JOptionPane.INFORMATION_MESSAGE );
47
48          System.exit( 0 );
49       }
50
51    }  // end class StringStartEnd
```



Outline

StringStartEnd.java

Line 40

13

10.6 `String` Method `hashCode`

- Hash table
  - Stores information using calculation on storable object
    - Produces *hash code*
      - Used to choose location in table at which to store object
  - Fast lookup just use calculation instead of searching(linear or binary search)

14

```
1   // Fig. 10.5: StringHashCode.java
2   // This program demonstrates the method
3   // hashCode of the String class.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringHashCode {
9
10     // test String hashCode method
11     public static void main( String args[] )
12     {
13        String s1 = "hello", s2 = "Hello";
14
15        String output =
16           "The hash code for \"" + s1 + "\" is " +
17           s1.hashCode() +
18           "\nThe hash code for \"" + s2 + "\" is " +
19           s2.hashCode();
20
21        JOptionPane.showMessageDialog( null, output,
22           "Demonstrating String Method hashCode",
23           JOptionPane.INFORMATION_MESSAGE );
24
25        System.exit( 0 );
26     }
27
28  }  // end class StringHashCode
```

Outline

StringHashCode.java

Line 17 and 19

15

10.7 Locating Characters and Substrings in `Strings`

- Search for characters in **String**
  - Method **indexOf→ return the index of the first occurrence of the character in the String, return –1 if not found**
  - Method **lastIndexOf→return the index of the last occurrence of the character in the String , return –1 if not found**

16

```
1   // Fig. 10.6: StringIndexMethods.java
2   // This program demonstrates the String
3   // class index methods.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringIndexMethods {
9
10     // String searching methods
11     public static void main( String args[] )
12     {
13        String letters = "abcdefghijklmabcdefghijklm";
14
15        // test indexOf to locate a character in a string
16        String output = "'c' is located at index " +
17           letters.indexOf( 'c' );
18
19        output += "\n'a' is located at index " +
20           letters.indexOf( 'a', 1 );
21
22        output += "\n'$' is located at index " +
23           letters.indexOf( '$' );
24
25        // test lastIndexOf to find a character in a string
26        output += "\n\nLast 'c' is located at index " +
27           letters.lastIndexOf( 'c' );
28
29        output += "\nLast 'a' is located at index " +
30           letters.lastIndexOf( 'a', 25 );
31
32        output += "\nLast '$' is located at index " +
33           letters.lastIndexOf( '$' );
34
```

Outline

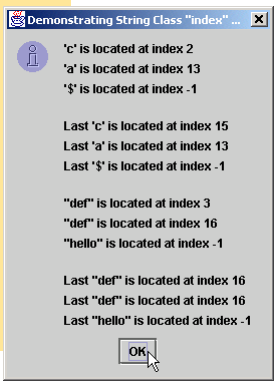
StringIndexMethods.java

Lines 16-23

Lines 26-33

17

```
35        // test indexOf to locate a substring in a string
36        output += "\n\n\"def\" is located at index " +
37           letters.indexOf( "def" );
38
39        output += "\n\"def\" is located at index " +
40           letters.indexOf( "def", 7 );
41
42        output += "\n\"hello\" is located at index " +
43           letters.indexOf( "hello" );
44
45        // test lastIndexOf to find a substring in a string
46        output += "\n\nLast \"def\" is located at index " +
47           letters.lastIndexOf( "def" );
48
49        output += "\nLast \"def\" is located at index " +
50           letters.lastIndexOf( "def", 25 );
51
52        output += "\nLast \"hello\" is located at index " +
53           letters.lastIndexOf( "hello" );
54
55        JOptionPane.showMessageDialog( null, output,
56           "Demonstrating String Class \"index\" Methods",
57           JOptionPane.INFORMATION_MESSAGE );
58
59        System.exit( 0 );
60     }
61
62  }  // end class StringIndexMethods
```

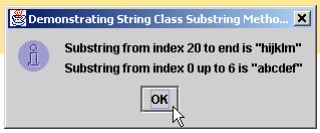


Outline

StringIndexMethods.java

18

```
1   // Fig. 10.7: SubString.java
2   // This program demonstrates the
3   // String class substring methods.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class SubString {
9
10     // test String substring methods
11     public static void main( String args[] )
12     {
13        String letters = "abcdefghijklmabcdefghijklm";
14
15        // test substring methods
16        String output = "Substring from index 20 to end is " +
17           "\"" + letters.substring( 20 ) + "\"\n";
18
19        output += "Substring from index 0 up to 6 is " +
20           "\"" + letters.substring( 0, 6 ) + "\"";
21
22        JOptionPane.showMessageDialog( null, output,
23           "Demonstrating String Class Substring Methods",
24           JOptionPane.INFORMATION_MESSAGE );
25
26        System.exit( 0 );
27     }
28
29  }  // end class SubString
```



Outline

SubString.java

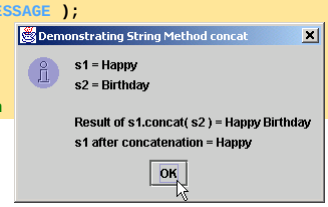
Line 17

Line 20

10.9 Concatenating `Strings`

- Method **`concat`**
  - Concatenate two **`String`** objects

```
1   // Fig. 10.8: StringConcatenation.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
6   // Java extension packages
7   import javax.swing.*;
8
9   public class StringConcatenation {
10
11     // test String method concat
12     public static void main( String args[] )
13     {
14        String s1 = new String( "Happy " ),
15               s2 = new String( "Birthday" );
16
17        String output = "s1 = " + s1 + "\ns2 = " + s2;
18
19        output += "\n\nResult of s1.concat( s2 ) = " +
20           s1.concat( s2 );
21
22        output += "\ns1 after concatenation = " + s1;
23
24        JOptionPane.showMessageDialog( null, output,
25           "Demonstrating String Method concat",
26           JOptionPane.INFORMATION_MESSAGE );
27
28        System.exit( 0 );
29     }
30
31  }  // end class StringConcatenation
```



Outline

StringConcatenation.java

Line 20

Line 22

21

10.10 Miscellaneous `String` Methods

- Miscellaneous **`String`** methods
  - Return modified copies of **`String replace(new,old)`**
  - Return character array

22

```
1   // Fig. 10.9: StringMiscellaneous2.java
2   // This program demonstrates the String methods replace,
3   // toLowerCase, toUpperCase, trim, toString and toCharArray
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringMiscellaneous2 {
9
10     // test miscellaneous String methods
11     public static void main( String args[] )
12     {
13        String s1 = new String( "hello" ),
14           s2 = new String( "GOOD BYE" ),
15           s3 = new String( "   spaces   " );
16
17        String output = "s1 = " + s1 + "\ns2 = " + s2 +
18           "\ns3 = " + s3;
19
20        // test method replace
21        output += "\n\nReplace 'l' with 'L' in s1: " +
22           s1.replace( 'l', 'L' );
23
24        // test toLowerCase and toUpperCase
25        output +=
26           "\n\ns1.toUpperCase() = " + s1.toUpperCase() +
27           "\ns2.toLowerCase() = " + s2.toLowerCase();
28
29        // test trim method
30        output += "\n\ns3 after trim = \"" + s3.trim() + "\"";
31
32        // test toString method
33        output += "\n\ns1 = " + s1.toString();
34
```

Outline

StringMiscellaneous2.java

Line 22

Line 26

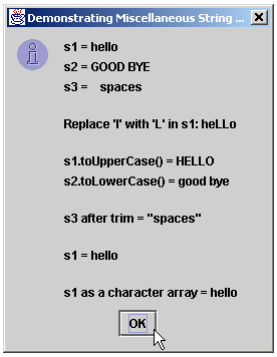
Line 27

Line 30

Line 33

23

```
35        // test toCharArray method
36        char charArray[] = s1.toCharArray();
37
38        output += "\n\ns1 as a character array = ";
39
40        for ( int count = 0; count < charArray.length; ++count )
41           output += charArray[ count ];
42
43        JOptionPane.showMessageDialog( null, output,
44           "Demonstrating Miscellaneous String Methods",
45           JOptionPane.INFORMATION_MESSAGE );
46
47        System.exit( 0 );
48     }
49
50  }  // end class StringMiscellaneous2
```



Outline

StringMiscellaneous2.java

Line 36

24

10.11 Using `String` Method `valueOf`

- **`String`** provides **`static`** class methods
  - Method **`valueOf`**
    - Convert the arguments to Strings and Returns the **`String`** object
- The reverse class method
  - Integer.parseInt(“123”)
  - Double.parseDouble(“12.3”);
  - Float.parseFloat(“34.6f”);
  - Long.parseLong(“32164752”)

25

Outline

StringValueOf.java

Lines 26-32

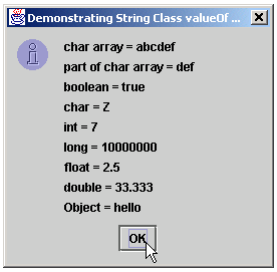
```
1   // Fig. 10.10: StringValueOf.java
2   // This program demonstrates the String class valueOf methods.
3
4   // Java extension packages
5   import javax.swing.*;
6
7   public class StringValueOf {
8
9      // test String valueOf methods
10     public static void main( String args[] )
11     {
12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

26

```
34         JOptionPane.showMessageDialog( null, output,
35            "Demonstrating String Class valueOf Methods",
36            JOptionPane.INFORMATION_MESSAGE );
37
38         System.exit( 0 );
39      }
40
41   }  // end class StringValueOf
```

Outline

StringValueOf.java



27

10.12 `String` Method `intern`

- String comparisons
  - Slow operation
  - Method **`intern`** improves this performance
    - Returns reference to **`String`**

      `If a program uses intern on extremely large Strings, the program can compare those Strings faster by using the == operator➔a fast operation`

    - Guarantees reference has same contents as original **`String`**

28

```
1   // Fig. 10.11: StringIntern.java
2   // This program demonstrates the intern method
3   // of the String class.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringIntern {
9
10     // test String method intern
11     public static void main( String args[] )
12     {
13        String s1, s2, s3, s4, output;
14
15        s1 = new String( "hello" );
16        s2 = new String( "hello" );
17
18        // test strings to determine if they are same
19        // String object in memory
20        if ( s1 == s2 )
21           output = "s1 and s2 are the same object in memory";
22        else
23           output = "s1 and s2 are not the same object in memory";
24
25        // test strings for equality of contents
26        if ( s1.equals( s2 ) )
27           output += "\ns1 and s2 are equal";
28        else
29           output += "\ns1 and s2 are not equal";
30
31        // use String intern method to get a unique copy of
32        // "hello" referred to by both s3 and s4
33        s3 = s1.intern();
34        s4 = s2.intern();
35
```

Outline

StringIntern.java

Lines 15-20

Line 26

Lines 33-34

29

```
36        // test strings to determine if they are same
37        // String object in memory
38        if ( s3 == s4 )
39           output += "\ns3 and s4 are the same object in memory";
40        else
41           output +=
42              "\ns3 and s4 are not the same object in memory";
43
44        // determine if s1 and s3 refer to same object
45        if ( s1 == s3 )
46           output +=
47              "\ns1 and s3 are the same object in memory";
48        else
49           output +=
50              "\ns1 and s3 are not the same object in memory";
51
52        // determine if s2 and s4 refer to same object
53        if ( s2 == s4 )
54           output += "\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 same object
60        if ( s1 == s4 )
61           output += "\ns1 and s4 are the same object in memory";
62        else
63           output +=
64              "\ns1 and s4 are not the same object in memory";
65
```

Outline

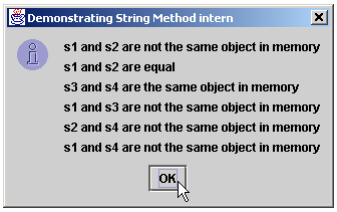
StringIntern.java

30

```
66          JOptionPane.showMessageDialog( null, output,
67             "Demonstrating String Method intern",
68             JOptionPane.INFORMATION_MESSAGE );
69
70          System.exit( 0 );
71       }
72
73    }  // end class StringIntern
```

Outline

StringIntern.java



31

10.13 `StringBuffer` Class

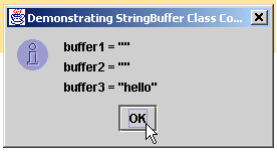
- Class **`StringBuffer`**
  - When **`String`** object is created, its contents cannot change
  - Used for creating and manipulating dynamic string data
    - i.e., modifiable **`String`**s
  - Can store characters based on capacity
    - Capacity expands dynamically to handle additional characters
  - Uses operators **`+`** and **`+=`** for **`String`** concatenation

10.14 `StringBuffer` Constructors

- Three **`StringBuffer`** constructors
  - Default creates **`StringBuffer`** with no characters
    - Capacity of 16 characters

32

```
1   // Fig. 10.12: StringBufferConstructors.java
2   // This program demonstrates the StringBuffer constructors.
3
4   // Java extension packages
5   import javax.swing.*;
6
7   public class StringBufferConstructors {
8
9      // test StringBuffer constructors
10     public static void main( String args[] )
11     {
12        StringBuffer buffer1, buffer2, buffer3;
13
14        buffer1 = new StringBuffer();
15        buffer2 = new StringBuffer( 10 );
16        buffer3 = new StringBuffer( "hello" );
17
18        String output =
19           "buffer1 = \"" + buffer1.toString() + "\"" +
20           "\nbuffer2 = \"" + buffer2.toString() + "\"" +
21           "\nbuffer3 = \"" + buffer3.toString() + "\"";
22
23        JOptionPane.showMessageDialog( null, output,
24           "Demonstrating StringBuffer Class Constructors",
25           JOptionPane.INFORMATION_MESSAGE );
26
27        System.exit( 0 );
28     }
29
30  }  // end class StringBufferConstructors
```



Outline

StringBufferConstructors.java

Line 14

Line 15

Line 16

Lines 19-21

33

10.15 StringBuffer Methods length, capacity, setLength and ensureCapacity

- Method **length**
  - Return the number of character currently in a **StringBuffer**
- Method **capacity**
  - Return the number of character can be stored in **StringBuffer**
- Method **setLength**
  - Increase or decrease **StringBuffer** length
- Method **ensureCapacity**
  - Set **StringBuffer** capacity
  - Guarantee that **StringBuffer** has minimum capacity

34

```
1   // Fig. 10.13: StringBufferCapLen.java
2   // This program demonstrates the length and
3   // capacity methods of the StringBuffer class.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferCapLen {
9
10     // test StringBuffer methods for capacity and length
11     public static void main( String args[] )
12     {
13        StringBuffer buffer =
14           new StringBuffer( "Hello, how are you?" );
15
16        String output = "buffer = " + buffer.toString() +
17           "\nlength = " + buffer.length() +
18           "\ncapacity = " + buffer.capacity();
19
20        buffer.ensureCapacity( 75 );
21        output += "\n\nNew capacity = " + buffer.capacity();
22
23        buffer.setLength( 10 );
24        output += "\n\nNew length = " + buffer.length() +
25           "\nbuf = " + buffer.toString();
26
27        JOptionPane.showMessageDialog( null, output,
28           "StringBuffer length and capacity Methods",
29           JOptionPane.INFORMATION_MESSAGE );
30
31        System.exit( 0 );
32     }
33
34  }  // end class StringBufferCapLen
```

Outline

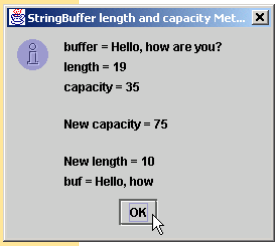
StringBufferCapLen.java

Line 17

Line 18

Line 20

Line 23



35

10.16 StringBuffer Methods charAt, setCharAt, getChars and reverse

- Manipulating **StringBuffer** characters
  - Method **charAt**
    - Return **StringBuffer** character at specified index
  - Method **setCharAt**
    - Set **StringBuffer** character at specified index
  - Method **getChars**
    - Return character array from **StringBuffer**
  - Method **reverse**
    - Reverse **StringBuffer** contents

10.17 StringBuffer append Methods

- Method **append**
  - Allow data-type values to be added to **StringBuffer**

36

```
1   // Fig. 10.14: StringBufferChars.java
2   // The charAt, setCharAt, getChars, and reverse methods
3   // of class StringBuffer.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferChars {
9
10     // test StringBuffer character methods
11     public static void main( String args[] )
12     {
13        StringBuffer buffer = new StringBuffer( "hello there" );
14
15        String output = "buffer = " + buffer.toString() +
16           "\nCharacter at 0: " + buffer.charAt( 0 ) +
17           "\nCharacter at 4: " + buffer.charAt( 4 );
18
19        char charArray[] = new char[ buffer.length() ];
20        buffer.getChars( 0, buffer.length(), charArray, 0 );
21        output += "\n\nThe characters are: ";
22
23        for ( int count = 0; count < charArray.length; ++count )
24           output += charArray[ count ];
25
26        buffer.setCharAt( 0, 'H' );
27        buffer.setCharAt( 6, 'T' );
28        output += "\n\nbuf = " + buffer.toString();
29
30        buffer.reverse();
31        output += "\n\nbuf = " + buffer.toString();
32
```

Outline

StringBufferChars.java

Lines 16-17

Line 20

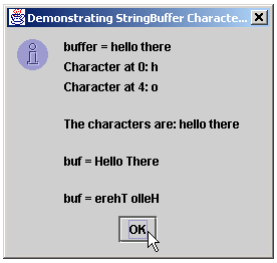
Lines 26-27

Line 30


```
33        JOptionPane.showMessageDialog( null, output,
34           "Demonstrating StringBuffer Character Methods",
35           JOptionPane.INFORMATION_MESSAGE );
36
37        System.exit( 0 );
38     }
39
40  }  // end class StringBufferChars
```

Outline

StringBufferChars.java



```
1   // Fig. 10.15: StringBufferAppend.java
2   // This program demonstrates the append
3   // methods of the StringBuffer class.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferAppend {
9
10     // test StringBuffer append methods
11     public static void main( String args[] )
12     {
13        Object o = "hello";
14        String s = "good bye";
15        char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
16        boolean b = true;
17        char c = 'Z';
18        int i = 7;
19        long l = 10000000;
20        float f = 2.5f;
21        double d = 33.333;
22        StringBuffer buffer = new StringBuffer();
23
24        buffer.append( o );
25        buffer.append( "  " );
26
```

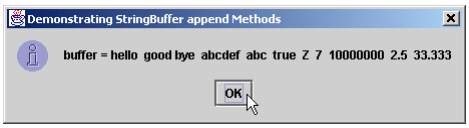
Outline

StringBufferAppend.java

Line 24

39

```
27        buffer.append( s );
28        buffer.append( "  " );
29        buffer.append( charArray );
30        buffer.append( "  " );
31        buffer.append( charArray, 0, 3 );
32        buffer.append( "  " );
33        buffer.append( b );
34        buffer.append( "  " );
35        buffer.append( c );
36        buffer.append( "  " );
37        buffer.append( i );
38        buffer.append( "  " );
39        buffer.append( l );
40        buffer.append( "  " );
41        buffer.append( f );
42        buffer.append( "  " );
43        buffer.append( d );
44
45        JOptionPane.showMessageDialog( null,
46           "buffer = " + buffer.toString(),
47           "Demonstrating StringBuffer append Methods",
48           JOptionPane.INFORMATION_MESSAGE );
49
50        System.exit( 0 );
51     }
52
53  }  // end StringBufferAppend
```



Outline

StringBufferAppend.java

Line 27

Line 29

Line 31

Lines 33-43

40

10.18 `StringBuffer` Insertion and Deletion Methods

- Method **`insert`**
  - Allow data-type values to be inserted into **`StringBuffer`**
- Methods **`delete`** and **`deleteCharAt`**
  - Allow characters to be removed from **`StringBuffer`**

Outline

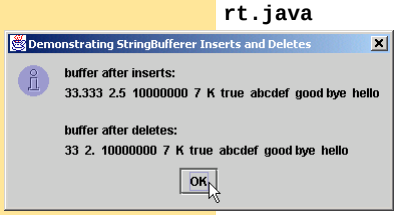
`StringBufferInsert.java`

```
1   // Fig. 10.16: StringBufferInsert.java
2   // This program demonstrates the insert and delete
3   // methods of class StringBuffer.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class StringBufferInsert {
9
10     // test StringBuffer insert methods
11     public static void main( String args[] )
12     {
13        Object o = "hello";
14        String s = "good bye";
15        char charArray[] = { 'a', 'b', 'c', 'd', 'e', 'f' };
16        boolean b = true;
17        char c = 'K';
18        int i = 7;
19        long l = 10000000;
20        float f = 2.5f;
21        double d = 33.333;
22        StringBuffer buffer = new StringBuffer();
23
```

```
24          buffer.insert( 0, o );
25          buffer.insert( 0, "  " );
26          buffer.insert( 0, s );
27          buffer.insert( 0, "  " );
28          buffer.insert( 0, charArray );
29          buffer.insert( 0, "  " );
30          buffer.insert( 0, b );
31          buffer.insert( 0, "  " );
32          buffer.insert( 0, c );
33          buffer.insert( 0, "  " );
34          buffer.insert( 0, i );
35          buffer.insert( 0, "  " );
36          buffer.insert( 0, l );
37          buffer.insert( 0, "  " );
38          buffer.insert( 0, f );
39          buffer.insert( 0, "  " );
40          buffer.insert( 0, d );
41
42          String output =
43             "buffer after inserts:\n" + buffer.toString();
44
45          buffer.deleteCharAt( 10 );  // delete 5 in 2.5
46          buffer.delete( 2, 6 );      // delete .333 in 33.333
47
48          output +=
49             "\n\nbuffer after deletes:\n" + buffer.toString();
50
51          JOptionPane.showMessageDialog( null, output,
52             "Demonstrating StringBufferer Inserts and Deletes",
53             JOptionPane.INFORMATION_MESSAGE );
54
55          System.exit( 0 );
56       }
57
58    }  // end class StringBufferInsert
```

Outline

StringBufferInsert.java



10.19 `Character` Class Examples

- Treat primitive variables as objects
  - Type wrapper classes derived from **Number (not include Character and Boolean)**
    - **`Boolean`**
    - **`Character`**
    - **`Double`**
    - **`Float`**
    - **`Byte`**
    - **`Short`**
    - **`Integer`**
    - **`Long`**
  - We examine class **`Character`**
    - **forDigit➔convert integer into a character**
    - **Digit → convert a character into a integer number system**

Outline

StaticCharMethods.java

```
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
6   // Java core packages
7   import java.awt.*;
8   import java.awt.event.*;
9
10  // Java extension packages
11  import javax.swing.*;
12
13  public class StaticCharMethods extends JFrame {
14     private char c;
15     private JLabel promptLabel;
16     private JTextField inputField;
17     private JTextArea outputArea;
18
19     // set up GUI
20     public StaticCharMethods()
21     {
22        super( "Static Character Methods" );
23
24        Container container = getContentPane();
25        container.setLayout( new FlowLayout() );
26
27        promptLabel =
28           new JLabel( "Enter a character and press Enter" );
29        container.add( promptLabel );
30
31        inputField = new JTextField( 5 );
32
33        inputField.addActionListener(
34
```

45

Outline

StaticCharMethods.java

```
35           // anonymous inner class
36           new ActionListener() {
37
38              // handle text field event
39              public void actionPerformed( ActionEvent event )
40              {
41                 String s = event.getActionCommand();
42                 c = s.charAt( 0 );
43                 buildOutput();
44              }
45
46           }  // end anonymous inner class
47
48        ); // end call to addActionListener
49
50        container.add( inputField );
51
52        outputArea = new JTextArea( 10, 20 );
53        container.add( outputArea );
54
55        setSize( 300, 250 );  // set the window size
56        show();               // show the window
57     }
58
59     // display character info in outputArea
60     public void buildOutput()
61     {
```

46

```
62          outputArea.setText(
63             "is defined: " + Character.isDefined( c ) +
64             "\nis digit: " + Character.isDigit( c ) +
65             "\nis Java letter: " +
66             Character.isJavaIdentifierStart( c ) +
67             "\nis Java letter or digit: " +
68             Character.isJavaIdentifierPart( c ) +
69             "\nis letter: " + Character.isLetter( c ) +
70             "\nis letter or digit: " +
71             Character.isLetterOrDigit( c ) +
72             "\nis lower case: " + Character.isLowerCase( c ) +
73             "\nis upper case: " + Character.isUpperCase( c ) +
74             "\nto upper case: " + Character.toUpperCase( c ) +
75             "\nto lower case: " + Character.toLowerCase( c ) );
76       }
77
78       // execute application
79       public static void main( String args[] )
80       {
81          StaticCharMethods application = new StaticCharMethods();
82
83          application.addWindowListener(
84
85             // anonymous inner class
86             new WindowAdapter() {
87
88                // handle event when user closes window
89                public void windowClosing( WindowEvent windowEvent )
90                {
91                   System.exit( 0 );
92                }
```

Outline

StaticCharMethods.java

Line 63

Line 66

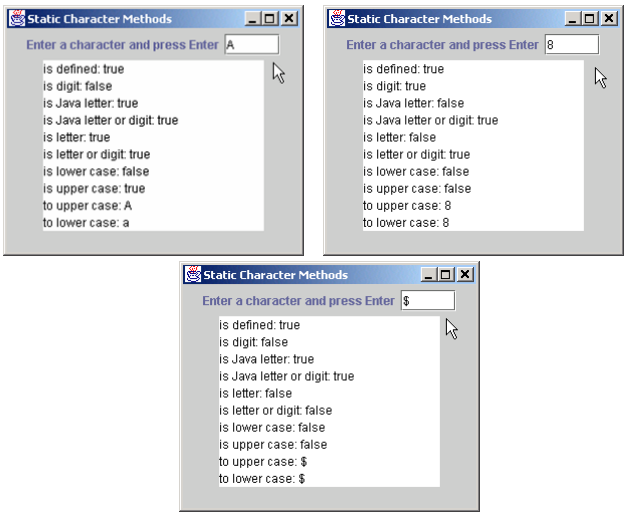
Line 68

Line 69

Line 71

Lines 72-73


```
93
94             }  // end anonymous inner class
95
96          ); // end call to addWindowListener
97
98       }  // end method main
99
100 }  // end class StaticCharMethods
```



Outline

StaticCharMethods.java

```
1   // Fig. 10.18: StaticCharMethods2.java
2   // Demonstrates the static character conversion methods
3   // of class Character from the java.lang package.
4
5   // Java core packages
6   import java.awt.*;
7   import java.awt.event.*;
8
9   // Java extension packages
10  import javax.swing.*;
11
12  public class StaticCharMethods2 extends JFrame {
13     private char c;
14     private int digit, radix;
15     private JLabel prompt1, prompt2;
16     private JTextField input, radixField;
17     private JButton toChar, toInt;
18
19     public StaticCharMethods2()
20     {
21        super( "Character Conversion Methods" );
22
23        // set up GUI and event handling
24        Container container = getContentPane();
25        container.setLayout( new FlowLayout() );
26
27        prompt1 = new JLabel( "Enter a digit or character " );
28        input = new JTextField( 5 );
29        container.add( prompt1 );
30        container.add( input );
31
32        prompt2 = new JLabel( "Enter a radix " );
33        radixField = new JTextField( 5 );
34        container.add( prompt2 );
35        container.add( radixField );
```

Outline

StaticCharMethods2.java

49

```
36
37        toChar = new JButton( "Convert digit to character" );
38
39        toChar.addActionListener(
40
41           // anonymous inner class
42           new ActionListener() {
43
44              // handle toChar JButton event
45              public void actionPerformed( ActionEvent actionEvent )
46              {
47                 digit = Integer.parseInt( input.getText() );
48                 radix =
49                    Integer.parseInt( radixField.getText() );
50                 JOptionPane.showMessageDialog( null,
51                    "Convert digit to character: " +
52                    Character.forDigit( digit, radix ) );
53              }
54
55           }  // end anonymous inner class
56
57        ); // end call to addActionListener
58
59        container.add( toChar );
60
61        toInt = new JButton( "Convert character to digit" );
62
63        toInt.addActionListener(
64
65           // anonymous inner class
66           new ActionListener() {
67
```

Outline

StaticCharMethods2.java

50

```
68             // handle toInt JButton event
69             public void actionPerformed( ActionEvent actionEvent )
70             {
71                String s = input.getText();
72                c = s.charAt( 0 );
73                radix =
74                   Integer.parseInt( radixField.getText() );
75                JOptionPane.showMessageDialog( null,
76                   "Convert character to digit: " +
77                   Character.digit( c, radix ) );
78             }
79
80          }  // end anonymous inner class
81
82       ); // end call to addActionListener
83
84       container.add( toInt );
85
86       setSize( 275, 150 );  // set the window size
87       show();               // show the window
88    }
89
90    // execute application
91    public static void main( String args[] )
92    {
93       StaticCharMethods2 application = new StaticCharMethods2();
94
95       application.addWindowListener(
96
97          // anonymous inner class
98          new WindowAdapter() {
99
```

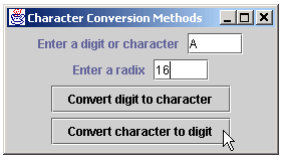
Outline

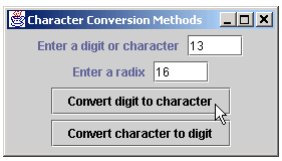
StaticCharMethods2.java

Line 77

51

```
100            // handle event when user closes window
101            public void windowClosing( WindowEvent windowEvent )
102            {
103               System.exit( 0 );
104            }
105
106         }  // end anonymous inner class
107
108      ); // end call to addWindowListener
109
110   }  // end method main
111
112}  // end class StaticCharMethods2
```





Outline

StaticCharMethods2.java

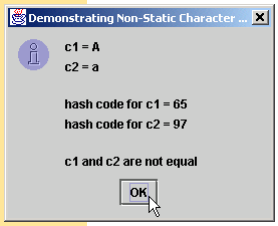
52

```
1   // Fig. 10.19: OtherCharMethods.java
2   // Demonstrate the non-static methods of class
3   // Character from the java.lang package.
4
5   // Java extension packages
6   import javax.swing.*;
7
8   public class OtherCharMethods {
9
10     // test non-static Character methods
11     public static void main( String args[] )
12     {
13        Character c1, c2;
14
15        c1 = new Character( 'A' );
16        c2 = new Character( 'a' );
17
18        String output = "c1 = " + c1.charValue() +
19           "\nc2 = " + c2.toString() +
20           "\n\nhash code for c1 = " + c1.hashCode() +
21           "\nhash code for c2 = " + c2.hashCode();
22
23        if ( c1.equals( c2 ) )
24           output += "\n\nc1 and c2 are equal";
25        else
26           output += "\n\nc1 and c2 are not equal";
27
28        JOptionPane.showMessageDialog( null, output,
29           "Demonstrating Non-Static Character Methods",
30           JOptionPane.INFORMATION_MESSAGE );
31
32        System.exit( 0 );
33     }
34
35  }  // end class OtherCharMethods
```

Outline

OtherCharMethods.java

Lines 18-23



10.20 Class `StringTokenizer`

- Tokenizer
  - Partition **`String`** into individual substrings
  - Use *delimiter*
  - Java offers **`java.util.StringTokenizer`**

```
1   // Fig. 10.20: TokenTest.java
2   // Testing the StringTokenizer class of the java.util package
3
4   // Java core packages
5   import java.util.*;
6   import java.awt.*;
7   import java.awt.event.*;
8
9   // Java extension packages
10  import javax.swing.*;
11
12  public class TokenTest extends JFrame {
13     private JLabel promptLabel;
14     private JTextField inputField;
15     private JTextArea outputArea;
16
17     // set up GUI and event handling
18     public TokenTest()
19     {
20        super( "Testing Class StringTokenizer" );
21
22        Container container = getContentPane();
23        container.setLayout( new FlowLayout() );
24
25        promptLabel =
26           new JLabel( "Enter a sentence and press Enter" );
27        container.add( promptLabel );
28
29        inputField = new JTextField( 20 );
30
31        inputField.addActionListener(
32
33           // anonymous inner class
34           new ActionListener() {
35
```

Outline

TokenTest.java

Line 29


```
36              // handle text field event
37              public void actionPerformed( ActionEvent event )
38              {
39                 String stringToTokenize =
40                    event.getActionCommand();
41                 StringTokenizer tokens =
42                    new StringTokenizer( stringToTokenize );
43
44                 outputArea.setText( "Number of elements: " +
45                    tokens.countTokens() + "\nThe tokens are:\n" );
46
47                 while ( tokens.hasMoreTokens() )
48                    outputArea.append( tokens.nextToken() + "\n" );
49              }
50
51           }  // end anonymous inner class
52
53        ); // end call to addActionListener
54
55        container.add( inputField );
56
57        outputArea = new JTextArea( 10, 20 );
58        outputArea.setEditable( false );
59        container.add( new JScrollPane( outputArea ) );
60
61        setSize( 275, 260 );  // set the window size
62        show();               // show the window
63     }
64
65     // execute application
66     public static void main( String args[] )
67     {
68        TokenTest application = new TokenTest();
69
```

Outline

TokenTest.java

Lines 41-42

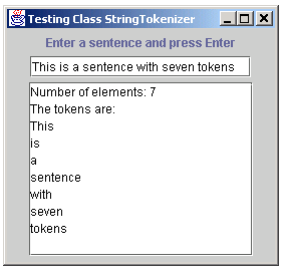
Line 45

Lines 47-48

```
70         application.addWindowListener(
71
72            // anonymous inner class
73            new WindowAdapter() {
74
75               // handle event when user closes window
76               public void windowClosing( WindowEvent windowEvent )
77               {
78                  System.exit( 0 );
79               }
80
81            }  // end anonymous inner class
82
83         ); // end call to addWindowListener
84
85      }  // end method main
86
87   }  // end class TokenTest
```

Outline

TokenTest.java



57

10.21 Card Shuffling and Dealing Simulation

- Develop **`DeckOfCards`** application
  - Create deck of 52 playing cards using Card objects
  - User deals card by clicking "**`Deal card`**" button
  - User shuffles deck by clicking "**`Shuffle cards`**" button
  - Use random-number generation

58

```
1   // Fig. 10.21: DeckOfCards.java
2   // Card shuffling and dealing program
3
4   // Java core packages
5   import java.awt.*;
6   import java.awt.event.*;
7
8   // Java extension packages
9   import javax.swing.*;
10
11  public class DeckOfCards extends JFrame {
12     private Card deck[];
13     private int currentCard;
14     private JButton dealButton, shuffleButton;
15     private JTextField displayField;
16     private JLabel statusLabel;
17
18     // set up deck of cards and GUI
19     public DeckOfCards()
20     {
21        super( "Card Dealing Program" );
22
23        String faces[] = { "Ace", "Deuce", "Three", "Four",
24           "Five", "Six", "Seven", "Eight", "Nine", "Ten",
25           "Jack", "Queen", "King" };
26        String suits[] =
27           { "Hearts", "Diamonds", "Clubs", "Spades" };
28
29        deck = new Card[ 52 ];
30        currentCard = -1;
31
32        // populate deck with Card objects
33        for ( int count = 0; count < deck.length; count++ )
34           deck[ count ] = new Card( faces[ count % 13 ],
35              suits[ count / 13 ] );
```

Outline

DeckOfCards.java

Lines 19 and 29

Line 30

Lines 33-35

59

```
36
37        // set up GUI and event handling
38        Container container = getContentPane();
39        container.setLayout( new FlowLayout() );
40
41        dealButton = new JButton( "Deal card" );
42        dealButton.addActionListener(
43
44           // anonymous inner class
45           new ActionListener() {
46
47              // deal one card
48              public void actionPerformed( ActionEvent actionEvent )
49              {
50                 Card dealt = dealCard();
51
52                 if ( dealt != null ) {
53                    displayField.setText( dealt.toString() );
54                    statusLabel.setText( "Card #: " + currentCard );
55                 }
56                 else {
57                    displayField.setText(
58                       "NO MORE CARDS TO DEAL" );
59                    statusLabel.setText(
60                       "Shuffle cards to continue" );
61                 }
62              }
63
64           }  // end anonymous inner class
65
66        ); // end call to addActionListener
67
68        container.add( dealButton );
69
```

Outline

DeckOfCards.java

Line 50

Line 53

60

```
70          shuffleButton = new JButton( "Shuffle cards" );
71          shuffleButton.addActionListener(
72
73             // anonymous inner class
74             new ActionListener() {
75
76                // shuffle deck
77                public void actionPerformed( ActionEvent actionEvent )
78                {
79                   displayField.setText( "SHUFFLING ..." );
80                   shuffle();
81                   displayField.setText( "DECK IS SHUFFLED" );
82                }
83
84             }  // end anonymous inner class
85
86          ); // end call to addActionListener
87
88          container.add( shuffleButton );
89
90          displayField = new JTextField( 20 );
91          displayField.setEditable( false );
92          container.add( displayField );
93
94          statusLabel = new JLabel();
95          container.add( statusLabel );
96
97          setSize( 275, 120 );  // set window size
98          show();               // show window
99       }
100
```

Outline

DeckOfCards.java

Line 80

61

```
101      // shuffle deck of cards with one-pass algorithm
102      public void shuffle()
103      {
104         currentCard = -1;
105
106         // for each card, pick another random card and swap them
107         for ( int first = 0; first < deck.length; first++ ) {
108            int second =  ( int ) ( Math.random() * 52 );
109            Card temp = deck[ first ];
110            deck[ first ] = deck[ second ];
111            deck[ second ] = temp;
112         }
113
114         dealButton.setEnabled( true );
115      }
116
117      // deal one card
118      public Card dealCard()
119      {
120         if ( ++currentCard < deck.length )
121            return deck[ currentCard ];
122         else {
123            dealButton.setEnabled( false );
124            return null;
125         }
126      }
127
128      // execute application
129      public static void main( String args[] )
130      {
131         DeckOfCards app = new DeckOfCards();
132
133         app.addWindowListener(
134
```

Outline

DeckOfCards.java

Lines 102-115

Lines 118-126

Lines 114 and 123

62

```
135          // anonymous inner class
136          new WindowAdapter() {
137
138             // terminate application when user closes window
139             public void windowClosing( WindowEvent windowEvent )
140             {
141                System.exit( 0 );
142             }
143
144          }  // end anonymous inner class
145
146       ); // end call to addWindowListener
147
148    }  // end method main
149
150 }  // end class DeckOfCards
151
152 // class to represent a card
153 class Card {
154    private String face;
155    private String suit;
156
157    // constructor to initialize a card
158    public Card( String cardFace, String cardSuit )
159    {
160       face = cardFace;
161       suit = cardSuit;
162    }
163
```

Outline

DeckOfCards.java

Lines 154-155


```
164    // return String represenation of Card
165    public String toString()
166    {
167       return face + " of " + suit;
168    }
169
170 }  // end class Card
```

Outline

DeckOfCards.java