```
 1  // Example 2.1 While1Demo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class for illustrating the while-statement.  A typical run:
 7  //
 8  // %> java While1Demo
 9  // Enter integer (a negative to stop): 4
10  // 4 is non-negative.
11  // Enter integer (a negative to stop): -3
12  //
13  // Enter integer (a negative to stop): 5
14  // 5 is non-negative.
15  // Enter integer (a negative to stop): -2
16  // Finally, enter integer you want to count to: 12
17  // Count 1 to 12: 1 2 3 4 5 6 7 8 9 10 11 12
18
19  public class While1Demo
20  {
21      public static void main( String[] args )
22      {
23          Terminal terminal = new Terminal();  // for input and output
24
25          // while tests a condition
26          int n = terminal.readInt("Enter integer (a negative to stop): ");
27          while ( n >= 0 ) {
28              terminal.println( n + " is non-negative." );
29              n = terminal.readInt("Enter integer (a negative to stop): ");
30          }
31          terminal.println();
32
33          // while tests a boolean variable
34          boolean more = true;
35          while ( more ) {
36              n = terminal.readInt("Enter integer (a negative to stop): ");
37              if ( n >= 0 ) {
38                  terminal.println( n + " is non-negative." );
39              }
40              else {
41                  more = false;
42              }
43          }
44
45          // while used for counting
46          n = terminal.
47              readInt("Finally, enter integer you want to count to: ");
48          int i = 1;
49          terminal.print( "Count 1 to " + n + ":" );
50          while ( i <= n ) {
51              terminal.print( " " + i );
52              i++;  //  same as i = i + 1
53          }
54          terminal.println();
55      }
56  }
```

```
 1  // Example 2.2 joi/examples/While2Demo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class for illustrating the while-statement.  A typical run:
 7  //
 8  // %> java While2Demo
 9  // Enter integer: 10
10  // Fibonacci numbers <= 10: 1 1 2 3 5 8
11  // Fibonacci numbers <= 10: 1 1 2 3 5 8
12  // First 10 Fibonacci numbers: 1 1 2 3 5 8 13 21 34 55
13
14  public class While2Demo
15  {
16      public static void main( String[] args )
17      {
18          Terminal terminal = new Terminal();  // for input and output
19
20          // Prompt for and read a single integer.
21          int n = terminal.readInt( "Enter integer: " );
22
23          // while tests a condition
24          terminal.print( "Fibonacci numbers <= " + n + ":" );
25          int thisOne = 1;
26          int lastOne = 1;
27          while ( lastOne <= n ) {
28              terminal.print( " " + lastOne );
29              int nextOne = thisOne + lastOne;
30              lastOne = thisOne;
31              thisOne = nextOne;
32          }
33          terminal.println();
34
35          // while tests a boolean variable
36          terminal.print( "Fibonacci numbers <= " + n + ":" );
37          thisOne = 1;
38          lastOne = 1;
39          boolean more = true;
40          while ( more ) {
41              if ( lastOne > n ) {
42                  more = false;
43              }
44              else {
45                  terminal.print( " " + lastOne );
46                  int nextOne = thisOne + lastOne;
47                  lastOne = thisOne;
48                  thisOne = nextOne;
49              }
50          }
51          terminal.println();
52
53          // while used for counting
54          terminal.print( "First " + n + " Fibonacci numbers:" );
55          thisOne = 1;
56          lastOne = 1;
```



```
57          int i = 1;
58          while ( i <= n ) {
59              terminal.print( " " + lastOne );
60              int nextOne = thisOne + lastOne;
61              lastOne = thisOne;
62              thisOne = nextOne;
63              i++;  // same as 'i = i + 1;'
64          }
65          terminal.println();
66      }
67  }
```

```
 1  // Example 2.3 IfDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6
 7  // A class illustrating the if-statement. A typical run:
 8  //
 9  // %> java IfDemo
10  // Enter an integer: 0
11  // If 0 is negative, say hello:
12  // isNegative is false
13  // The integer 0 is zero
14  // Finally: 0 is still nonnegative
15  // because it's zero
16
17  public class IfDemo
18  {
19      public static void main( String[] args )
20      {
21          Terminal terminal = new Terminal();  // for input and output
22
23          // Prompt for and read a single integer.
24          int var = terminal.readInt( "Enter an integer: " );
25
26          // simple if statement
27          terminal.println( "If " + var + " is negative, say hello:" );
28          if (var < 0) {
29              terminal.println( "hello" );
30          }
31
32          // an if-else statement testing a boolean variable
33          boolean isNegative = ( var < 0 );
34          if (isNegative) {
35              terminal.println( "isNegative is true" );
36          }
37          else {
38              terminal.println( "isNegative is false" );
39          }
40
41          // if-else-if statement
42          terminal.print( "The integer " + var + " is ");
43          if (var > 0) {
44              terminal.println("positive");
45          }
46          else if (var < 0) {
47              terminal.println( "negative" );
48          }
49          else { // just one case left!
50              terminal.println( "zero" );
51          }
52
53          // finally, nested if-(if)-else: note the indenting
54          terminal.print( "Finally: " + var + " is still ");
55          if (var >= 0) {
56              terminal.println("nonnegative");
```



```
57              if (var == 0 ) {
58                  terminal.println("because it's zero ");
59              }
60          }
61          else {
62              terminal.println( "negative" );
63          }
64      }
65  }
```

```
 1  // Example 3.1 joi/examples/StaticDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // Demonstrate the interplay between static members (fields and methods)
 7  // and instance (non-static) members.
 8  //
 9  // %> java StaticDemo
10  // 0: counter = 1; objectField = 0
11  // StaticDemo.classMethod() = 100
12  // classMethod() = 100
13  // 1: counter = 2; objectField = 1
14  // StaticDemo.classMethod() = 101
15  // classMethod() = 101
16  // 2: counter = 3; objectField = 2
17  // StaticDemo.classMethod() = 103
18  // classMethod() = 103
19  // 3: counter = 4; objectField = 3
20  // StaticDemo.classMethod() = 106
21  // classMethod() = 106
22  // 4: counter = 5; objectField = 4
23  // StaticDemo.classMethod() = 110
24  // classMethod() = 110
25  //
26  // last classMethod() = 110
27
28  public class StaticDemo
29  {
30      // Declare three (static) class variables
31      // A class variable is associated with the (one) class.
32
33      private static int counter  = 0;
34      private static int classVar = 0;
35      private static Terminal terminal = new Terminal();
36
37      int objectField = 0;     // an instance variable; one per object
38
39      // The constructor keeps track of how many StaticDemo objects
40      // have been constructed.
41
42      public StaticDemo( int objectFieldValue )
43      {
44          objectField = objectFieldValue; // set the instance variable
45          counter++;  // increment counter (counting the StaicDemos made)
46      }
47
48      public void instanceMethod()
49      {
50          // Instance methods can refer to both instance variables
51          // and class variables.
52
53          terminal.println( "counter = " + counter +
54                            "; objectField = " + objectField );
55          classVar = classVar + objectField;
56
```



```
57      }
58
59      public static int classMethod()
60      {
61          // Class methods may refer only to class variables
62          // (and other class methods), as well as to local variables.
63
64          // What happens if we comment out the next line?
65          int counter = 100;
66
67          return counter + classVar;
68      }
69
70      public static void main( String[] args )
71      {
72          for (int i = 0; i < 5; i++) {
73              StaticDemo sd = new StaticDemo( i );
74              terminal.print( i + ": " );
75              sd.instanceMethod();
76
77              // classMethod()
78              //   is equivalent to
79              // StaticDemo.classMethod()
80              terminal.println( "StaticDemo.classMethod() = "
81                                + StaticDemo.classMethod() );
82              terminal.println( "classMethod() = "
83                                + classMethod() );
84          }
85          terminal.println();
86          terminal.println( "last classMethod() = " + classMethod() );
87      }
88  }
```

```
 1  // Example 3.2 joi/examples/ForDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class illustrating the For-statement.  A typical run:
 7  //
 8  // %> java ForDemo
 9  // Enter integer: 7
10  // 7 integers starting at 0: 0 1 2 3 4 5 6
11  // First 7 Fibonacci numbers   (for): 1 1 2 3 5 8 13
12  // First 7 Fibonacci numbers (while): 1 1 2 3 5 8 13
13  // 49 @'s:
14  // @@@@@@@
15  // @@@@@@@
16  // @@@@@@@
17  // @@@@@@@
18  // @@@@@@@
19  // @@@@@@@
20  // @@@@@@@
21
22  public class ForDemo
23  {
24      public static void main( String[] args )
25      {
26          Terminal terminal = new Terminal();  // for input and output
27
28          // Prompt for and read a single integer.
29          int n = terminal.readInt( "Enter integer:" );
30
31          terminal.print( n + " integers starting at 0:" );
32          for ( int i = 0; i < n; i++ ) {
33              terminal.print( " " + i );  // all one one line
34          }
35          terminal.println();             // the newline
36
37          // Build Fibonacci numbers 1, 1, 2, 3, 5, 8,
38          // by adding last two together to make the next
39          // Use three int variables and a loop:
40
41          int thisOne, lastOne, nextOne;
42          terminal.println( "First " + n + " Fibonacci numbers:" );
43
44          terminal.print("(for):               " );
45          thisOne = 1;
46          lastOne = 1;
47          for ( int i = 1; i <= n; i++ ) {
48              terminal.print( " " + lastOne );
49              nextOne = thisOne + lastOne;
50              lastOne = thisOne;
51              thisOne = nextOne;
52          }
53          terminal.println();
54
55          // Since i is never used in the body of the previous loop
56          // we can count down to get the same output:
```



```
57          terminal.print("(for, counting down):" );
58          thisOne = 1;
59          lastOne = 1;
60          for ( int counter = n; counter > 0; counter-- ) {
61              terminal.print( " " + lastOne );
62              nextOne = thisOne + lastOne;
63              lastOne = thisOne;
64              thisOne = nextOne;
65          }
66          terminal.println();
67
68          // Replace the for loop with a while loop
69          terminal.print("(while):             " );
70          thisOne = 1;
71          lastOne = 1;
72          int i = 1;
73          while ( i <= n ) {
74              terminal.print( " " + lastOne );
75              nextOne = thisOne + lastOne;
76              lastOne = thisOne;
77              thisOne = nextOne;
78              i++;
79          }
80          terminal.println();
81
82          terminal.println("Nested for loops: " + (n*n) + " @'s:" );
83          for ( int row = 1; row <= n ; row++ ) {
84              for ( int col = 1; col <= n; col++ ) {
85                  terminal.print( " @" );
86              }
87              terminal.println();
88          }
89      }
90  }
```

```
 1  // Example 3.3 joi/examples/BreakAndContinueDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  public class BreakAndContinueDemo
 7  {
 8      private static Terminal t = new Terminal();
 9
10      public static void main( String[] args )
11      {
12          t.println("invoking loop");
13          BreakAndContinueDemo.loop();  // could say just loop();
14          t.println("returned from loop, leaving main");
15      }
16
17      private static void loop()
18      {
19          t.println("starting infinite loop");
20          while( true ) {
21              String command = t.readWord(
22                  "normal, break, continue, return, exit, oops ? > ");
23              if (command.startsWith("n")) {
24                  t.println("normal flow of control");
25              }
26              if (command.startsWith("b")) {
27                  t.println("break from looping");
28                  break;
29              }
30              if (command.startsWith("c")) {
31                  t.println("continue looping");
32                  continue;
33              }
34              if (command.startsWith("r")) {
35                  t.println("return prematurely from loop method");
36                  return;
37              }
38              if (command.startsWith("e")) {
39                  t.println("exit prematurely from program");
40                  System.exit(0);
41              }
42              if (command.startsWith("o")) {
43                  t.println("program about to crash ...");
44                  Terminal foo = null;
45                  foo.println("crash the program");
46              }
47              t.println("last line in loop body");
48          }
49          t.println("first line after loop body");
50          t.println("returning normally from loop method");
51      }
52  }
```

```
 1  // Example 4.1 joi/examples/CommandLineArgsDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class illustrating the use of command line arguments.
 7  //
 8  // %> java CommandLineArgsDemo foo      bar "b q"
 9  // Echo command line arguments,
10  // surrounded by |...|
11  // |foo|
12  // |bar|
13  // |b q|
14  //
15  // Note the use of quotes to get embedded blanks.
16
17  public class CommandLineArgsDemo
18  {
19      public static void main( String[] args )
20      {
21          System.out.println("Echo command line arguments, ");
22          System.out.println("surrounded by |...| ");
23          for (int i = 0; i < args.length; i++) {
24              System.out.println('|' + args[i] + '|');
25          }
26      }
27  }
```

```
 1  // Example 4.2 joi/examples/ArrayDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class illustrating arrays
 7  //
 8  // Build an array of Fibonacci numbers 1, 1, 2, 3, 5, 8, ...
 9  // and play with it. Sample output:
10  //
11  // %> java ArrayDemo 8
12  // Sum first 8 Fibonacci numbers
13  // 1 1 2 3 5 8 13 21
14  // total: 54
15  //
16  // First 8 Fibonacci numbers (reverse order)
17  // 21 13 8 5 3 2 1 1
18  // Every other fib
19  // 1    1
20  // 3    2
21  // 5    5
22  // 7    13
23
24  public class ArrayDemo
25  {
26      public static void main( String[] args )
27      {
28          int n = 6; // default
29          if (args.length > 0) {
30              n = Integer.parseInt(args[0]);
31          }
32
33          int[] fibs = new int[n]; // declare and create array
34
35          fibs[0] = fibs[1] = 1;   // fill first two positions
36          for ( int i = 2; i < n; i++ ) { // fill the rest
37              fibs[i] = fibs[i-1] + fibs[i-2];
38          }
39
40          // standard idiom for accumulating total of an array
41          int total = 0;
42          System.out.println("Sum first " + n + " Fibonacci numbers");
43          for ( int i = 0; i < n; i++ ) {
44              System.out.print(fibs[i] + " ");
45              total += fibs[i];
46          }
47          System.out.println("\ntotal: " + total);
48          System.out.println();
49
50          System.out.
51              println("First " + n + " Fibonacci numbers (reverse order)");
52          for ( int i = n-1; i >= 0 ; i-- ) {
53              System.out.print(fibs[i] + " ");
54          }
55          System.out.println();
56
```



```
57          System.out.println("Every other fib");
58          for ( int i = 0; i < n ; i += 2 ) {
59              System.out.println((i+1) + "\t" + fibs[i]);
60          }
61          System.out.println();
62      }
63  }
```

```
// Example 4.3 ArrayListDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

// Tell the java compiler that the ArrayList class is in
// the java.util part of the library.

import java.util.ArrayList;

//  Exercise the most important parts of the ArrayList API.
//
// %> java ArrayListDemo
// Create a list containing three SimpleObjects.
// 0    zero
// 1    one
// 2    two
// Replace the object at position 0.
// Put a new object at 2 and push the rest along.
// Print out the list again.
// 0    new zero
// 1    one
// 2    one point five
// 3    two

public class ArrayListDemo
{
    public static void main( String[] args )
    {
        System.out.println("Create a list containing three SimpleObjects.

        // Create a new, empty ArrayList
        // with the ArrayList constructor.
        ArrayList myList = new ArrayList();

        // Put three things on it with the add()
        // method - each add appends to the list.
        myList.add(new SimpleObject("zero"));
        myList.add(new SimpleObject("one"));
        myList.add(new SimpleObject("two"));

        // Print the list with a for loop.
        // size() method tells how long the list is.
        // get(int index) method retrieves value stored at position index
        // The (SimpleObject) cast tells Java what type of thing you got
        for (int i = 0; i < myList.size() ; i++ ) {
            SimpleObject foo = (SimpleObject)myList.get(i);
            System.out.println(i + "\t" + foo.name);
        }

        // set(int index) method changes value stored at position index
        System.out.println("Replace the object at position 0.");
        myList.set(0, new SimpleObject("new zero"));

        System.out.println("Put a new object at 2 and push the rest along
        myList.add(2, new SimpleObject("one point five"));
```



```

        System.out.println("Print out the list again.");
        for (int i = 0; i < myList.size() ; i++ ) {
            SimpleObject foo = (SimpleObject)myList.get(i); // note cast!
            System.out.println(i + "\t" + foo.name);
        }
    }

    // This really simple class exists only to provide
    // things to put in the ArrayList.
    //
    // It's an inner class, declared inside the ArrayListDemo
    // class, which is its scope.
    //
    // Since it's visible only here, we are using a public
    // name field rather than a private field and a public
    // getName()

    private static class SimpleObject
    {
        public String name;

        public SimpleObject( String name )
        {
            this.name = name;
        }
    } // end of body of inner class SimpleObject
} // end of body of ArrayList Demo
```

```
// Example 4.4 joi/examples/TreeMapDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.util.TreeMap;
import java.util.Iterator;
import java.util.Set;
import java.util.Collection;
import java.util.Map;

// A class illustrating the use of TreeMap.  A typical run:
//
// %> java TreeMapDemo
// Store 3 wrapped ints, keys "one", "two", "three".
// The wrapped int stored for "two" is 2
//
// Iterate over keys, get each value.
// Note that key order is aphabetical:
// The value for key one is 1
// The value for key three is 3
// The value for key two is 2
//
// Iterate over the values:
// 1
// 3
// 2
//
// Iterate over the key-value pairs:
// The value for the entry with key one is 1
// The value for the entry with key three is 3
// The value for the entry with key two is 2
//
// How a TreeMap represents itself as a String:
// {one=1, three=3, two=2}
//
// Store a different value at key "two"
// {one=1, three=3, two=2222}
//
// Store map.get( "one" ) at key "two"
// {one=1, three=3, two=1}
//
// A TreeMap with Integer keys mapping to String values
// {1=I, 2=II, 3=III}
// %>

public class TreeMapDemo
{
    public static void main( String[] args )
    {
        Terminal terminal = new Terminal(); // for input and output

        TreeMap map = new TreeMap();

        // Put in some ints (each wrapped up as an Integer object)
        terminal.println(
```



```
          "Store 3 wrapped ints, keys \"one\", \"two\", \"three\".");
        map.put("one",   new Integer(1) );
        map.put("two",   new Integer(2) );
        map.put("three", new Integer(3) );

        // get the value associated with a key;
        // notice the required cast.
        Integer wrappedInt = (Integer) map.get( "two" );

        // And print the wrapped int
        terminal.println( "The wrapped int stored for \"two\" is "
                          + wrappedInt);


        // The set of keys.
        Set keys = map.keySet();
        // The iterator over this "set" of keys will return
        // the keys in key-order.
        terminal.println( "\nIterate over keys, get each value." );
        terminal.println( "Note that key order is aphabetical:" );
        Iterator keysIterator = keys.iterator();
        while ( keysIterator.hasNext() ) {
            String key = (String) keysIterator.next();
            terminal.println( "The value for key " + key + " is "
                              + ((Integer) map.get( key)) );
        }

        // Iterate over the collection of values;
        // notice the order is the same (ie the key-order).
        terminal.println( "\nIterate over the values:" );
        Iterator valuesIterator = map.values().iterator();
        while ( valuesIterator.hasNext() ) {
            terminal.println( ((Integer) valuesIterator.next()));
        }

        // The set of Map.Entry objects (key-value pairs);
        // Map.Entry is an inner class of Map.

        // Iterate over the entries.
        terminal.println( "\nIterate over the key-value pairs:" );
        Iterator entriesIterator = map.entrySet().iterator();
        while ( entriesIterator.hasNext() ) {
            Map.Entry entry = (Map.Entry) entriesIterator.next();
            terminal.println( "The value for the entry with key "
                      + entry.getKey() + " is "
                      + ((Integer) entry.getValue()));
        }

        // how a TreeMap represents itself as a String:
        terminal.println(
            "\nHow a TreeMap represents itself as a String:");
        terminal.println(map.toString());
        terminal.println();

        // We can overwrite the value stored under a key
        terminal.println(
```

```
113             "Store a different value at key \"two\"");
114         map.put("two", new Integer(2222));
115         terminal.println(map.toString());
116         terminal.println();
117
118         // We can store the same value under two keys
119         terminal.println(
120             "Store map.get( \"one\" ) at key \"two\"");
121         map.put("two", map.get( "one" ) );
122         terminal.println(map.toString());
123         terminal.println();
124
125         // And keys don't necessarily have to be Strings;
126         // Here's a TreeMap mapping Integers to strings.
127         terminal.println(
128             "A TreeMap with Integer keys mapping to String values");
129         map = new TreeMap();
130         map.put( new Integer( 1 ), "I" );
131         map.put( new Integer( 2 ), "II" );
132         map.put( new Integer( 3 ), "III" );
133         terminal.println(map.toString());
134         terminal.println();
135     }
136 }
```

```
// Example 5.1 joi/examples/SwitchDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

// A class illustrating the Switch statement

// %> java SwitchDemo
// Enter an integer: 2
// two
// Notice the importance of the breaks!
// The same statement without the breaks:
// two
// three
// Not one, two or three!
// Enter a character: y
// yes
// %>

public class SwitchDemo
{
    public static void main( String[] args )
    {
        Terminal terminal = new Terminal();

        int i = terminal.readInt( "Enter an integer: " );

        switch (i) {
        case 1:
            terminal.println( "one" );
            break;
        case 2:
            terminal.println( "two" );
            break;
        case 3:
            terminal.println( "three" );
            break;
        default:
            terminal.println( "Not one, two or three!" );
        }

        terminal.println( "Notice the importance of the breaks!" );
        terminal.println( "The same statement without the breaks:" );

        switch (i) {
        case 1:
            terminal.println( "one" );
        case 2:
            terminal.println( "two" );
        case 3:
            terminal.println( "three" );
        default:
            terminal.println( "Not one, two or three!" );
        }

        switch (terminal.readChar( "Enter a character: " )) {
```



```
        case 'y':
            terminal.println( "yes" );
            break;
        case 'n':
            terminal.println( "no" );
            break;
        default:
            terminal.println( "Neither yes nor no." );
        }
    }
}
```

```
 1  // Example 5.2 joi/examples/OverridingDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // Small program to illustrate overriding and toString()
 7  //
 8  // Here's what the output looks like:
 9  //
10  // %> java Overriding jessica benjamin
11  // Terminal t = new Terminal();
12  // nobj = new NamedObject( args[0] );
13  // nobj.toString():           jessica
14  // nobj:                      jessica
15  // nobj.toStringfromObject(): NamedObject@206fdf64
16  // nobj = new NamedObject( args[1] );
17  // nobj.toString():           benjamin
18  // nobj:                      benjamin
19  // nobj.toStringfromObject(): NamedObject@2103df64
20  //
21  // toString():                Terminal@201bdf64
22  // t:                         Terminal@201bdf64
23
24  public class OverridingDemo
25  {
26      public static void main( String[] args )
27      {
28          Terminal t = new Terminal();
29          NamedObject nobj;
30
31          t.println("Terminal t = new Terminal();");
32          t.println("nobj = new NamedObject( args[0] );");
33          nobj = new NamedObject( args[0] );
34          t.println( "nobj.toString():           " + nobj.toString() );
35          t.println( "nobj:                      " + nobj );
36          t.println( "nobj.toStringfromObject(): " +
37                     nobj.toStringfromObject());
38
39          t.println("nobj = new NamedObject( args[1] );");
40          nobj = new NamedObject( args[1] );
41          t.println( "nobj.toString():           " + nobj.toString() );
42          t.println( "nobj:                      " + nobj );
43          t.println( "nobj.toStringfromObject(): " +
44                     nobj.toStringfromObject());
45
46          t.println( "\ntoString():               " + t.toString() );
47          t.println( "t:                         " + t );
48      }
49  }
50
51  // A simple class whose instances have one field
52  // and several toString methods. Visible only inside
53  // the OverridingDemo class.
54
55  // you can put two classes in one file as long as only one of them
56  // is public
```



```
57
58  class NamedObject // extends Object, by default
59  {
60      private String name;
61
62      // constructor does the obvious thing
63
64      public NamedObject( String name )
65      {
66          this.name = name;
67      }
68
69      // override toString in class Object
70
71      public String toString()
72      {
73          return name;
74      }
75
76      // access to the overridden method using super
77
78      public String toStringfromObject()
79      {
80          return super.toString();
81      }
82  }
```

```
// Example 5.3 joi/examples/EqualsDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

// A class illustrating == and equals().
//
// %> java EqualsDemo
// Different objects, same field:
// e1 == e1LookAlike ->         false
// e1.equals( e1LookAlike ) ->  true
// Same object:
// e1 == e1Too ->               true
//
// Different ArrayLists with equal (but not ==) elements:
// alist0 == alist1 ->      false
// alist0.equals(alist1) -> true
//
// Different TreeMaps with equal keys
// mapping to equal (but !=) values:
// tmap0 == tmap1 ->      false
// tmap0.equals(tmap1) -> true
//
// tmap0.toString() -> {sillykey = EqualsDemo value 1}
// tmap1.toString() -> {sillykey = EqualsDemo value 1}
// Are these Strings == ?    false
// Are these Strings equal ? true

import java.util.ArrayList;
import java.util.TreeMap;

public class EqualsDemo
{
    // Properties of an EqualsDemo object.
    //
    // Override equals: two of these objects are equal if
    // their integer field has the same (i.e. ==) value.
    //
    // When you override equals it's customary to override
    // toString too, so that equal objects return equal
    // Strings, so we do.

    private int field;

    public EqualsDemo( int field )
    {
        this.field = field;
    }

    public boolean equals( Object other )
    {
        return (other instanceof EqualsDemo)
            && (this.field == ((EqualsDemo)other).field);
    }

    public String toString()
```



```
    {
        return " EqualsDemo value " + field;
    }

    public static void main( String[] args )
    {
        Terminal t = new Terminal();

        // EqualsDemo object == vs equals()
        EqualsDemo e1          = new EqualsDemo( 1 );
        EqualsDemo e1LookAlike = new EqualsDemo( 1 ); // same field.
        EqualsDemo e1Too       = e1;                  // same object

        t.println("Different objects, same field:");
        t.println("e1 == e1LookAlike ->         " + (e1 == e1LookAlike));
        t.println("e1.equals( e1LookAlike ) ->  " + e1.equals( e1LookAlik
        t.println("Same object:");
        t.println("e1 == e1Too ->               " + (e1 == e1Too));
        t.println();

        // Arrays and Maps
        ArrayList alist0 = new ArrayList();
        ArrayList alist1 = new ArrayList();

        alist0.add( e1);
        alist1.add( e1LookAlike );

        t.println( "Different ArrayLists with equal (but not ==) elements
        t.println( "alist0 == alist1 ->      "  + (alist0 == alist1));
        t.println( "alist0.equals(alist1) -> "  + alist0.equals(alist1));
        t.println();

        TreeMap tmap0 = new TreeMap();
        TreeMap tmap1 = new TreeMap();

        tmap0.put( "sillykey ", e1 );
        tmap1.put( "sillykey ", e1LookAlike );

        t.println( "Different TreeMaps with equal keys" );
        t.println( "mapping to equal (but !=) values:" );

        t.println( "tmap0 == tmap1 ->      " + (tmap0 == tmap1));
        t.println( "tmap0.equals(tmap1) -> " + tmap0.equals(tmap1));
        t.println();

        // Test Strings for == and equal
        String s0 = tmap0.toString();
        String s1 = tmap1.toString();
        t.println( "tmap0.toString() -> " + s0);
        t.println( "tmap1.toString() -> " + s1);
        t.println( "Are these Strings == ?    " + (s0 == s1) );
        t.println( "Are these Strings equal ? " + (s0.equals(s1)) );
    }
}
```

```
 1  // Example 7.1 joi/examples/RumpelStiltskinDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // Practice with simple exceptions - some from the API, one declared here
 7
 8  public class RumpelStiltskinDemo
 9  {
10      public static void main( String[] args )
11      {
12          Wizard rumpelstiltskin = new Wizard("RumpelStiltskin");
13          Wizard aNullWizard     = null;
14
15          // see if the first command line argument ( args[0] )
16          // is the right name
17          try {
18              System.out.println("Is your name " + args[0] +'?');
19              rumpelstiltskin.guessName(args[0]);
20              System.out.println("Yes! How did you guess?");
21              System.exit(0);
22          }
23          // come here right away if there is no args[0]
24          catch (IndexOutOfBoundsException e) {
25              System.out.println( "usage: java RumpelStiltskinDemo guess");
26              System.exit(0); // leave the program gracefully
27          }
28          // come here from guessName if exception thrown
29          catch (BadGuessException e) {
30              System.err.println("sorry - " + args[0] + " is not my name");
31          }
32
33          System.out.println(
34             "\nIntentionally generate a NullPointerException,\n" +
35             "see what the Exception's toString method returns");
36          try {
37              aNullWizard.guessName("who am I?");
38          }
39          catch (Exception e) {
40              System.out.println(e);
41          }
42
43          System.out.println(
44             "\nExperiment with the printStackTrace() method:");
45          try {
46              rumpelstiltskin.makeMischief();
47          }
48          catch (Exception e) {
49              e.printStackTrace();
50          }
51
52          // perhaps throw an uncaught IndexOutOfBoundsException
53          System.out.println(
54              "\nLook for a second command line argument,\n" +
55              "see what happens if it's not there:" );
56          System.out.println(args[1]);
```



```
57      }
58
59      // two inner classes, used only in this file
60      private static class Wizard
61      {
62          private String name;
63
64          public Wizard( String name )
65          {
66              this.name = name;
67          }
68
69          public void guessName( String name )
70              throws BadGuessException
71          {
72              if (!name.equals(this.name))
73                  throw new BadGuessException( );
74          }
75
76          public void makeMischief()
77              throws BadGuessException
78          {
79              this.guessName("??");
80          }
81      }
82
83      private static class BadGuessException extends Exception
84      {
85          // empty body
86      }
87  }
```

```
 1  // Example 8.1 joi/examples/EscapeDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class illustrating the escape character '\' in quoted Strings
 7  //
 8  // %> java EscapeDemo
 9  // argument to println     output
10  // "hello world"           hello world
11  // "hello\nworld"          hello
12  // world
13  // "\"hello world\""       "hello world"
14  // "hello\tworld"          hello   world
15  // "hello\bworld"          hellworld
16  //
17  // Note the use of quotes to get embedded blanks.
18
19  public class EscapeDemo
20  {
21      public static void main( String[] args )
22      {
23          System.out.println("argument to println\toutput");
24
25          System.out.print("\"hello world\"\t\t");
26          System.out.println("hello world");
27
28          System.out.print("\"hello\\nworld\"\t\t");
29          System.out.println("hello\nworld");
30
31          System.out.print("\"\\\"hello world\\\"\"\t");
32          System.out.println("\"hello world\"");
33
34          System.out.print("\"hello\\tworld\"\t\t");
35          System.out.println("hello\tworld");
36
37          System.out.print("\"hello\\bworld\"\t\t");
38          System.out.println("hello\bworld");
39      }
40  }
```

```
 1  // Example 8.2 joi/examples/StringDemo.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  // A class illustrating Strings
 7  //
 8  // %> java StringDemo
 9  // certainly = "yes!"
10  // bankName = "Dewey, Cheatham and Howe"
11  // bankName.charAt( 0 ) = D
12  // bankName.charAt( 5 ) = ,
13  // bankName.indexOf('e') = 1
14  // bankName.indexOf('e', 6) = 9
15  // bankName.indexOf('x') = -1
16  // "cake".compareTo("care") = -7
17  // bankName.substring( 7, 12 ) = Cheat
18  // bankName.substring( 7 ) = Cheatham and Howe
19  // bankName.toUpperCase() = "DEWEY, CHEATHAM AND HOWE"
20  // bankName.replace('e', 'x') = "Dxwxy, Chxatham and Howx"
21  // bankName.concat("!") = "Dewey, Cheatham and Howe!"
22  // "  x y z \t\b".trim() = "x y z"
23  // %>
24
25  public class StringDemo
26  {
27      public static void main( String[] args )
28      {
29          Terminal t = new Terminal();
30
31          String bankName = "Dewey, Cheatham and Howe";
32          String alias = new String( bankName );
33          char[] carray = {'y', 'e', 's', '!'};
34          String certainly = new String(carray);
35
36          t.println( "certainly = \"" + certainly + "\"" );
37
38          t.println( "bankName = \"" + bankName + "\"" );
39          t.println( "bankName.charAt( 0 ) = " + bankName.charAt( 0 ) );
40          t.println( "bankName.charAt( 5 ) = " + bankName.charAt( 5 ) );
41
42          t.println("bankName.indexOf('e') = " + bankName.indexOf('e'));
43          t.println("bankName.indexOf('e', 6) = " +
44                    bankName.indexOf('e', 6));
45          t.println("bankName.indexOf('x') = " + bankName.indexOf('x'));
46
47          t.println( "\"cake\".compareTo(\"care\") = " +
48                     "cake".compareTo("care") );
49
50          t.println( "bankName.substring( 7, 12 ) = " +
51                     bankName.substring( 7, 12 ) );
52          t.println( "bankName.substring( 7 ) = " +
53                     bankName.substring( 7 ) );
54
55          t.println( "bankName.toUpperCase() = \"" +
56                     bankName.toUpperCase() + "\"" );
```



```
57          t.println( "bankName.replace('e', 'x') = \"" +
58                     bankName.replace('e', 'x') + "\"" );
59          t.println( "bankName.concat(\"!\") = \"" +
60                     bankName.concat("!") + "\"" );
61          t.println( "\"  x y z \\t\\b\".trim() = \"" +
62                     "  x y z \t\b".trim() + "\"" );
63      }
64  }
```

```
// Example 8.3 joi/examples/ReflectionDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.lang.reflect.*;

// A short program to illustrate how Java uses
// class information dynamically.
//
// This file declares class Greeting as well as class
// ReflectionDemo. Java requires that a public class be
// declared in a file that matches its name, but Greeting
// is not marked public.
//
// %> java ReflectionDemo
// Greeting@93dee9 is an instance of class Greeting
// classOfG.toString(): class Greeting
// classOfG.getName():  Greeting
// fields in class Greeting (not inherited):
// name: message, type: class java.lang.String
// methods in class Greeting (not inherited):
// invoking hello
// hello, world!
// Creating an object when you know the name of its class:
// g = (Greeting)Class.forName("Greeting").newInstance();
// g.toString(): Greeting@6f0472
// Try to create an instance of nonexistent class Foo:
// java.lang.ClassNotFoundException: Foo

public class ReflectionDemo
{
    public static void main( String[] args )
    {
        Greeting g = new Greeting();
        Class classOfG = g.getClass();
        out(g.toString() + " is an instance of " +
                           classOfG.toString());
        out("classOfG.toString(): " + classOfG.toString());
        out("classOfG.getName():  " + classOfG.getName());

        out("fields in class Greeting (not inherited):");

        Field[] greetingFields = classOfG.getFields();
        for (int i=0; i < greetingFields.length; i++) {
            Field f = greetingFields[i];
            if (f.getDeclaringClass() == classOfG) {
                out("name: " + f.getName() + ", type: " + f.getType());
            }
        }

        out("methods in class Greeting (not inherited):");

        Method[] greetingMethods = classOfG.getMethods();
        for (int i=0; i < greetingMethods.length; i++) {
            Method m = greetingMethods[i];
```



```
            if (m.getDeclaringClass() == classOfG) {
                out("invoking " + m.getName());
                try {
                    m.invoke(g, null);
                }
                catch( Exception e) {
                    out(e.toString());
                }
            }
        }

        out("Creating an object when you know the name of its class:");
        out("g = (Greeting)Class.forName(\"Greeting\").newInstance();");
        try {
            g = (Greeting)Class.forName("Greeting").newInstance();
            out( "g.toString(): " + g.toString());
        }
        catch (Exception e) { // couldn't find class
            out(e.toString());
        }

        out("Try to create an instance of nonexistent class Foo:");
        Object o;
        try {
            o = Class.forName("Foo").newInstance();
        }
        catch (Exception e) { // couldn't find class
            out(e.toString());
        }
    }

    // too lazy to type "System.out.println()
    public static void out( String s )
    {
        System.out.println(s);
    }
}

class Greeting
{
    public String message = "hello, world";

    public void hello()
    {
        System.out.println(message + "!");
    }
}
```

```
// Example 9.1 SerializationDemo.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

import java.io.*;
import java.util.*;

// Test of Java serialization.
//
// %> java SerializationDemo
// Wrote: round blue Mon Jan 06 20:14:44 EST 2003
// Read:  round null Mon Jan 06 20:14:44 EST 2003
//
// interesting observations:
//
// %> wc -c SerializationDemo* tmp
//    1207 SerializationDemo$Circle.class
//    1611 SerializationDemo.class
//    3221 SerializationDemo.java
//     271 tmp
//
// the "strings" command finds ascii strings in a file
//
// %> strings SerializationDemo.class | wc
//    45      45     813
//
// %> strings tmp
// SerializationDemo$Circle?
// attributest
//     Ljava/util/Map;L
// selft
//      LSerializationDemo$Circle;xpsr
// java.util.HashMap
// loadFactorI
// thresholdxp?@
// datesr
// java.util.Datehj
// thisq
// namet
// roundxq
//
// %> strings tmp | wc -c
// 180

public class SerializationDemo
{
    public static void main (String[] args)
    {
        Circle circle = new Circle("round");
        write(circle,"tmp");
        System.out.println("Wrote: "+ circle);

        Circle circleCopy = (Circle)read("tmp");
        System.out.println("Read:  " + circleCopy);
    }
```



```

    public static void write (Object obj, String pathname)
    {
        try {
            FileOutputStream f = new FileOutputStream(pathname);
            ObjectOutput s = new ObjectOutputStream(f);
            s.writeObject(obj);
            s.flush();  s.close();  f.close();
        }  catch (Exception e) { e.printStackTrace(); }
    }

    public static Object read(String pathname)
    {
        try {
            Object obj;
            FileInputStream in = new FileInputStream(pathname);
            ObjectInputStream s = new ObjectInputStream(in);
            obj = s.readObject();
            s.close(); in.close();
            return(obj);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return(null);
    }

    // To implement the Serializable interface you just _say_ so.
    // You don't have to _do_ anything, although you may choose to
    // overwrite the writeObject() and readObject() methods.

    private static class Circle implements Serializable
    {
        private Circle self;    // a circular reference
        private Map attributes; // saved with its contents

        // Don't bother saving whatever the current color
        //(user settable) happens to be:
        transient private String color;

        Circle(String name)
        {
            attributes = new HashMap();
            attributes.put("this", this); // a circular reference
            attributes.put("name", name);
            attributes.put("date", new Date());
            this.color = "blue";
            self = this;
        }

        public void setColor(String color)
        {
            this.color = color;
        }

        // NOTE: serialization does not call toString-- it calls
```

```
   113          // a smarter serialization method.
   114          public String toString()
   115          {
   116              return( (String)attributes.get("name") + " " +
   117                      color + " " + (Date)attributes.get("date") );
   118          }
   119      }
   120  }
```