

```

1 // Example 2.1 WhileDemo.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 WhileDemo
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 WhileDemo
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 + ":\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     {
17         Terminal terminal = new Terminal(); // for input and output
18     }
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 foi/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 StaticDemos 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         }
86         terminal.println();
87         terminal.println( "last classMethod() = " + classMethod() );
88     }

```

```

1 // Example 3.2 foi/examples/FordDemo.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 FordDemo
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 FordDemo
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 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 foi/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 }

```



```

1 // Example 4.3 ArrayListDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 // Tell the java compiler that the ArrayList class is in
7 // the java.util part of the library.
8
9 import java.util.ArrayList;
10
11 // Exercise the most important parts of the ArrayList API.
12 //
13 // %> java ArrayListDemo
14 // Create a list containing three SimpleObjects.
15 // 0 zero
16 // 1 one
17 // 2 two
18 // Replace the object at position 0.
19 // Put a new object at 2 and push the rest along.
20 // Print out the list again.
21 // 0 new zero
22 // 1 one
23 // 2 one point five
24 // 3 two
25
26 public class ArrayListDemo
27 {
28     public static void main( String[] args )
29     {
30         System.out.println("Create a list containing three SimpleObjects.
31
32         // Create a new, empty ArrayList
33         // with the ArrayList constructor.
34         ArrayList myList = new ArrayList();
35
36         // Put three things on it with the add()
37         // method - each add appends to the list.
38         myList.add(new SimpleObject("zero"));
39         myList.add(new SimpleObject("one"));
40         myList.add(new SimpleObject("two"));
41
42         // Print the list with a for loop.
43         // size() method tells how long the list is.
44         // get(int index) method retrieves value stored at position index
45         // The (SimpleObject) cast tells Java what type of thing you got
46         for (int i = 0; i < myList.size(); i++) {
47             SimpleObject foo = (SimpleObject)myList.get(i);
48             System.out.println(i + "\t" + foo.name);
49         }
50
51         // set(int index) method changes value stored at position index
52         System.out.println("Replace the object at position 0.");
53         myList.set(0, new SimpleObject("new zero"));
54
55         System.out.println("Put a new object at 2 and push the rest along
56         myList.add(2, new SimpleObject("one point five"));

```

```

57
58     System.out.println("Print out the list again.");
59     for (int i = 0; i < myList.size(); i++) {
60         SimpleObject foo = (SimpleObject)myList.get(i); // note cast!
61         System.out.println(i + "\t" + foo.name);
62     }
63
64
65     // This really simple class exists only to provide
66     // things to put in the ArrayList.
67
68     // It's an inner class, declared inside the ArrayListDemo
69     // class, which is its scope.
70
71     // Since it's visible only here, we are using a public
72     // name field rather than a private field and a public
73     // getName()
74
75     private static class SimpleObject {
76
77         public String name;
78
79         public SimpleObject( String name ) {
80             this.name = name;
81         }
82     } // end of body of inner class SimpleObject
83
84 } // end of body of ArrayList Demo

```

```

1 // Example 4.4 joi/examples/TreeMapDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.util.TreeMap;
7 import java.util.Iterator;
8 import java.util.Set;
9 import java.util.Collection;
10 import java.util.Map;
11
12 // A class illustrating the use of TreeMap. A typical run:
13 //
14 // %> java TreeMapDemo
15 // Store 3 wrapped ints, keys "one", "two", "three".
16 // The wrapped int stored for "two" is 2
17 //
18 // Iterate over keys, get each value.
19 // Note that key order is alphabetical:
20 // The value for key one is 1
21 // The value for key three is 3
22 // The value for key two is 2
23 //
24 // Iterate over the values:
25 // 1
26 // 3
27 // 2
28 //
29 // Iterate over the key-value pairs:
30 // The value for the entry with key one is 1
31 // The value for the entry with key three is 3
32 // The value for the entry with key two is 2
33 //
34 // How a TreeMap represents itself as a String:
35 // {one=1, three=3, two=2}
36 //
37 // Store a different value at key "two"
38 // {one=1, three=3, two=2222}
39 //
40 // Store map.get("one") at key "two"
41 // {one=1, three=3, two=1}
42 //
43 // A TreeMap with Integer keys mapping to String values
44 // {1=I, 2=II, 3=III}
45 // %>
46
47 public class TreeMapDemo
48 {
49     public static void main( String[] args )
50     {
51         Terminal terminal = new Terminal(); // for input and output
52
53         TreeMap map = new TreeMap();
54
55         // Put in some ints (each wrapped up as an Integer object)
56         terminal.println(

```

```

57         "Store 3 wrapped ints, keys \"one\", \"two\", \"three\".");
58         map.put("one", new Integer(1) );
59         map.put("two", new Integer(2) );
60         map.put("three", new Integer(3) );
61
62         // get the value associated with a key;
63         // notice the required cast.
64         Integer wrappedInt = (Integer) map.get( "two" );
65
66         // And print the wrapped int
67         terminal.println( "The wrapped int stored for \"two\" is "
68             + wrappedInt );
69
70
71         // The set of keys.
72         Set keys = map.keySet();
73         // The iterator over this "set" of keys will return
74         // the keys in key-order.
75         terminal.println( "\nIterate over keys, get each value." );
76         terminal.println( "Note that key order is alphabetical:" );
77         Iterator keyIterator = keys.iterator();
78         while ( keyIterator.hasNext() ) {
79             String key = (String) keyIterator.next();
80             terminal.println( "The value for key " + key + " is "
81                 + ((Integer) map.get( key )) );
82         }
83
84         // Iterate over the collection of values;
85         // notice the order is the same (ie the key-order).
86         terminal.println( "\nIterate over the values:" );
87         Iterator valuesIterator = map.values().iterator();
88         while ( valuesIterator.hasNext() ) {
89             terminal.println( ((Integer) valuesIterator.next());
90         }
91
92         // The set of Map.Entry objects (key-value pairs);
93         // Map.Entry is an inner class of Map.
94
95         // Iterate over the entries.
96         terminal.println( "\nIterate over the key-value pairs:" );
97         Iterator entriesIterator = map.entrySet().iterator();
98         while ( entriesIterator.hasNext() ) {
99             Map.Entry entry = (Map.Entry) entriesIterator.next();
100             terminal.println( "The value for the entry with key "
101                 + entry.getKey() + " is "
102                 + ((Integer) entry.getValue());
103         }
104
105         // how a TreeMap represents itself as a String:
106         terminal.println(
107             "\nHow a TreeMap represents itself as a String:" );
108         terminal.println( map.toString() );
109         terminal.println();
110
111         // We can overwrite the value stored under a key
112         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
135 }
136 }
```

```

1 // Example 5.1 foi/examples/SwitchDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 // A class illustrating the Switch statement
7
8 // %> java SwitchDemo
9 // Enter an integer: 2
10 // two
11 // Notice the importance of the breaks!
12 // The same statement without the breaks:
13 // two
14 // three
15 // Not one, two or three!
16 // Enter a character: y
17 // yes
18 // %>
19
20 public class SwitchDemo
21 {
22     public static void main( String[] args )
23     {
24         Terminal terminal = new Terminal();
25
26         int i = terminal.readInt( "Enter an integer: " );
27
28         switch ( i ) {
29             case 1:
30                 terminal.println( "one" );
31                 break;
32             case 2:
33                 terminal.println( "two" );
34                 break;
35             case 3:
36                 terminal.println( "three" );
37                 break;
38             default:
39                 terminal.println( "Not one, two or three!" );
40         }
41
42         terminal.println( "Notice the importance of the breaks!" );
43         terminal.println( "The same statement without the breaks:" );
44
45         switch ( i ) {
46             case 1:
47                 terminal.println( "one" );
48             case 2:
49                 terminal.println( "two" );
50             case 3:
51                 terminal.println( "three" );
52             default:
53                 terminal.println( "Not one, two or three!" );
54         }
55
56         switch ( terminal.readChar( "Enter a character: " ) ) {

```

```

57         case 'y':
58             terminal.println( "yes" );
59             break;
60         case 'n':
61             terminal.println( "no" );
62             break;
63         default:
64             terminal.println( "Neither yes nor no." );
65     }
66 }
67 }

```

```

1 // Example 5.2 foi/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():
14 // nobj:
15 // nobj.toStringFromObject(): NamedObject@206fddf64
16 // nobj = new NamedObject( args[1] );
17 // nobj.toString():
18 // nobj:
19 // nobj.toStringFromObject(): NamedObject@2103ddf64
20 //
21 // toString():
22 // t:
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 }

```

```

1 // Example 5.3 foi/examples/EqualsDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 // A class illustrating == and equals().
7 //
8 // %> java EqualsDemo
9 // Different objects, same field:
10 // el == ellookAlIke -> false
11 // el.equals( ellookAlIke ) -> true
12 // Same object:
13 // el == elToo -> true
14 //
15 // Different ArrayLists with equal (but not ==) elements:
16 // alist0 == alist1 -> false
17 // alist0.equals(alist1) -> true
18 //
19 // Different TreeMapS with equal keys
20 // mapping to equal (but !=) values:
21 // tmap0 == tmap1 -> false
22 // tmap0.equals(tmap1) -> true
23 //
24 // tmap0.toString() -> {sillykey = EqualsDemo value 1}
25 // tmap1.toString() -> {sillykey = EqualsDemo value 1}
26 // Are these Strings == ? false
27 // Are these Strings equal ? true
28
29 import java.util.ArrayList;
30 import java.util.TreeMap;
31
32 public class EqualsDemo
33 {
34     // Properties of an EqualsDemo object.
35     //
36     // Override equals: two of these objects are equal if
37     // their integer field has the same (i.e. ==) value.
38     //
39     // When you override equals it's customary to override
40     // toString too, so that equal objects return the equal
41     // Strings, so we do.
42     private int field;
43
44
45     public EqualsDemo( int field )
46     {
47         this.field = field;
48     }
49
50     public boolean equals( Object other )
51     {
52         return (other instanceof EqualsDemo)
53             && (this.field == ((EqualsDemo)other).field);
54     }
55
56     public String toString()

```

```

57     {
58         return " EqualsDemo value " + field;
59     }
60
61     public static void main( String[] args )
62     {
63         Terminal t = new Terminal();
64
65         // EqualsDemo object == vs equals()
66         EqualsDemo el = new EqualsDemo( 1 );
67         EqualsDemo ellookAlIke = new EqualsDemo( 1 ); // same field.
68         EqualsDemo elToo = el; // same object
69
70         t.println("Different objects, same field:");
71         t.println("el == ellookAlIke -> " + (el == ellookAlIke));
72         t.println("el.equals( ellookAlIke ) -> " + el.equals( ellookAlIke ));
73         t.println("Same object:");
74         t.println("el == elToo -> " + (el == elToo));
75         t.println();
76
77         // Arrays and Maps
78         ArrayList alist0 = new ArrayList();
79         ArrayList alist1 = new ArrayList();
80
81         alist0.add( el );
82         alist1.add( ellookAlIke );
83
84         t.println( "Different ArrayLists with equal (but not ==) elements
85         t.println( "alist0 == alist1 -> " + (alist0 == alist1));
86         t.println( "alist0.equals(alist1) -> " + alist0.equals(alist1));
87         t.println();
88
89         TreeMap tmap0 = new TreeMap();
90         TreeMap tmap1 = new TreeMap();
91
92         tmap0.put( "sillykey ", el );
93         tmap1.put( "sillykey ", ellookAlIke );
94
95         t.println( "Different TreeMapS with equal keys" );
96         t.println( "mapping to equal (but !=) values:" );
97
98         t.println( "tmap0 == tmap1 -> " + (tmap0 == tmap1));
99         t.println( "tmap0.equals(tmap1) -> " + tmap0.equals(tmap1));
100         t.println();
101
102         // Test Strings for == and equal
103         String s0 = tmap0.toString();
104         String s1 = tmap1.toString();
105         t.println( "tmap0.toString() -> " + s0);
106         t.println( "tmap1.toString() -> " + s1);
107         t.println( "Are these Strings == ? " + (s0 == s1) );
108         t.println( "Are these Strings equal ? " + (s0.equals(s1)) );
109
110     }

```

```

1 // Example 7.1 foi/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             "\nhook 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 foi/examples/EscapedDemo.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 EscapedDemo
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 EscapedDemo
20 {
21     public static void main( String[] args )
22     {
23         System.out.println("argument to println\output");
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\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 foi/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("i") = "Dewey, Cheatham and Howei"
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', 'i'};
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(\"i\") = \" + " +
60             bankName.concat("i") + "\"" );
61         t.println( " \" + " +
62             " x y z \t\b\".trim() = \" + " +
63             " x y z \t\b\".trim() + "\"" );
64     }

```

```

1 // Example 8.3 foi/examples/ReflectionDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5 import java.lang.reflect.*;
6
7
8 // A short program to illustrate how Java uses
9 // class information dynamically.
10 //
11 // This file declares class Greeting as well as class
12 // ReflectionDemo. Java requires that a public class be
13 // declared in a file that matches its name, but Greeting
14 // is not marked public.
15 //
16 // %> java ReflectionDemo
17 // Greeting@93dee9 is an instance of class Greeting
18 // classOfG.toString(): class Greeting
19 // classOfG.getName(): Greeting
20 // fields in class Greeting (not inherited):
21 // name: message, type: class java.lang.String
22 // methods in class Greeting (not inherited):
23 // invoking hello
24 // hello, world!
25 // Creating an object when you know the name of its class:
26 // g = (Greeting)Class.forName("Greeting").newInstance();
27 // g.toString(): Greeting@6f0472
28 // Try to create an instance of nonexistent class Foo:
29 // java.lang.ClassNotFoundException: Foo
30
31 public class ReflectionDemo
32 {
33     public static void main( String[] args )
34     {
35         Greeting g = new Greeting();
36         Class classOfG = g.getClass();
37         out(g.toString() + " is an instance of " +
38             classOfG.toString());
39         out("classOfG.toString(): " + classOfG.toString());
40         out("classOfG.getName(): " + classOfG.getName());
41
42         out("fields in class Greeting (not inherited):");
43
44         Field[] greetingFields = classOfG.getFields();
45         for (int i=0; i < greetingFields.length; i++) {
46             Field f = greetingFields[i];
47             if (f.getDeclaringClass() == classOfG) {
48                 out("name: " + f.getName() + ", type: " + f.getType());
49             }
50
51             out("methods in class Greeting (not inherited):");
52
53             Method[] greetingMethods = classOfG.getMethods();
54             for (int i=0; i < greetingMethods.length; i++) {
55                 Method m = greetingMethods[i];
56

```

```

57         if (m.getDeclaringClass() == classOfG) {
58             out("invoking " + m.getName());
59             try {
60                 m.invoke(g, null);
61             }
62             catch( Exception e) {
63                 out(e.toString());
64             }
65         }
66     }
67
68     out("Creating an object when you know the name of its class:");
69     out("g = (Greeting)Class.forName(\"Greeting\").newInstance()");
70     try {
71         g = (Greeting)Class.forName("Greeting").newInstance();
72         out("g.toString(): " + g.toString());
73     }
74     catch (Exception e) { // couldn't find class
75         out(e.toString());
76     }
77
78     out("Try to create an instance of nonexistent class Foo:");
79     Object o;
80     try {
81         o = Class.forName("Foo").newInstance();
82     }
83     catch (Exception e) { // couldn't find class
84         out(e.toString());
85     }
86
87     // too lazy to type "System.out.println()
88     // public static void out( String s )
89     {
90         System.out.println(s);
91     }
92
93     }
94
95     class Greeting
96     {
97         public String message = "hello, world";
98
99         public void hello()
100         {
101             System.out.println(message + "!");
102         }
103     }

```

```

1 // Example 9.1 SerializationDemo.java
2 //
3 //
4 // Copyright 2003 Bill Campbell and Ethan Bolker
5
6 import java.io.*;
7 import java.util.*;
8
9 // Test of Java serialization.
10 //
11 // %> java SerializationDemo
12 // Wrote: round blue Mon Jan 06 20:14:44 EST 2003
13 // Read: round null Mon Jan 06 20:14:44 EST 2003
14 //
15 // interesting observations:
16 //
17 // %> wc -c SerializationDemo* tmp
18 // 1207 SerializationDemo$Circle.class
19 // 1611 SerializationDemo.class
20 // 3221 SerializationDemo.java
21 // 271 tmp
22 //
23 // the "strings" command finds ascii strings in a file
24 //
25 // %> strings SerializationDemo.class | wc
26 // 45 45 813
27 //
28 // %> strings tmp
29 // SerializationDemo$Circle?
30 // attribute?
31 // Ljava/util/Map;L
32 // self?
33 // LSerializationDemo$Circle;xpar
34 // java.util.HashMap
35 // loadFactor?
36 // thresholdxp?@
37 // datestr
38 // java.util.Datehj
39 // thisq
40 // namet
41 // roundxq
42 //
43 // %> strings tmp | wc -c
44 // 180
45
46 public class SerializationDemo
47 {
48     public static void main (String[] args)
49     {
50         Circle circle = new Circle("round");
51         write(circle,"tmp");
52         System.out.println("Wrote: " + circle);
53
54         Circle circleCopy = (Circle)read("tmp");
55         System.out.println("Read: " + circleCopy);
56     }

```

```

57
58     public static void write (Object obj, String pathname)
59     {
60         try {
61             FileOutputStream f = new FileOutputStream(pathname);
62             ObjectOutputStream s = new ObjectOutputStream(f);
63             s.writeObject(obj);
64             s.flush(); s.close();
65             catch (Exception e) { e.printStackTrace(); }
66         }
67
68         public static Object read(String pathname)
69         {
70             try {
71                 Object obj;
72                 FileInputStream in = new FileInputStream(pathname);
73                 ObjectInputStream s = new ObjectInputStream(in);
74                 obj = s.readObject();
75                 s.close(); in.close();
76                 return(obj);
77             }
78             catch (Exception e) {
79                 e.printStackTrace();
80             }
81             return(null);
82         }
83
84         // To implement the Serializable interface you just say so.
85         // You don't have to do anything, although you may choose to
86         // overwrite the writeObject() and readObject() methods.
87
88         private static class Circle implements Serializable
89         {
90             private Circle self; // a circular reference
91             private Map attributes; // saved with its contents
92
93             // Don't bother saving whatever the current color
94             //(user settable) happens to be:
95             transient private String color;
96
97             Circle(String name)
98             {
99                 attributes = new HashMap();
100                 attributes.put("this", this); // a circular reference
101                 attributes.put("name", name);
102                 attributes.put("date", new Date());
103                 this.color = "blue";
104                 self = this;
105             }
106
107             public void setColor(String color)
108             {
109                 this.color = color;
110             }
111
112             // NOTE: serialization does not call toString-- it calls

```

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