7.14 Using Command-Line Arguments

It’s possible to pass arguments from the command line to an application via method `main`’s `String[]` parameter, which receives an array of `String`s. By convention, this parameter is named `args`. When an application is executed using the `java` command, Java passes the **command-line arguments** that appear after the class name in the `java` command to the application’s `main` method as `String`s in the array `args`. The number of command-line arguments is obtained by accessing the array’s `length` attribute. Common uses of command-line arguments include passing options and filenames to applications.

Our next example uses command-line arguments to determine the size of an array, the value of its first element and the increment used to calculate the values of the array’s remaining elements. The command

```
java InitArray 5 0 4
```

passes three arguments, `5`, `0` and `4`, to the application `InitArray`. Command-line arguments are separated by white space, *not* commas. When this command executes, `InitArray`’s `main` method receives the three-element array

`args` (i.e., `args.length` is 3) in which `args[0]` contains the `String "5"`, `args[1]` contains the `String "0"` and `args[2]` contains the `String "4"`. The program determines how to use these arguments—in Fig. 7.21 we convert the three command-line arguments to `int` values and use them to initialize an array. When the program executes, if `args.length` is not 3, the program prints an error message and terminates (lines 7–11). Otherwise, lines 12–32 initialize and display the array based on the values of the command-line arguments.

```
1   // Fig. 7.21: InitArray.java
2   // Initializing an array using command-line argu
3
4   public class InitArray {
5      public static void main(String[] args) {
6         // check number of command-line arguments
7         if (args.length != 3) {
8            System.out.printf(
9               "Error: Please re-enter the entire c
10              "an array size, initial value and in
11           }
12        else {
13           // get array size from first command-li
14           int arrayLength = Integer.parseInt(args
15           int[] array = new int[arrayLength];
16
17           // get initial value and increment from
18           int initialValue = Integer.parseInt(arg
19           int increment = Integer.parseInt(args[2
20
21           // calculate value for each array eleme
22           for (int counter = 0; counter < array.l
23              array[counter] = initialValue + incr
24           }
25
```

```
26            System.out.printf("%s%8s%n", "Index", "
                        27
  28            // display array index and value
29            for (int counter = 0; counter < array.l
30               System.out.printf("%5d%8d%n", counte
                 31            }
                   32         }
                     33      }
                       34   }
```

java InitArray

Error: Please re-enter the entire command, including an array size, initial value and increment.

java InitArray 5 0 4

Index	Value
0	0
1	4
2	8
3	12
4	16

java InitArray 8 1 2

Index	Value

0	1
1	3
2	5
3	7
4	9
5	11
6	13
7	15

Fig. 7.21

Initializing an array using command-line arguments.

Line 14 gets `args[0]`—a `String` that specifies the array size—and converts it to an `int` value that the program uses to create the array in line 15. The `static` method `parseInt` of class `Integer` converts its `String` argument to an `int`.

Lines 18–19 convert the `args[1]` and `args[2]` command-line arguments to `int` values and store them in `initialValue` and `increment`, respectively. Lines 22–24 calculate the value for each array element.

The output of the first execution shows that the application received an insufficient number of command-line arguments. The second execution uses command-line arguments 5, 0 and 4 to specify the size of the array (5), the value of the first

element (0) and the increment of each value in the array (4), respectively. The corresponding output shows that these values create an array containing the integers 0, 4, 8, 12 and 16. The output from the third execution shows that the command-line arguments 8, 1 and 2 produce an array whose 8 elements are the nonnegative odd integers from 1 to 15.