

7.7 Enhanced for Statement

The **enhanced for statement** iterates through the elements of an array *without* using a counter, thus avoiding the possibility of “stepping outside” the array. We show how to use the enhanced **for** statement with the Java API’s prebuilt data structures (called collections) in [Section 7.16](#). The syntax of an enhanced **for** statement is:

```
for (parameter : arrayName) {  
    statement  
}
```

where *parameter* has a *type* and an *identifier* (e.g., `int number`), and *arrayName* is the array through which to iterate. The type of the parameter must be *consistent* with the type of the elements in the array. As the next example illustrates, the identifier represents successive element values in the array on successive iterations of the loop.

[Figure 7.12](#) uses the enhanced **for** statement (lines 10–12) to sum the integers in an array of student grades. The enhanced **for**’s parameter is of type `int`, because `array` contains `int` values—the loop selects one `int` value from the array during each iteration. The enhanced **for** statement iterates

through successive values in the array one by one. The statement's header can be read as "for each iteration, assign the next element of `array` to `int` variable `number`, then execute the following statement." Thus, for each iteration, identifier `number` represents an `int` value in `array`. Lines 10–12 are equivalent to the following counter-controlled iteration used in lines 10–12 of [Fig. 7.5](#) to total the integers in `array`, except that the counting details are hidden from you in the enhanced `for` statement:

```
for (int counter = 0; counter < array.length; counter++)
    total += array[counter];
}
```

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using the enhanced for statement to total int
3
4  public class EnhancedForTest {
5      public static void main(String[] args) {
6          int[] array = {87, 68, 94, 100, 83, 78, 85};
7          int total = 0;
8
9          // add each element's value to total
10         for (int number : array) {
11             total += number;
12         }
13
14         System.out.printf("Total of array elements: ");
15         System.out.print(total);
16     }
}
```

Total of array elements: 849



Fig. 7.12

Using the enhanced `for` statement to total integers in an array.

The enhanced `for` statement can be used *only* to obtain array elements—it *cannot* be used to *modify* elements. If your program needs to modify elements, use the traditional counter-controlled `for` statement.

The enhanced `for` statement can be used in place of the counter-controlled `for` statement whenever code looping through an array does *not* require access to the counter indicating the index of the current array element. For example, totaling the integers in an array requires access only to the element values—the index of each element is irrelevant. However, if a program must use a counter for some reason other than simply to loop through an array (e.g., to print an index number next to each array element value, as in the examples earlier in this chapter), use the counter-controlled `for` statement.



Error-Prevention Tip 7.2

The enhanced `for` statement simplifies iterating through an array. This makes the code more readable and eliminates

several error possibilities, such as improperly specifying the control variable's initial value, the loop-continuation test and the increment expression.

Java SE 8

8

The `for` statement and the enhanced `for` statement each iterate sequentially from a starting value to an ending value. In [Chapter 17, Lambdas and Streams](#), you'll learn about streams. As you'll see, streams provide an elegant, more concise and less error-prone means for iterating through collections in a manner that enables some iterations to occur in parallel with others to achieve better multi-core system performance.