7.10 Case Study: Class `GradeBook` Using an Array to Store Grades

We now present the first part of our case study on developing a `GradeBook` class that instructors can use to maintain students' grades on an exam and display a grade report that includes the grades, class average, lowest grade, highest grade and a grade-distribution bar chart. The version of class `GradeBook` presented in this section stores the grades for one exam in a one-dimensional array. In Section 7.12, we present a version of class `GradeBook` that uses a two-dimensional array to store students' grades for *several* exams.

Storing Student Grades in an Array in Class `GradeBook`

Class `GradeBook` (Fig. 7.14) uses an array of `int`s to store several students' grades on a single exam. Array `grades` is declared as an instance variable (line 6), so each `GradeBook` object maintains its *own* set of grades. The constructor (lines 9–12) has two parameters—the name of the course and an array of grades. When an application (e.g., class `GradeBookTest` in Fig. 7.15) creates a `GradeBook` object, the application passes an existing `int` array to the constructor, which assigns the array's reference to instance variable `grades` (line 11). The `grades` array's *size* is determined by the `length` instance variable of the constructor's array parameter. Thus, a `GradeBook` object can process a *variable* number of grades. The grade values in the argument could have been input from a user, read from a file on disk (as discussed in Chapter 15) or come from a variety of other sources. In class `GradeBookTest`, we initialize an array with grade values (Fig. 7.15, line 7). Once the grades are stored in *instance variable* `grades` of class `GradeBook`, all the class's methods can access the elements of `grades`.

```
1   // Fig. 7.14: GradeBook.java
2   // GradeBook class using an array to store test
3
4   public class GradeBook {
5      private String courseName; // name of course
6      private int[] grades; // array of student gra
7
8      // constructor
9      public GradeBook(String courseName, int[] gra
10        this.courseName = courseName;
11        this.grades = grades;
12     }
13
14     // method to set the course name
15     public void setCourseName(String courseName)
16        this.courseName = courseName;
17     }
18
19     // method to retrieve the course name
```

```java
   public String getCourseName() {
      return courseName;
   }

   // perform various operations on the data
   public void processGrades() {
      // output grades array
      outputGrades();

      // call method getAverage to calculate the
      System.out.printf("%nClass average is %.2f

      // call methods getMinimum and getMaximum
      System.out.printf("Lowest grade is %d%nHig
         getMinimum(), getMaximum());

      // call outputBarChart to print grade dist
      outputBarChart();
   }

   // find minimum grade
   public int getMinimum() {
      int lowGrade = grades[0]; // assume grades

      // loop through grades array
      for (int grade : grades) {
         // if grade lower than lowGrade, assign
         if (grade < lowGrade) {
            lowGrade = grade; // new lowest grad
         }
      }

      return lowGrade;
   }

   // find maximum grade
   public int getMaximum() {
      int highGrade = grades[0]; // assume grade

      // loop through grades array
      for (int grade : grades) {
         // if grade greater than highGrade, ass
         if (grade > highGrade) {
            highGrade = grade; // new highest gr
         }
      }

      return highGrade;
   }

   // determine average grade for test
   public double getAverage() {
      int total = 0;

      // sum grades for one student
      for (int grade : grades) {
         total += grade;
      }

      // return average of grades
      return (double) total / grades.length;
   }

   // output bar chart displaying grade distribu
   public void outputBarChart() {
      System.out.println("Grade distribution:");

      // stores frequency of grades in each rang
      int[] frequency = new int[11];

      // for each grade, increment the appropria
      for (int grade : grades) {
         ++frequency[grade / 10];
      }

      // for each grade frequency, print bar in
      for (int count = 0; count < frequency.len
         // output bar label ("00-09: ", …, "90
         if (count == 10) {
            System.out.printf("%5d: ", 100);
```

```
100            }
101            else {
102               System.out.printf("%02d-%02d: ", co
103            }
104
105            // print bar of asterisks
106            for (int stars = 0; stars < frequency[
107               System.out.print("*");
108            }
109
110            System.out.println();
111         }
112      }
113
114      // output the contents of the grades array
115      public void outputGrades() {
116         System.out.printf("The grades are:%n%n");
117
118         // output each student's grade
119         for (int student = 0; student < grades.le
120            System.out.printf("Student %2d: %3d%n"
121               student + 1, grades[student]);
122         }
123      }
124   }
```

Fig. 7.14

`GradeBook` class using an array to store test grades.

Method `processGrades` (lines 25–38) contains a series of method calls that output a report summarizing the grades. Line 27 calls method `outputGrades` to print the contents of the array `grades`. Lines 119–122 in method `outputGrades` output the students' grades. A counter-controlled `for` statement *must* be used in this case, because lines 120–121 use counter variable `student`'s value to output each grade next to a particular student number (see the output in Fig. 7.15). Although array indices start at 0, a professor might typically number students starting at 1. Thus, lines 120–121 output `student + 1` as the student number to produce grade labels `"Student 1: "`, `"Student 2: "`, and so on.

Method `processGrades` next calls method `getAverage` (line 30) to obtain the average of the grades in the array. Method `getAverage` (lines 71–81) uses an enhanced `for` statement to total the values in array `grades` before calculating the average. The parameter in the enhanced `for`'s header (e.g., `int grade`) indicates that for each iteration, the `int` variable `grade` takes on a value in the array `grades`. The averaging calculation in line 80 uses `grades.length` to determine the number of grades being averaged.

Lines 33–34 in method `processGrades` call methods `getMinimum` and `getMaximum` to determine the lowest and highest grades of any student on the exam, respectively. Each of these methods uses an enhanced `for` statement to loop through array `grades`. Lines 45–50 in method `getMinimum` loop through the array. Lines 47–49 compare each grade to `lowGrade`; if a grade is less than `lowGrade`, `lowGrade` is set to that grade. When line 52 executes, `lowGrade` contains the lowest grade in the array. Method `getMaximum` (lines 56–68) works similarly to method `getMinimum`.

Finally, line 37 in method `processGrades` calls `outputBarChart` to print a grade-distribution chart using a technique similar to that in Fig. 7.6. In that example, we manually calculated the number of grades in each category (i.e., 0–9, 10–19, …, 90–99 and 100) by simply looking at a set of grades. Here, lines 91–93 use a technique similar to that in Figs. 7.7–7.8 to calculate the frequency of grades in each category. Line 88 declares and creates array `frequency` of 11 `int`s to store the frequency of grades in each category. For each `grade` in array `grades`, lines 91–93 increment the appropriate `frequency` array element. To determine which one to increment, line 92 divides the current `grade` by 10 using *integer division*—e.g., if `grade` is `85`, line 92 increments `frequency[8]` to update the count of grades in the range 80–89. Lines 96–111 print the bar chart (as shown in Fig. 7.15) based on the values in array `frequency`. Lines 106–108 of Fig. 7.14 use a value in array `frequency` to determine the number of asterisks to display in each bar.

Class `GradeBookTest` That Demonstrates Class `GradeBook`

The application of Fig. 7.15 creates an object of class `GradeBook` using the `int` array `gradesArray` (declared and initialized in line 7). Lines 9–10 pass a course name and `gradesArray` to the `GradeBook` constructor. Lines 11–12 display a welcome message that includes the course name stored in the `GradeBook` object. Line 13 invokes the `GradeBook` object's `processGrades` method. The output summarizes the 10 grades in `myGradeBook`.

Software Engineering Observation 7.2

A test harness (or test application) is responsible for creating an object of the class being tested and providing it with data. This data could come from any of several sources. Test data can be placed directly into an array with an array initializer, it can come from the user at the keyboard, from a file (as you'll see in Chapter 15), from a database (as you'll see in Chapter 24) or from a network (as you'll see in online Chapter 28). After passing this data to the class's constructor to instantiate the object, the test harness should call upon the object to test its methods and manipulate its data. Gathering data in the test harness like this allows the class to be more reusable, able to manipulate data from several sources.

```
1   // Fig. 7.15: GradeBookTest.java
2   // GradeBookTest creates a GradeBook object usin
3   // then invokes method processGrades to analyze
4   public class GradeBookTest {
5      public static void main(String[] args) {
6         // array of student grades
7         int[] gradesArray = {87, 68, 94, 100, 83,
```

```java

      GradeBook myGradeBook = new GradeBook(
         "CS101 Introduction to Java Programming
      System.out.printf("Welcome to the grade bo
         myGradeBook.getCourseName());
      myGradeBook.processGrades();
   }
}
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming

The grades are:

Student  1:  87
Student  2:  68
Student  3:  94
Student  4: 100
Student  5:  83
Student  6:  78
Student  7:  85
Student  8:  91
Student  9:  76
Student 10:  87

Class average is 84.90
Lowest grade is 68
Highest grade is 100

Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
```

```
80-89: ****
90-99: **
  100: *
```

Fig. 7.15

`GradeBookTest` creates a `GradeBook` object using an array of grades, then invokes method `processGrades` to analyze them.

Java SE 8

8

In Chapter 17, Lambdas and Streams, the example of Fig. 17.9 uses stream methods `min`, `max`, `count` and `average` to process the elements of an `int` array elegantly and concisely without having to write iteration statements. In Chapter 23, Concurrency, the example of Fig. 23.30 uses stream method `summaryStatistics` to perform all of these operations in one method call.