7.12 Case Study: Class `GradeBook` Using a Two-Dimensional Array

In Section 7.10, we presented class `GradeBook` (Fig. 7.14), which used a one-dimensional array to store student grades on a single exam. In most semesters, students take several exams. Professors are likely to want to analyze grades across the entire semester, both for a single student and for the class as a whole.

Storing Student Grades in a Two-Dimensional Array in Class `GradeBook`

Figure 7.18 contains a `GradeBook` class that uses a two-dimensional array `grades` to store the grades of *several* students on *multiple* exams. Each *row* of the array represents a *single* student's grades for the entire course, and each *column* represents the grades of *all* the students who took a particular exam. Class `GradeBookTest` (Fig. 7.19) passes the array as an argument to the `GradeBook` constructor. In this example, we use a ten-by-three array for ten students' grades on three exams. Five methods perform array manipulations to process the grades. Each method is similar to its counterpart in the earlier one-dimensional array version of `GradeBook` (Fig. 7.14). Method `getMinimum` (lines 39–55 of Fig. 7.18) determines the lowest grade of any student for the semester. Method `getMaximum` (lines 58–74) determines the highest grade of any student for the semester. Method `getAverage` (lines 77–87) determines a particular student's semester average. Method `outputBarChart` (lines 90–121) outputs a grade bar chart for the entire semester's student grades. Method `outputGrades` (lines 124–148) outputs the array in a tabular format, along with each student's semester average.

```
1   // Fig. 7.18: GradeBook.java
2   // GradeBook class using a two-dimensional array
3
4   public class GradeBook {
5      private String courseName; // name of course
6      private int[][] grades; // two-dimensional ar
7
8      // two-argument constructor initializes cours
9      public GradeBook(String courseName, int[][] g
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
20     public String getCourseName() {
21        return courseName;
22     }
```

```java

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

      // call methods getMinimum and getMaximum
      System.out.printf("%n%s %d%n%s %d%n%n",
         "Lowest grade in the grade book is", ge
         "Highest grade in the grade book is", g

      // output grade distribution chart of all
      outputBarChart();
   }

   // find minimum grade
   public int getMinimum() {
      // assume first element of grades array is
      int lowGrade = grades[0][0];

      // loop through rows of grades array
      for (int[] studentGrades : grades) {
         // loop through columns of current row
         for (int grade : studentGrades) {
            // if grade less than lowGrade, assi
            if (grade < lowGrade) {
               lowGrade = grade;
            }
         }
      }

      return lowGrade;
   }

   // find maximum grade
   public int getMaximum() {
      // assume first element of grades array is
      int highGrade = grades[0][0];

      // loop through rows of grades array
      for (int[] studentGrades : grades) {
         // loop through columns of current row
         for (int grade : studentGrades) {
            // if grade greater than highGrade,
            if (grade > highGrade) {
               highGrade = grade;
            }
         }
      }

      return highGrade;
   }

   // determine average grade for particular set
   public double getAverage(int[] setOfGrades) {
      int total = 0;

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

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

   // output bar chart displaying overall grade
   public void outputBarChart() {
      System.out.println("Overall grade distribu

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

      // for each grade in GradeBook, increment
      for (int[] studentGrades : grades) {
         for (int grade : studentGrades) {
            ++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);
         }
         else {
            System.out.printf("%02d-%02d: ",
               count * 10, count * 10 + 9);
         }

         // print bar of asterisks
         for (int stars = 0; stars < frequency[
            System.out.print("*");
         }

         System.out.println();
      }
   }

   // output the contents of the grades array
   public void outputGrades() {
      System.out.printf("The grades are:%n%n");
      System.out.print("          "); // align

      // create a column heading for each of th
      for (int test = 0; test < grades[0].lengt
         System.out.printf("Test %d ", test + 1
      }

      System.out.println("Average"); // student

      // create rows/columns of text representi
      for (int student = 0; student < grades.le
         System.out.printf("Student %2d", stude

         for (int test : grades[student]) { //
            System.out.printf("%8d", test);
         }

```

```
            // call method getAverage to calculate
            // pass row of grades as the argument
            double average = getAverage(grades[stu
            System.out.printf("%9.2f%n", average);
         }
      }
   }
```

Fig. 7.18

`GradeBook` class using a two-dimensional array to store grades.

Methods `getMinimum` and `getMaximum`

Methods `getMinimum`, `getMaximum`, `outputBarChart` and `outputGrades` each loop through array `grades` by using nested `for` statements—for example, the nested enhanced `for` statement (lines 44–52) from the declaration of method `getMinimum`. The outer enhanced `for` statement iterates through the two-dimensional array `grades`, assigning successive rows to parameter `studentGrades` on successive iterations. The square brackets following the parameter name indicate that `studentGrades` refers to a one-dimensional `int` array—namely, a row in array `grades` containing one student's grades. To find the lowest overall grade, the inner `for` statement compares the elements of the current one-dimensional array `studentGrades` to variable `lowGrade`. For example, on the first iteration of the outer `for`, row 0 of `grades` is assigned to parameter `studentGrades`. The inner enhanced `for` statement then loops through `studentGrades` and compares each `grade` value with `lowGrade`. If a grade is less than `lowGrade`, `lowGrade` is set to that grade. On the second iteration of the outer enhanced `for` statement, row 1 of `grades` is assigned to `studentGrades`, and the elements of this row are compared with variable `lowGrade`. This repeats until all rows of `grades` have been traversed. When execution of the nested statement is complete, `lowGrade` contains the lowest grade in the two-dimensional array. Method `getMaximum` works similarly to method `getMinimum`.

Method `outputBarChart`

Method `outputBarChart` in Fig. 7.18 is nearly identical to the one in Fig. 7.14. However, to output the overall grade distribution for a whole semester, the method here uses nested enhanced `for` statements (lines 97–101) to create the one-dimensional array `frequency` based on all the grades in the two-dimensional array. The rest of the code in each of the two `outputBarChart` methods that displays the chart is identical.

Method `outputGrades`

Method `outputGrades` (lines 124–148) uses nested `for` statements to output values of the array `grades` and each student’s semester average. The output (Fig. 7.19) shows the result, which resembles the tabular format of a professor’s physical grade book. Lines 129–131 of of Fig. 7.18 print the column headings for each test. We use a counter-controlled `for` statement here so that we can identify each test with a number. Similarly, the `for` statement in lines 136–147 first outputs a row label using a counter variable to identify each student (line 137). Although array indices start at 0, lines 130 and 137 output `test + 1` and `student + 1`, respectively, to produce test and student numbers starting at 1 (see the output in Fig. 7.19). The inner `for` statement (lines 139–141 of Fig. 7.18) uses the outer `for` statement’s counter variable `student` to loop through a specific row of array `grades` and output each student’s test grade. An enhanced `for` statement can be nested in a counter-controlled `for` statement, and vice versa. Finally, line 145 obtains each student’s semester average by passing the current row of `grades` (i.e., `grades[student]`) to method `getAverage`.

Method `getAverage`

Method `getAverage` (lines 77–87) takes one argument—a one-dimensional array of test results for a particular student. When line 145 calls `getAverage`, the argument is `grades[student]`, which specifies that a particular row of the two-dimensional array `grades` should be passed to `getAverage`. For example, based on the array created in Fig. 7.19, the argument `grades[1]` represents the three values (a one-dimensional array of grades) stored in row `1` of the two-dimensional array `grades`. Recall that a two-dimensional array is one whose elements are one-dimensional arrays. Method `getAverage` calculates the sum of the array elements, divides the total by the number of test results and returns the floating-point result as a `double` value (line 86 of Fig. 7.18).

Class `GradeBookTest` That Demonstrates Class `GradeBook`

Figure 7.19 creates an object of class `GradeBook` using the two-dimensional array of `int`s named `gradesArray` (declared and initialized in lines 8–17). Lines 19–20 pass a course name and `gradesArray` to the `GradeBook` constructor. Lines 21–22 display a welcome message containing the course name, then line 23 invokes `myGradeBook`’s `processGrades` method to display a report summarizing the students’ grades for the semester.

```
 1  // Fig. 7.19: GradeBookTest.java
 2  // GradeBookTest creates GradeBook object using
 3  // of grades, then invokes method processGrades
```

```
public class GradeBookTest {
    // main method begins program execution
    public static void main(String[] args) {
        // two-dimensional array of student grades
        int[][] gradesArray = {{87, 96, 70},
                               {68, 87, 90},
                               {94, 100, 90},
                               {100, 81, 82},
                               {83, 65, 85},
                               {78, 87, 65},
                               {85, 75, 83},
                               {91, 94, 100},
                               {76, 72, 84},
                               {87, 93, 73}};

        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:
```

	Test 1	Test 2	Test 3	Average
Student 1	87	96	70	84.33
Student 2	68	87	90	81.67
Student 3	94	100	90	94.67
Student 4	100	81	82	87.67

Student 5	83	65	85	77.67
Student 6	78	87	65	76.67
Student 7	85	75	83	81.00
Student 8	91	94	100	95.00
Student 9	76	72	84	77.33
Student 10	87	93	73	84.33

```
Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Overall grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: ***
70-79: ******
80-89: ***********
90-99: *******
  100: ***
```

Fig. 7.19

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