```
46      }
47
48      System.out.println("Your birthday is " + day);
49    }
50  }
```

A three-dimensional array `dates` is created in Lines 8–28. This array stores five sets of numbers. Each set is a 4-by-4 two-dimensional array.

The loop starting from line 33 displays the numbers in each set and prompts the user to answer whether the birthday is in the set (lines 41–42). If the day is in the set, the first number (`dates[i][0][0]`) in the set is added to variable `day` (line 45).

8.8 Declare an array variable for a three-dimensional array, create a $4 \times 6 \times 5$ `int` array, and assign its reference to the variable.

8.9 Assume `int[][][] x = new char[12][5][2]`, how many elements are in the array? What are `x.length`, `x[2].length`, and `x[0][0].length`?

8.10 Show the output of the following code:

```
int[][][] array = {{{1, 2}, {3, 4}}, {{5, 6},{7, 8}}};
System.out.println(array[0][0][0]);
System.out.println(array[1][1][1]);
```

Chapter Summary

1. A two-dimensional array can be used to store a table.

2. A variable for two-dimensional arrays can be declared using the syntax: `elementType[][] arrayVar`.

3. A two-dimensional array can be created using the syntax: `new elementType [ROW_SIZE][COLUMN_SIZE]`.

4. Each element in a two-dimensional array is represented using the syntax: `arrayVar[rowIndex][columnIndex]`.

5. You can create and initialize a two-dimensional array using an array initializer with the syntax: `elementType[][] arrayVar = {{row values}, . . . , {row values}}`.

6. You can use arrays of arrays to form multidimensional arrays. For example, a variable for three-dimensional arrays can be declared as `elementType[][][] arrayVar`, and a three-dimensional array can be created using `new elementType[size1][size2][size3]`.

Quiz

Answer the quiz for this chapter online at www.cs.armstrong.edu/liang/intro10e/quiz.html.

Programming Exercises

MyProgrammingLab™

*8.1 (*Sum elements column by column*) Write a method that returns the sum of all the elements in a specified column in a matrix using the following header:

```
public static double sumColumn(double[][] m, int columnIndex)
```

Write a test program that reads a 3-by-4 matrix and displays the sum of each column. Here is a sample run:

```
Enter a 3-by-4 matrix row by row:
1.5 2 3 4 Enter
5.5 6 7 8 Enter
9.5 1 3 1 Enter
Sum of the elements at column 0 is 16.5
Sum of the elements at column 1 is 9.0
Sum of the elements at column 2 is 13.0
Sum of the elements at column 3 is 13.0
```

***8.2** (*Sum the major diagonal in a matrix*) Write a method that sums all the numbers in the major diagonal in an $n \times n$ matrix of `double` values using the following header:

```
public static double sumMajorDiagonal(double[][] m)
```

Write a test program that reads a 4-by-4 matrix and displays the sum of all its elements on the major diagonal. Here is a sample run:

```
Enter a 4-by-4 matrix row by row:
1 2 3 4.0 Enter
5 6.5 7 8 Enter
9 10 11 12 Enter
13 14 15 16 Enter
Sum of the elements in the major diagonal is 34.5
```

***8.3** (*Sort students on grades*) Rewrite Listing 8.2, GradeExam.java, to display the students in increasing order of the number of correct answers.

****8.4** (*Compute the weekly hours for each employee*) Suppose the weekly hours for all employees are stored in a two-dimensional array. Each row records an employee's seven-day work hours with seven columns. For example, the following array stores the work hours for eight employees. Write a program that displays employees and their total hours in decreasing order of the total hours.

	Su	M	T	W	Th	F	Sa
Employee 0	2	4	3	4	5	8	8
Employee 1	7	3	4	3	3	4	4
Employee 2	3	3	4	3	3	2	2
Employee 3	9	3	4	7	3	4	1
Employee 4	3	5	4	3	6	3	8
Employee 5	3	4	4	6	3	4	4
Employee 6	3	7	4	8	3	8	4
Employee 7	6	3	5	9	2	7	9

8.5 (*Algebra: add two matrices*) Write a method to add two matrices. The header of the method is as follows:

```
public static double[][] addMatrix(double[][] a, double[][] b)
```

In order to be added, the two matrices must have the same dimensions and the same or compatible types of elements. Let **c** be the resulting matrix. Each element c_{ij} is $a_{ij} + b_{ij}$. For example, for two 3×3 matrices **a** and **b**, **c** is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{pmatrix}$$


VideoNote
Multiply two matrices

Write a test program that prompts the user to enter two 3×3 matrices and displays their sum. Here is a sample run:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 Enter
The matrices are added as follows
 1.0 2.0 3.0      0.0 2.0 4.0      1.0 4.0 7.0
 4.0 5.0 6.0  +   1.0 4.5 2.2  =   5.0 9.5 8.2
 7.0 8.0 9.0      1.1 4.3 5.2      8.1 12.3 14.2
```

**8.6 (*Algebra: multiply two matrices*) Write a method to multiply two matrices. The header of the method is:

```
public static double[][]
    multiplyMatrix(double[][] a, double[][] b)
```

To multiply matrix **a** by matrix **b**, the number of columns in **a** must be the same as the number of rows in **b**, and the two matrices must have elements of the same or compatible types. Let **c** be the result of the multiplication. Assume the column size of matrix a is **n**. Each element c_{ij} is $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \ldots + a_{in} \times b_{nj}$. For example, for two 3×3 matrices **a** and **b**, **c** is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

where $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$.

Write a test program that prompts the user to enter two 3×3 matrices and displays their product. Here is a sample run:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 Enter
The multiplication of the matrices is
 1 2 3      0 2.0 4.0      5.3 23.9 24
 4 5 6   *  1 4.5 2.2  =   11.6 56.3 58.2
 7 8 9      1.1 4.3 5.2    17.9 88.7 92.4
```

*8.7 (*Points nearest to each other*) Listing 8.3 gives a program that finds two points in a two-dimensional space nearest to each other. Revise the program so that it finds two points in a three-dimensional space nearest to each other. Use a two-dimensional array to represent the points. Test the program using the following points:

```
double[][] points = {{-1, 0, 3}, {-1, -1, -1}, {4, 1, 1},
  {2, 0.5, 9}, {3.5, 2, -1}, {3, 1.5, 3}, {-1.5, 4, 2},
  {5.5, 4, -0.5}};
```

The formula for computing the distance between two points `(x1, y1, z1)` and `(x2, y2, z2)` is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

**8.8 (*All closest pairs of points*) Revise Listing 8.3, FindNearestPoints.java, to display all closest pairs of points with the same minimum distance. Here is a sample run:

```
Enter the number of points: 8 Enter
Enter 8 points: 0 0 1 1 -1 -1  2 2 -2 -2 -3 -3 -4 -4 5 5 Enter
The closest two points are (0.0, 0.0) and (1.0, 1.0)
The closest two points are (0.0, 0.0) and (-1.0, -1.0)
The closest two points are (1.0, 1.0) and (2.0, 2.0)
The closest two points are (-1.0, -1.0) and (-2.0, -2.0)
The closest two points are (-2.0, -2.0) and (-3.0, -3.0)
The closest two points are (-3.0, -3.0) and (-4.0, -4.0)
Their distance is 1.4142135623730951
```

***8.9 (*Game: play a tic-tac-toe game*) In a game of tic-tac-toe, two players take turns marking an available cell in a 3 × 3 grid with their respective tokens (either X or O). When one player has placed three tokens in a horizontal, vertical, or diagonal row on the grid, the game is over and that player has won. A draw (no winner) occurs when all the cells on the grid have been filled with tokens and neither player has achieved a win. Create a program for playing tic-tac-toe.

The program prompts two players to enter an X token and O token alternately. Whenever a token is entered, the program redisplays the board on the console and determines the status of the game (win, draw, or continue). Here is a sample run:

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter a row (0, 1, or 2) for player X: 1 Enter
Enter a column (0, 1, or 2) for player X: 1 Enter

-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------
Enter a row (0, 1, or 2) for player O: 1 Enter
Enter a column (0, 1, or 2) for player O: 2 Enter

-------------
|   |   |   |
-------------
|   | X | O |
-------------
|   |   |   |
-------------
```

```
Enter a row (0, 1, or 2) for player X:

 . . .

-------------
| X |   |   |
-------------
| O | X | O |
-------------
|   |   | X |
-------------
X player won
```

**8.10 (*Largest row and column*) Write a program that randomly fills in 0s and 1s into a 4-by-4 matrix, prints the matrix, and finds the first row and column with the most 1s. Here is a sample run of the program:

```
0011
0011
1101
1010
The largest row index: 2
The largest column index: 2
```

**8.11 (*Game: nine heads and tails*) Nine coins are placed in a 3-by-3 matrix with some face up and some face down. You can represent the state of the coins using a 3-by-3 matrix with values 0 (heads) and 1 (tails). Here are some examples:

```
0 0 0    1 0 1    1 1 0    1 0 1    1 0 0
0 1 0    0 0 1    1 0 0    1 1 0    1 1 1
0 0 0    1 0 0    0 0 1    1 0 0    1 1 0
```

Each state can also be represented using a binary number. For example, the preceding matrices correspond to the numbers

```
000010000 101001100 110100001 101110100 100111110
```

There are a total of 512 possibilities, so you can use decimal numbers 0, 1, 2, 3, . . . , and 511 to represent all states of the matrix. Write a program that prompts the user to enter a number between 0 and 511 and displays the corresponding matrix with the characters H and T. Here is a sample run:

```
Enter a number between 0 and 511: 7 Enter
H H H
H H H
T T T
```

The user entered 7, which corresponds to 000000111. Since 0 stands for H and 1 for T, the output is correct.

**8.12 (*Financial application: compute tax*) Rewrite Listing 3.5, ComputeTax.java, using arrays. For each filing status, there are six tax rates. Each rate is applied to a certain amount of taxable income. For example, from the taxable income of $400,000 for a single filer, $8,350 is taxed at 10%, (33,950 − 8,350) at 15%,

(82,250 − 33,950) at 25%, (171,550 − 82,550) at 28%, (372,550 − 82,250) at 33%, and (400,000 − 372,950) at 36%. The six rates are the same for all filing statuses, which can be represented in the following array:

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.35};
```

The brackets for each rate for all the filing statuses can be represented in a two-dimensional array as follows:

```
int[][] brackets = {
  {8350, 33950, 82250, 171550, 372950},  // Single filer
  {16700, 67900, 137050, 20885, 372950}, // Married jointly
                                         // -or qualifying widow(er)
  {8350, 33950, 68525, 104425, 186475},  // Married separately
  {11950, 45500, 117450, 190200, 372950} // Head of household
};
```

Suppose the taxable income is $400,000 for single filers. The tax can be computed as follows:

```
tax = brackets[0][0] * rates[0] +
  (brackets[0][1] - brackets[0][0]) * rates[1] +
  (brackets[0][2] - brackets[0][1]) * rates[2] +
  (brackets[0][3] - brackets[0][2]) * rates[3] +
  (brackets[0][4] - brackets[0][3]) * rates[4] +
  (400000 - brackets[0][4]) * rates[5]
```

***8.13** (*Locate the largest element*) Write the following method that returns the location of the largest element in a two-dimensional array.

```
public static int[] locateLargest(double[][] a)
```

The return value is a one-dimensional array that contains two elements. These two elements indicate the row and column indices of the largest element in the two-dimensional array. Write a test program that prompts the user to enter a two-dimensional array and displays the location of the largest element in the array. Here is a sample run:

```
Enter the number of rows and columns of the array: 3 4 Enter
Enter the array:
23.5 35 2 10 Enter
4.5 3 45 3.5 Enter
35 44 5.5 9.6 Enter
The location of the largest element is at (1, 2)
```

****8.14** (*Explore matrix*) Write a program that prompts the user to enter the length of a square matrix, randomly fills in 0s and 1s into the matrix, prints the matrix, and finds the rows, columns, and diagonals with all 0s or 1s. Here is a sample run of the program:

```
Enter the size for the matrix: 4 Enter
0111
0000
0100
1111
All 0s on row 1
All 1s on row 3
No same numbers on a column
No same numbers on the major diagonal
No same numbers on the sub-diagonal
```

***8.15** (*Geometry: same line?*) Programming Exercise 6.39 gives a method for testing whether three points are on the same line.

Write the following method to test whether all the points in the array `points` are on the same line.

```
public static boolean sameLine(double[][] points)
```

Write a program that prompts the user to enter five points and displays whether they are on the same line. Here are sample runs:

```
Enter five points: 3.4 2 6.5 9.5 2.3 2.3 5.5 5 -5 4 Enter
The five points are not on the same line
```

```
Enter five points: 1 1 2 2 3 3 4 4 5 5 Enter
The five points are on the same line
```

***8.16** (*Sort two-dimensional array*) Write a method to sort a two-dimensional array using the following header:

```
public static void sort(int m[][])
```

The method performs a primary sort on rows and a secondary sort on columns. For example, the following array

```
{{4, 2},{1, 7},{4, 5},{1, 2},{1, 1},{4, 1}}
```

will be sorted to

```
{{1, 1},{1, 2},{1, 7},{4, 1},{4, 2},{4, 5}}.
```

*****8.17** (*Financial tsunami*) Banks lend money to each other. In tough economic times, if a bank goes bankrupt, it may not be able to pay back the loan. A bank's total assets are its current balance plus its loans to other banks. The diagram in Figure 8.8 shows five banks. The banks' current balances are 25, 125, 175, 75, and 181 million dollars, respectively. The directed edge from node 1 to node 2 indicates that bank 1 lends 40 million dollars to bank 2.

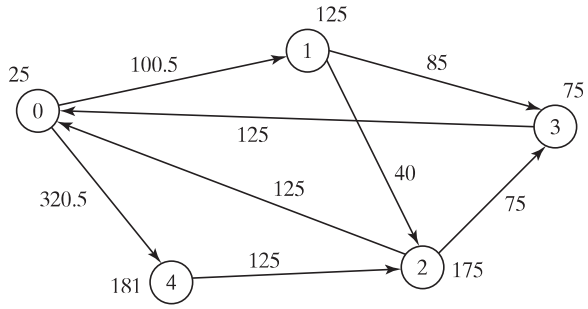


FIGURE 8.8 Banks lend money to each other.

If a bank's total assets are under a certain limit, the bank is unsafe. The money it borrowed cannot be returned to the lender, and the lender cannot count the loan in its total assets. Consequently, the lender may also be unsafe, if its total assets are under the limit. Write a program to find all the unsafe banks. Your program reads the input as follows. It first reads two integers `n` and `limit`, where `n` indicates the number of banks and `limit` is the minimum total assets for keeping a bank safe. It then reads `n` lines that describe the information for `n` banks with IDs from `0` to `n-1`.

The first number in the line is the bank's balance, the second number indicates the number of banks that borrowed money from the bank, and the rest are pairs of two numbers. Each pair describes a borrower. The first number in the pair is the borrower's ID and the second is the amount borrowed. For example, the input for the five banks in Figure 8.8 is as follows (note that the limit is 201):

```
5 201
25 2 1 100.5 4 320.5
125 2 2 40 3 85
175 2 0 125 3 75
75 1 0 125
181 1 2 125
```

The total assets of bank 3 are (75 + 125), which is under 201, so bank 3 is unsafe. After bank 3 becomes unsafe, the total assets of bank 1 fall below (125 + 40). Thus, bank 1 is also unsafe. The output of the program should be

```
Unsafe banks are 3 1
```

(*Hint*: Use a two-dimensional array `borrowers` to represent loans. `borrowers[i][j]` indicates the loan that bank `i` loans to bank `j`. Once bank `j` becomes unsafe, `borrowers[i][j]` should be set to `0`.)

***8.18** (*Shuffle rows*) Write a method that shuffles the rows in a two-dimensional `int` array using the following header:

```
public static void shuffle(int[][] m)
```

Write a test program that shuffles the following matrix:

```
int[][] m = {{1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10}};
```

****8.19** (*Pattern recognition: four consecutive equal numbers*) Write the following method that tests whether a two-dimensional array has four consecutive numbers of the same value, either horizontally, vertically, or diagonally.

```
public static boolean isConsecutiveFour(int[][] values)
```

Write a test program that prompts the user to enter the number of rows and columns of a two-dimensional array and then the values in the array and displays true if the array contains four consecutive numbers with the same value. Otherwise, display false. Here are some examples of the true cases:

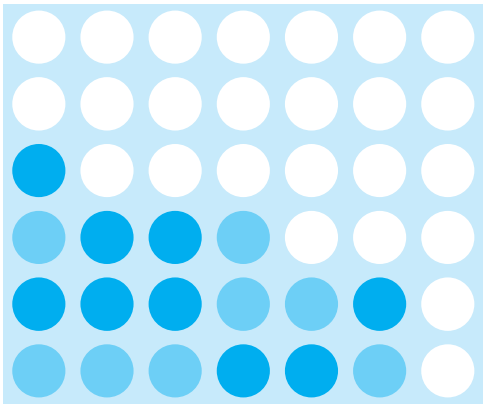
0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	8	2	9
6	5	6	1	1	9	1
1	3	6	1	4	0	7
3	3	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	5	2	1	8	2	9
6	5	6	1	1	9	1
1	5	6	1	4	0	7
3	5	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	6	2	9
6	5	6	6	1	9	1
1	3	6	1	4	0	7
3	6	3	3	4	0	7

0	1	0	3	1	6	1
0	1	6	8	6	0	1
9	6	2	1	8	2	9
6	9	6	1	1	9	1
1	3	9	1	4	0	7
3	3	3	9	4	0	7

***8.20 (*Game: connect four*) Connect four is a two-player board game in which the players alternately drop colored disks into a seven-column, six-row vertically suspended grid, as shown below.



The objective of the game is to connect four same-colored disks in a row, a column, or a diagonal before your opponent can do likewise. The program prompts two players to drop a red or yellow disk alternately. In the preceding figure, the red disk is shown in a dark color and the yellow in a light color. Whenever a disk is dropped, the program redisplays the board on the console and determines the status of the game (win, draw, or continue). Here is a sample run:

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
-----------------
Drop a red disk at column (0-6): 0 Enter

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | | | | | |
-----------------
```

```
Drop a yellow disk at column (0-6): 3 Enter

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | |Y| | | |

. . .
. . .
. . .

Drop a yellow disk at column (0-6): 6 Enter

| | | | | | | |
| | | | | | | |
| | | |R| | | |
| | | |Y|R|Y| |
| | |R|Y|Y|Y|Y|
|R|Y|R|Y|R|R|R|
---------------
The yellow player won
```

***8.21** (*Central city*) Given a set of cities, the central city is the city that has the shortest total distance to all other cities. Write a program that prompts the user to enter the number of the cities and the locations of the cities (coordinates), and finds the central city and its total distance to all other cities.

```
Enter the number of cities: 5 Enter
Enter the coordinates of the cities:
  2.5 5 5.1 3 1 9 5.4 54 5.5 2.1 Enter
The central city is at (2.5, 5.0)
The total distance to all other cities is 60.81
```

***8.22** (*Even number of 1s*) Write a program that generates a 6-by-6 two-dimensional matrix filled with 0s and 1s, displays the matrix, and checks if every row and every column have an even number of 1s.

VideoNote
Even number of 1s

***8.23** (*Game: find the flipped cell*) Suppose you are given a 6-by-6 matrix filled with 0s and 1s. All rows and all columns have an even number of 1s. Let the user flip one cell (i.e., flip from 1 to 0 or from 0 to 1) and write a program to find which cell was flipped. Your program should prompt the user to enter a 6-by-6 array with 0s and 1s and find the first row *r* and first column *c* where the even number of the 1s property is violated (i.e., the number of 1s is not even). The flipped cell is at (*r*, *c*). Here is a sample run:

```
Enter a 6-by-6 matrix row by row:
1 1 1 0 1 1 Enter
1 1 1 1 0 0 Enter
0 1 0 1 1 1 Enter
1 1 1 1 1 1 Enter
0 1 1 1 1 0 Enter
1 0 0 0 0 1 Enter
The flipped cell is at (0, 1)
```

***8.24** (*Check Sudoku solution*) Listing 8.4 checks whether a solution is valid by checking whether every number is valid in the board. Rewrite the program by checking whether every row, every column, and every small box has the numbers 1 to 9.

***8.25** (*Markov matrix*) An $n \times n$ matrix is called a *positive Markov matrix* if each element is positive and the sum of the elements in each column is 1. Write the following method to check whether a matrix is a Markov matrix.

```
public static boolean isMarkovMatrix(double[][] m)
```

Write a test program that prompts the user to enter a 3×3 matrix of double values and tests whether it is a Markov matrix. Here are sample runs:

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 Enter
0.55 0.005 0.225 Enter
0.30 0.12 0.4 Enter
It is a Markov matrix
```

```
Enter a 3-by-3 matrix row by row:
0.95 -0.875 0.375 Enter
0.65 0.005 0.225 Enter
0.30 0.22 -0.4 Enter
It is not a Markov matrix
```

***8.26** (*Row sorting*) Implement the following method to sort the rows in a two-dimensional array. A new array is returned and the original array is intact.

```
public static double[][] sortRows(double[][] m)
```

Write a test program that prompts the user to enter a 3×3 matrix of double values and displays a new row-sorted matrix. Here is a sample run:

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 Enter
0.55 0.005 0.225 Enter
0.30 0.12 0.4 Enter

The row-sorted array is
0.15 0.375 0.875
0.005 0.225 0.55
0.12 0.30 0.4
```

***8.27** (*Column sorting*) Implement the following method to sort the columns in a two-dimensional array. A new array is returned and the original array is intact.

```
public static double[][] sortColumns(double[][] m)
```

Write a test program that prompts the user to enter a 3 × 3 matrix of double values and displays a new column-sorted matrix. Here is a sample run:

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 Enter
0.55 0.005 0.225 Enter
0.30 0.12 0.4 Enter

The column-sorted array is
0.15 0.0050 0.225
0.3  0.12   0.375
0.55 0.875  0.4
```

8.28 (*Strictly identical arrays*) The two-dimensional arrays `m1` and `m2` are *strictly identical* if their corresponding elements are equal. Write a method that returns `true` if `m1` and `m2` are strictly identical, using the following header:

```
public static boolean equals(int[][] m1, int[][] m2)
```

Write a test program that prompts the user to enter two 3 × 3 arrays of integers and displays whether the two are strictly identical. Here are the sample runs.

```
Enter list1: 51 22 25 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are strictly identical
```

```
Enter list1: 51 25 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are not strictly identical
```

8.29 (*Identical arrays*) The two-dimensional arrays `m1` and `m2` are *identical* if they have the same contents. Write a method that returns `true` if `m1` and `m2` are identical, using the following header:

```
public static boolean equals(int[][] m1, int[][] m2)
```

Write a test program that prompts the user to enter two 3 × 3 arrays of integers and displays whether the two are identical. Here are the sample runs.

```
Enter list1: 51 25 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are identical
```

```
Enter list1: 51 5 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are not identical
```

***8.30** (*Algebra: solve linear equations*) Write a method that solves the following 2×2 system of linear equations:

$$a_{00}x + a_{01}y = b_0 \qquad x = \frac{b_0a_{11} - b_1a_{01}}{a_{00}a_{11} - a_{01}a_{10}} \qquad y = \frac{b_1a_{00} - b_0a_{10}}{a_{00}a_{11} - a_{01}a_{10}}$$
$$a_{10}x + a_{11}y = b_1$$

The method header is

```
public static double[] linearEquation(double[][] a, double[] b)
```

The method returns `null` if $a_{00}a_{11} - a_{01}a_{10}$ is `0`. Write a test program that prompts the user to enter a_{00}, a_{01}, a_{10}, a_{11}, b_0, and b_1, and displays the result. If $a_{00}a_{11} - a_{01}a_{10}$ is `0`, report that "The equation has no solution." A sample run is similar to Programming Exercise 3.3.

***8.31** (*Geometry: intersecting point*) Write a method that returns the intersecting point of two lines. The intersecting point of the two lines can be found by using the formula shown in Programming Exercise 3.25. Assume that (`x1`, `y1`) and (`x2`, `y2`) are the two points on line 1 and (`x3`, `y3`) and (`x4`, `y4`) are on line 2. The method header is

```
public static double[] getIntersectingPoint(double[][] points)
```

The points are stored in a 4-by-2 two-dimensional array `points` with (`points[0][0]`, `points[0][1]`) for (`x1`, `y1`). The method returns the intersecting point or `null` if the two lines are parallel. Write a program that prompts the user to enter four points and displays the intersecting point. See Programming Exercise 3.25 for a sample run.

***8.32** (*Geometry: area of a triangle*) Write a method that returns the area of a triangle using the following header:

```
public static double getTriangleArea(double[][] points)
```

The points are stored in a 3-by-2 two-dimensional array `points` with `points[0][0]` and `points[0][1]` for (`x1`, `y1`). The triangle area can be computed using the formula in Programming Exercise 2.19. The method returns `0` if the three points are on the same line. Write a program that prompts the user to enter three points of a triangle and displays the triangle's area. Here is a sample run of the program:

```
Enter x1, y1, x2, y2, x3, y3: 2.5 2 5 -1.0 4.0 2.0 Enter
The area of the triangle is 2.25
```

```
Enter x1, y1, x2, y2, x3, y3: 2 2 4.5 4.5 6 6 Enter
The three points are on the same line
```

***8.33** (*Geometry: polygon subareas*) A convex 4-vertex polygon is divided into four triangles, as shown in Figure 8.9.

Write a program that prompts the user to enter the coordinates of four vertices and displays the areas of the four triangles in increasing order. Here is a sample run:

```
Enter x1, y1, x2, y2, x3, y3, x4, y4:
  -2.5 2 4 4 3 -2 -2 -3.5 Enter
The areas are 6.17 7.96 8.08 10.42
```

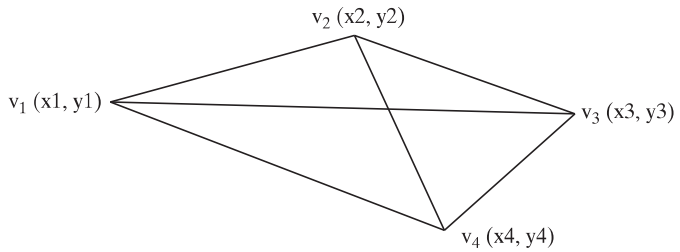


FIGURE 8.9 A 4-vertex polygon is defined by four vertices.

***8.34** (*Geometry: rightmost lowest point*) In computational geometry, often you need to find the rightmost lowest point in a set of points. Write the following method that returns the rightmost lowest point in a set of points.

```
public static double[]
       getRightmostLowestPoint(double[][] points)
```

Write a test program that prompts the user to enter the coordinates of six points and displays the rightmost lowest point. Here is a sample run:

```
Enter 6 points: 1.5 2.5 -3 4.5 5.6 -7 6.5 -7 8 1 10 2.5 Enter
The rightmost lowest point is (6.5, -7.0)
```

****8.35** (*Largest block*) Given a square matrix with the elements 0 or 1, write a program to find a maximum square submatrix whose elements are all 1s. Your program should prompt the user to enter the number of rows in the matrix. The program then displays the location of the first element in the maximum square submatrix and the number of the rows in the submatrix. Here is a sample run:

```
Enter the number of rows in the matrix: 5 Enter
Enter the matrix row by row:
1 0 1 0 1 Enter
1 1 1 0 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter

The maximum square submatrix is at (2, 2) with size 3
```

Your program should implement and use the following method to find the maximum square submatrix:

```
public static int[] findLargestBlock(int[][] m)
```

The return value is an array that consists of three values. The first two values are the row and column indices for the first element in the submatrix, and the third value is the number of the rows in the submatrix.

****8.36** (*Latin square*) A Latin square is an *n*-by-*n* array filled with `n` different Latin letters, each occurring exactly once in each row and once in each column. Write a

program that prompts the user to enter the number n and the array of characters, as shown in the sample output, and checks if the input array is a Latin square. The characters are the first n characters starting from A.

```
Enter number n: 4 Enter
Enter 4 rows of letters separated by spaces:
A B C D Enter
B A D C Enter
C D B A Enter
D C A B Enter
The input array is a Latin square
```

```
Enter number n: 3 Enter
Enter 3 rows of letters separated by spaces:
A F D Enter
Wrong input: the letters must be from A to C
```

**8.37 (*Guess the capitals*) Write a program that repeatedly prompts the user to enter a capital for a state. Upon receiving the user input, the program reports whether the answer is correct. Assume that 50 states and their capitals are stored in a two-dimensional array, as shown in Figure 8.10. The program prompts the user to answer all states' capitals and displays the total correct count. The user's answer is not case-sensitive.

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

FIGURE 8.10 A two-dimensional array stores states and their capitals.

Here is a sample run:

```
What is the capital of Alabama? Montogomery Enter
The correct answer should be Montgomery
What is the capital of Alaska? Juneau Enter
Your answer is correct
What is the capital of Arizona? ...
...
The correct count is 35
```