1.

(*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

2.

(*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
```

3.

(*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}};
```

4.

***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)
```

5.

(*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.

6.

(*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
```

7.

(*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.

(*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
```