Chapter 17 (14-17): Arrays - Exercises

Exercise 17 - 01: Analyzing scores

Write a program that reads an unspecified number of scores and determines how many scores are above or equal to the average and how many scores are below the average. Enter a negative number to signify the end of the input. Assume that the maximum number of scores is 100.

Exercise 17 - 02: Averaging an array

Write two overloaded methods that return the average of an array with the following headers:
public static int average(int[] array);
public static double average(double[] array);

Exercise 17 - 03: Finding the index of the smallest element

Write a method that returns the index of the smallest element in an array of integers. If there is more than one such element, return the smallest index. Use {1, 2, 4, 5, 10, 100, 2, –22} to test the method.

Exercise 17 - 04: Reversing an array

Write a method to reverse an array without creating new arrays. Use {1, 2, 3, 4, 5, 6} to test the method.

Exercise 17 - 05: Computing average

Write a method that returns the average of an unspecified number of numeric arguments.

Exercise 17 - 06: Timing execution

Write a program that randomly generates an array of 100000 integers and a key. Estimate the execution time of invoking the linearSearch method. Sort the array and estimate the execution time of invoking the binarySearch method. You can use the following code template to obtain the execution time:

```
long startTime = System.currentTimeMillis();
perform the task;
long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;

public class LinearSearch {
  /** The method for finding a key in the list */
  public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
```

```
        if (key == list[i])
          return i;
      return -1;
    }
}

public class BinarySearch {
  /** Use binary search to find the key in the list */
  public static int binarySearch(int[] list, int key) {
    int low = 0;
    int high = list.length - 1;

    while (high >= low) {
      int mid = (low + high) / 2;
      if (key < list[mid])
        high = mid - 1;
      else if (key == list[mid])
        return mid;
      else
        low = mid + 1;
    }

    return -low - 1;
  }
}
```

Exercise 17 - 07: Bubble sort

Write a sort method that uses the bubble-sort algorithm. The bubble-sort algorithm makes several passes through the array. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order, its values are swapped; otherwise, the values remain unchanged. The technique is called a bubble sort or sinking sort because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom.

The algorithm can be described as follows:

```
boolean changed = true;
do {
  changed = false;
  for (int j = 0; j < list.length - 1; j++)
    if (list[j] > list[j + 1]) {
      swap list[j] with list[j + 1];
      changed = true;
    }
} while (changed);
```

Clearly, the list is in increasing order when the loop terminates. It is easy to show that the do loop executes at most list.length –1 times.

Use {6.0, 4.4, 1.9, 2.9, 3.4, 2.9, 3.5} to test the method.

Exercise 17 - 08: Summing all the numbers in a matrix

Write a method that sums all the integers in a matrix of integers. Use {{1, 2, 4, 5}, {6, 7, 8, 9}, {10, 11, 12, 13}, {14, 15, 16, 17}} to test the method.

Exercise 17 - 09: Adding two matrices

Write a method to add two matrices. The header of the method is as follows:
public static int[][] addMatrix(int[][] a, int[][] b)

To test randomly generates two integer arrays.

In order to be added, the two matrices must have the same dimensions and the same or compatible types of elements. As shown below, two matrices are added by adding the two elements of the arrays with the same index:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} + b_{11}a_{12} + b_{12}a_{13} + b_{13}a_{14} + b_{14}a_{15} + b_{15} \\ a_{21} + b_{21}a_{22} + b_{22}a_{23} + b_{23}a_{24} + b_{24}a_{25} + b_{25} \\ a_{31} + b_{31}a_{32} + b_{32}a_{33} + b_{33}a_{34} + b_{34}a_{35} + b_{35} \\ a_{41} + b_{41}a_{42} + b_{42}a_{43} + b_{43}a_{44} + b_{44}a_{45} + b_{45} \\ a_{51} + b_{51}a_{52} + b_{52}a_{53} + b_{53}a_{54} + b_{54}a_{55} + b_{55} \end{pmatrix}$$