Chapter 7: Single-Dimensional Arrays

Opening Problem
Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Introducing Arrays
Array is a data structure that represents a collection of the same types of data.

Declaring Array Variables
- `datatype[] arrayRefVar;`
  - Example:
    ```java
double[] myList;
    ```
- `datatype arrayRefVar[];`  // This style is allowed, but not preferred
  - Example:
    ```java
double myList[];
    ```

Creating Arrays
`arrayRefVar = new datatype[arraySize];`

Declaring and Creating in One Step
- `datatype[] arrayRefVar = new datatype[arraySize];`
  - `double[] myList = new double[10];`
- `datatype arrayRefVar[] = new datatype[arraySize];`
  - `double myList[] = new double[10];`

Example:
- `myList = new double[10];`
  - `myList[0]` references the first element in the array
  - `myList[9]` references the last element in the array
The Length of an Array

Once an array is created, its size is fixed. It cannot be changed. You can find its size using

arrayRefVar.length

For example,

myList.length returns 10

Default Values

When an array is created, its elements are assigned the default value of

0 for the numeric primitive data types, 
"\u0000" for char types, and
false for boolean types.

Indexed Variables

The array elements are accessed through the index. The array indices are 0-based, i.e., it starts from 0 to arrayRefVar.length-1. In the example in Figure 6.1, myList holds ten double values and the indices are from 0 to 9.

Each element in the array is represented using the following syntax, known as an indexed variable:

arrayRefVar[index];

Using Indexed Variables

After an array is created, an indexed variable can be used in the same way as a regular variable. For example, the following code adds the value in myList[0] and myList[1] to myList[2].

myList[2] = myList[0] + myList[1];

Array Initializers

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand syntax must be in one statement.

Declaring, creating, initializing

Using the Shorthand Notation

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

This shorthand notation is equivalent to the following statements:

```
double[] myList = new double[4];
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```
CAUTION
Using the shorthand notation, you have to declare, create, and initialize the array all in one statement. Splitting it would cause a syntax error. For example, the following is wrong:

```java
double[] myList;  
myList = {1.9, 2.9, 3.4, 3.5};
```

Trace Program with Arrays
```
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i - 1];
            values[0] = values[1] + values[4];
        }
    }
}
```

After the array is created

After the first iteration

After i++, i becomes 2

After this line is executed, value[1] is 1

After the array is created

After the first iteration

animation

animation

animation

animation
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}

After the first iteration:

0 1 2 3 4
0 1 0 0 0

i (= 2) is less than 5

After this line is executed, values[2] is 3 (2 + 1)

After the second iteration:

0 1 2 3 4
0 1 3 0 0

After this, i becomes 3.

After this, values[3] becomes 6 (3 + 3)

After this, i becomes 4

After the third iteration:

0 1 2 3 4
0 1 3 6 0

i (=3) is still less than 5.
public class Test {
    public static void main(String[] args) {
        int[] values = new int[5];
        for (int i = 1; i < 5; i++) {
            values[i] = i + values[i-1];
        }
        values[0] = values[1] + values[4];
    }
}

After i++, i becomes 5

After this, values[4] becomes 10 (4 + 6)

After i (=4) is still less than 5

After this, values[0] becomes 11 (1 + 10)

Initializing arrays with input values

```java
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
    myList[i] = input.nextDouble();
```
Initializing arrays with random values

```java
for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random() * 100;
}
```

Printing arrays

```java
for (int i = 0; i < myList.length; i++) {
    System.out.print(myList[i] + " ");
}
```

Summing all elements

```java
double total = 0;
for (int i = 0; i < myList.length; i++) {
    total += myList[i];
}
```

Finding the largest element

```java
double max = myList[0];
for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
}
```

Shifting Elements

```java
double temp = myList[0]; // Retain the first element
// Shift elements left
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}
// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```

Enhanced for Loop (for-each loop)

JDK 1.5 introduced a new for loop that enables you to traverse the complete array sequentially without using an index variable. For example, the following code displays all elements in the array myList:

```java
for (double value : myList)
    System.out.println(value);
```

In general, the syntax is

```java
for (elementType value : arrayRefVar) {
    // Process the value
}
```

You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.
Analyze Numbers
Read one hundred numbers, compute their average, and find out how many numbers are above the average.

Problem: Deck of Cards
The problem is to write a program that picks four cards randomly from a deck of 52 cards. All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows:

```java
int[] deck = new int[52];
// Initialize cards
for (int i = 0; i < deck.length; i++)
    deck[i] = i;
```

Problem: Deck of Cards, cont.
```
Random shuffle
```
Card number 6 is the 7 (5 % 13 = 7) of Spades (7 / 13 = 0)
Card number 40 is the 10 (40 % 13 = 10) of Clubs (40 / 13 = 3)
Card number 11 is the Queen (11 % 13 = 11) of Spades (11 / 13 = 8)
Card number 24 is the Queen (24 % 13 = 11) of Hearts (24 / 13 = 1)

cardNumber / 13 =

<table>
<thead>
<tr>
<th>cardNumber % 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Problem: Deck of Cards, cont.
```
GUI Demo (picking four cards)
```

Problem: Lotto Numbers
Suppose you play the Pick-10 lotto. Each ticket has 10 unique numbers ranging from 1 to 99. You buy a lot of tickets. You like to have your tickets to cover all numbers from 1 to 99. Write a program that reads the ticket numbers from a file and checks whether all numbers are covered. Assume the last number in the file is 0.

Problem: Lotto Numbers
```
Lotto Numbers Sample Data
```

Problem: Lotto Numbers, cont.
```
LottoNumbers
```
See Exercise 20.15.

http://www.cs.armstrong.edu/liang/animation/web/24Point.html

Run 24 Point Game

http://www.cs.armstrong.edu/liang/animation/web/24Point.html

Companion Website

LottoNumbers

Run
Problem: Lotto Numbers

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) isCovered = [false, false, false, false, false]
(b) isCovered = [false, true, false, false, false]
(c) isCovered = [false, true, true, false, false]
(d) isCovered = [false, true, true, true, false]
(e) isCovered = [false, true, true, true, true]

Copying Arrays

Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```java
list2 = list1;
```

Using a loop:

```java
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];
for (int i = 0; i < sourceArrays.length; i++)
    targetArray[i] = sourceArray[i];
```

The arraycopy Utility

```java
arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);
```

Example:

```java
System.arraycopy(sourceArray, 0, targetArray, 0, sourceArray.length);
```

Searching Arrays

Searching is the process of looking for a specific element in an array; for example, discovering whether a certain score is included in a list of scores. Searching is a common task in computer programming. There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, linear search and binary search.

```java
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; 
  } 
}
```

```java
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; 
  } 
}
```
Linear Search

The linear search approach compares the key element, `key`, sequentially with each element in the array `list`. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns `-1`.

From Idea to Solution

```java
/** 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;
}
```

Trace the method

```java
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4); // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

Binary Search

For binary search to work, the elements in the array must already be ordered. Without loss of generality, assume that the array is in ascending order.

e.g., 2 4 7 10 11 14 50 59 60 66 69 70 79

The binary search first compares the key with the element in the middle of the array.

Binary Search, cont.

Consider the following three cases:

- If the key is less than the middle element, you only need to search the key in the first half of the array.
- If the key is equal to the middle element, the search ends with a match.
- If the key is greater than the middle element, you only need to search the key in the second half of the array.
Binary Search

<table>
<thead>
<tr>
<th>Key</th>
<th>List</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
<tr>
<td>B</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
<tr>
<td>B</td>
<td>1 2 3 4 6 7 8 9</td>
</tr>
</tbody>
</table>

Binary Search, cont.

The binarySearch method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns -insertion point - 1.

- The insertion point is the point at which the key would be inserted into the list.

```java
/** 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 -1 - low;
}
```
The Arrays.binarySearch Method

Since binary search is frequently used in programming, Java provides several overloaded binarySearch methods for searching a key in an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code searches the keys in an array of numbers and an array of characters.

```java
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println(java.util.Arrays.binarySearch(list, 11)); // Return is 4
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println(java.util.Arrays.binarySearch(chars, 't')) ; // Return is -4 (insertion point is 3, so return is -3+1)
```

For the binarySearch method to work, the array must be pre-sorted in increasing order.

Sorting Arrays

Sorting, like searching, is also a common task in computer programming. Many different algorithms have been developed for sorting. This section introduces a simple, intuitive sorting algorithms: selection sort.

Selection Sort

Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Animation

http://www.cs.armstrong.edu/liang/animation/web/SelectionSort.html

From Idea to Solution

```java
for (int i = 0; i < listSize; i++) {
  select the smallest element in list[i..listSize-1];
  swap the smallest with list[i], if necessary;
  // list[i] is in its correct position.
  // The next iteration apply on list[i+1..listSize-1]
}
```

```
...  
```
for (int i = 0; i < listSize; i++) {
    select the smallest element in list[i..listSize-1];
    swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration apply on list[i..listSize-1]
}

double currentMin = list[i];
int currentMinIndex = i;
for (int j = i; j < list.length; j++) {
    if (currentMin > list[j]) {
        currentMin = list[j];
        currentMinIndex = j;
    }
}

if (currentMinIndex != i) {
    list[currentMinIndex] = list[i];
    list[i] = currentMin;
}

The Arrays.sort Method

Since sorting is frequently used in programming, Java provides several overloaded sort methods for sorting an array of int, double, char, short, long, and float in the java.util.Arrays class. For example, the following code sorts an array of numbers and an array of characters.

double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

cchar[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
Java 8 now provides Arrays.parallelSort(list) that utilizes the multicore for fast sorting.

The Arrays.toString(list) Method

The Arrays.toString(list) method can be used to return a string representation for the list.