Chapter 7

Arrays and References

7.1 Passing Arrays to Methods

When you pass an array to a method, you are passing a *copy of the reference* to the array. Consider this code:

```
import java.util.Arrays;

public static void main(String[] args) {
    double [] data = {4.0, 7.0, 2.5};
    squareArray(data);
    System.out.println(Arrays.toString(data));
}

public static void squareArray(double[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = arr[i] * arr[i];
    }
}
```

Figure 7.1 is what memory looks like upon entry to `squareArray`. Java does not copy the argument array's contents into the parameter. Instead, Java copies the *reference* to the argument `data` into the parameter `arr`.

When the loop ends, the elements in the array have all changed, as show in Figure 7.2.

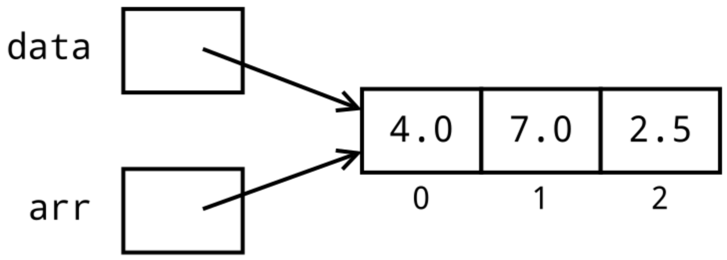

Figure 7.1: Variable `arr` is a copy of the reference variable `data`

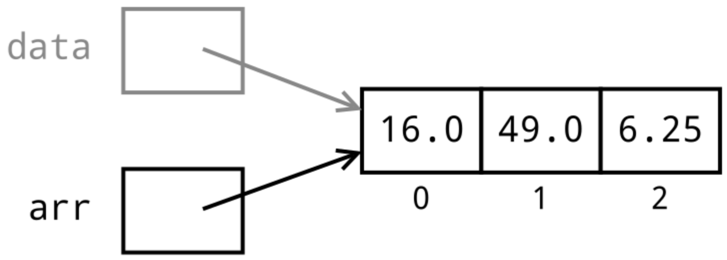

Figure 7.2: `arr` refers to updated array contents

When method `squareArray` ends, its stack frame goes away and we are left with `data` referring to the changed array, as shown in Figure 7.3.

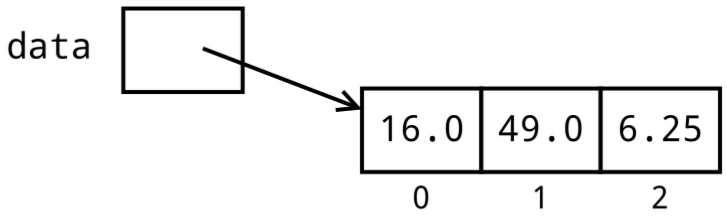

Figure 7.3: Variable `data` refers to changed array contents

When you pass an array to a method, you get a copy of the *reference*. Your methods can modify the contents of the array via the reference.

Why does Java pass a copy of the reference instead of a copy of the contents? Because back when Java was invented, CPUs were slow and memory was limited. Copying the four- or eight-byte memory address of a thousand-element array is much faster than copying all thousand elements. That's the good news.

The bad news is that you can change an array's contents inadvertently. In the preceding example, once you call `squareArray`, your original data is gone. If you call `squareArray` again, you end up with the elements of the original array to the fourth power!

7.2 Returning Arrays from Methods

We're no longer working in a "scarcity model," where every byte and microsecond is precious. Nowadays, CPUs are fast enough and memory plentiful enough that it makes sense to return a new array from a method, leaving the original untouched.

Consider this revised code, with line numbers for reference:

```
 1 import java.util.Arrays;
 2
 3 public static void main(String[] args) {
 4     double [] data = {4.0, 7.0, 2.5};
 5     double [] squaredData = newSquareArray(data);
 6     System.out.println(Arrays.toString(data));
 7     System.out.println(Arrays.toString(squaredData));
 8 }
 9
10 public static double[] newSquareArray(double[] arr) {
11     double[] result = new double[arr.length];
12
13     for (int i = 0; i < arr.length; i++) {
14         result[i] = arr[i] * arr[i];
15     }
16     return result;
17 }
```

Notice the return type in line 10: the method returns an array of `double`—the square brackets indicate an array.

When line 5 calls `newSquareArray`, line 11 creates a reference named `result` that refers to a new array with the same length as the original array. The loop

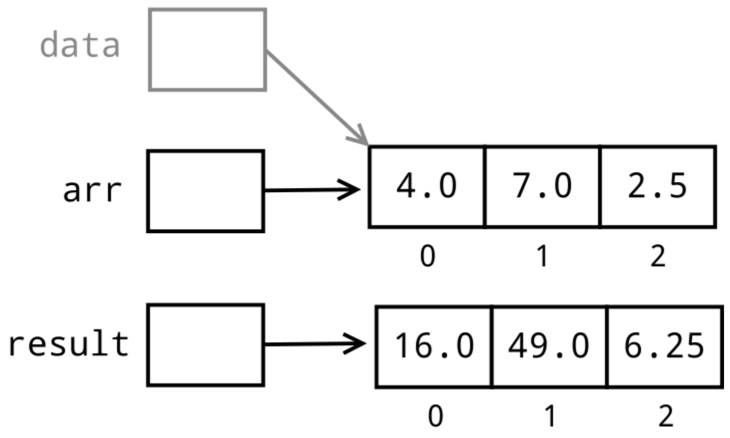

Figure 7.4: `arr` and `result` refer to different arrays

changes the value in `result`, leaving the argument `arr` unmodified. Figure 7.4 shows the memory diagram at the end of the loop.

In line 16, the method returns `result`, which, again, is a reference to the squared values. The return value is stored in `squaredData` on the left side of the assignment in line 5. Figure 7.5 shows the memory diagram after line 5.

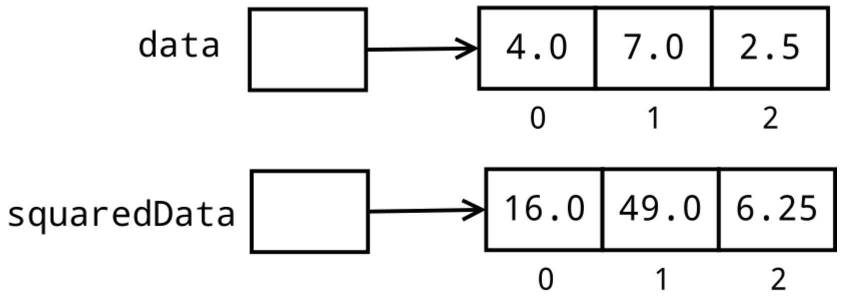

Figure 7.5: `data` and `squaredData` refer to different arrays

So, which should you do? Update arrays in place (as in the first example) or return new arrays (as in the second example)? There is a programming paradigm called *functional programming* that strongly prefers the second approach. From a functional programming standpoint, the ability to change an array in place is bad style at best and a dangerous source of programming errors at worst. In my opinion, and this is *only an opinion*, I agree with that viewpoint. As a guideline (not a rule), I suggest that you write your programs to return new arrays rather than modifying array arguments. However, if you are programming in an environment where memory is at a premium and programs run on slow processors, as can happen when you are programming embedded systems (https://en.wikipedia.org/wiki/Embedded_system), then saving space and time by modifying the array may be the better approach.

In the following exercises, you will be writing some methods that update an array in place and others that return a new array. This will let you practice both approaches.

7.3 Exercises

Exercise 7.1 Write methods named `getMax` and `getMin` that take an array of integers as their parameters and return the largest and smallest values in the array. Presume that the array has at least one element, and use a loop to traverse the elements. *Hint*: Set the return value to the first element of the array before you enter the loop. This guarantees that your return value will be an element in the array.

Write a `main` method to test `getMax` and `getMin`. For example:

```
Enter array length: 6
Enter 6 integers separated by spaces: 47 505 10 66 11 217
Minimum value: 10
Maximum value: 505
```

Exercise 7.2 The *Think Java* book has code to generate a histogram of scores from 0-100. In this exercise, you will generalize this to a method that accepts an array of integers as its parameter and displays the histogram. The method is declared as:

```java
public static void displayCounts(int[] values,
     int categories, int min, int max)
```

- `values` an array of integers with the values whose histogram you want
- `categories` how many “bars” the histogram has
- `min` the minimum value of the histogram categories

- `max` the maximum value of the histogram categories

For example, given this array and these calls:

```java
int[] data =  {35, 37, 19, 45, 49, 68, 95, 7, 5, 82, 84};
displayCounts(data, 5, 0, 100);
System.out.println("-----------");

// use getMin and getMax methods from preceding exercise
displayCounts(data, 4, getMin(data), getMax(data));
```

Generates this output:

```
0-19: 3
20-39: 2
40-59: 2
60-79: 1
80-100: 3
-----------
5-26: 3
27-48: 3
49-70: 2
71-95: 3
```

You have to do some extra work to make sure that the maximum value is included in the output of the last category range. Notice, for example, in the first histogram, all the categories have a range of 20 except the last, which has a range of 21.

Exercise 7.3 In statistics, people often want to know how many standard deviations a value is above (or below) the average. This is called a *standard score*. It's useful because it gives us a common basis for comparing the distributions of values in two sets of data that might have different means and standard deviations. In this exercise, you'll take an array of values and return a new array of standardized values.

Write the following methods, each of which takes an array of `double` values as a parameter:

- The `mean` method returns the arithmetic mean (average) of the array elements.

- The `stdev` method returns the standard deviation of the array elements, using this formula

  $$s = \sqrt{\frac{n \sum x_i^2 - \left(\sum x_i\right)^2}{n(n-1)}}$$

  where n is the number of items in the array, $\sum x_i^2$ is the sum of the squares of the individual items, and $\left(\sum x_i\right)^2$ is the square of the total of all the items in the array.

- The `standardize` returns a new array of the values converted to *standard scores.* First, calculate the mean m and standard deviation s of the entries in the original array. Each entry in the result array will be `(arr[i] - m) / s`. This method must call the `mean` and `stdev` methods.

  Write a `main` method to test the `standardize` method. For example, an array with values `{47.0, 11.0, 10.0, 66.0, 8.5}` will generate these standard scores: `{0.700, -0.662, -0.700, 1.418, -0.756}` (to three decimal places).

Exercise 7.4 Unlike the string reversal method shown in *Think Java*, where a program built a new string that had its characters in the reverse order of the original, in this exercise you will write a `void` method named `reverseInPlace` that takes an array of integers as its parameter and reverses the order of the items in the array–in place. In other words, you won't create a new array; you will change the order of the items within the array itself.

Write a `main` method that will test the method by reversing and displaying:

1. An empty array with no elements
2. An array with one element
3. An array with an even number of elements (more than two elements)
4. An array with an odd number of elements (more than three elements)

To make your job easier, you might want to create a "helper method" named `reverseAndDisplay` that prints the array, calls `reverseInPlace` and prints the array (which should now be reversed).

Exercise 7.5 You can use an array of three `double` values to represent a *vector* in three-dimensional space. The first element represents the vector's x-component, the second its y component, and the third its z component, written as (x, y, z). Write a program that has these methods, each of which takes two three-element arrays as parameters:

- `add`: Returns a vector (an array of length three) representing the sum of the parameters. The sum of (x_1, y_1, z_1) and (x_2, y_2, z_2) is the vector $(x_1 + x_2, y_1 + y_2, z_1 + z_2)$
- `dotProduct`: Returns the *dot product* of the parameter vectors. The dot product of (x_1, y_1, z_1) and (x_2, y_2, z_2) is calculated as $x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2$
- `distance`: Returns the distance between the vectors (x_1, y_1, z_1) and (x_2, y_2, z_2) using the formula $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$

The `main` method will ask the user to enter two vectors and then display the sum, dot product, and distance between the vectors. To avoid repetitious code, you might want to write a `getVector` method that has a prompt and a `Scanner` as its parameters. This method will prompt the user for the three vector components and return an array of three `double` values.

Here is an example of what the program might look like:

```
Enter components of the first vector,
separated by spaces: 2 1.5 4.1
Enter components of the second vector,
separated by spaces: 7 3.5 1.2
Sum of vectors: (9.000, 5.000, 5.300)
Distance between vectors: 6.116
Dot product of vectors: 24.170
```

Exercise 7.6 In this exercise, you will find the correlation between the elements in two arrays of equal length. Your program will ask the user for the number of entries in each array, read them in, and see how good a linear relationship they have to each other by calculating the *correlation coefficient*. This is a number that ranges from 1.0 (the two arrays are perfectly related to each other) to -1.0 (the arrays are perfectly inversely correlated to one another—as an example, think of the heights of the two ends of a seesaw in motion). A correlation of zero means the values in the two arrays have no linear relationship.

Write a method named `correlation` that takes two arrays of double and returns the correlation coefficient r according to this formula, where $\bar{x}$ stands for the mean of x and $\bar{y}$ stands for the mean of y.

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

This is the *classical formula* for correlation, and it requires you to have an array. The code will have to go through the arrays once to find the averages, and again to do the subtractions in the equation. There is a *computational formula* for correlation which can do the calculation without needing to store the numbers in an array, but that would defeat the purpose of giving you practice with arrays.

Here is an example of what the output might look like. The data are people's height in centimeters for the first array and their weight in kilograms for the second array.

```
Enter number of elements in each array: 5
Enter the 5 elements in the first array: 178 176 160 180 186
Enter the 5 elements in the second array: 84.2 77.3 60 65.5 82.1
The correlation coefficient is 0.710.
```

Exercise 7.7 Write a method called `separate` which takes an array of integers as its argument and returns a new array that consists of all odd elements in the first array followed by all the even numbers in the first array, in the same relative order.

This method will require two passes (traversals) through the array.

Then, write a `main` method to test the `separate` method. Here is what it might look like:

```
Enter number of items in array: 6
Enter the items, separated by spaces: 10 47 11 66 5 98
Array with odd, then even elements: [47, 11, 5, 10, 66, 98]
```

You may use the `java.utils.Arrays.toString` method to print the result array. Make sure you test your program with an array that contains only odd numbers and again with only even numbers.