Chapter 7 - Arrays

Chapter 7 introduces a data structure called an array that has the ability to store multiple values of a particular data type simultaneously. Other concepts that this chapter covers involve passing arrays as arguments in methods, array algorithms, String arrays, two-dimensional arrays, and applying search algorithms in arrays.

A variable can store one value of a particular data type, while an array can store multiple values of a particular data type. The number of values in an array is dependent on the size of the array, meaning that the number of values in an array cannot exceed the size of the array as initially declared. The command to declare an array is:

int[ ] numbers = new int[5];

The *int[ ]* part represents the data type that the array can hold, in this case, it holds an int, but it may also hold other data types such as double or String. The *numbers* part represents the name of the array. Similar to how variables require names, arrays also require a name. The *new int[5]* part represents the array being initialized, and the number inside the square brackets represent the size of the array that is initialized. The following diagram shows an example of how arrays are stored:

Element Number	Index 0	Index 1	Index 2	Index 3	Index 4
Array Content					

Arrays can be filled in two ways, during initialization or by accessing and manipulating the value. The following is an example of filling in an array during initialization.

int[ ] numbers = {9, 18, 27, 36, 45};

The following is an example of filling in an array by accessing and manipulating the values

numbers[0] = 9;

numbers[1] = 18;

In order to obtain a value from an array, the index of the specific value in the array is accessed. It is important to know that the index always starts from zero, meaning that if the value being accessed is the third value in the array, the index of this value will be two and not three. The command *arrayName.length* can be used to find the length of an array. With all of this knowledge beforehand, different algorithms can be applied to arrays. These include finding the maximum or minimum value in arrays, finding the sum of the array, or finding the average of the values in an array. These algorithms are usually a separate method written in the class, meaning that the array has to be passed as an argument in the method in order for the algorithm to run. ProgramOneCSeven.java demonstrates the process of applying such algorithms to an array.

The examples given above apply for one-dimensional arrays. Java also has the option for programmers to create a two-dimensional array. Most of the naming conventions from 1D arrays also apply in 2D arrays, but 2D arrays require two sizes for the array, one size for the row and one size for the column. The following diagram provides a visual aid to understand 2D arrays:

	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0					
Row 1					
Row 2					

This array is declared using the following command:

int[ ][ ] values = new int[3][5];

When declaring a 2D array, the first number represents the number of rows, and the second number represents the number of columns. ProgramTwoCSeven.java shows how a 2D array works in Java.

Other algorithms that may be applied to arrays include a sorting algorithm called selection sort, and search algorithms such as sequential search and binary search. Applying a selection sort arranges the array in ascending order, placing the smaller values in the lower elements and the higher values in the higher elements. The sort is possible because methods can also return arrays, therefore when applying the sort, the return will be the sorted array. A sequential search algorithm can locate a specific value in an array by using a loop to sequentially compare all of the values in an array from start to end to a single value. This algorithm may sometimes be simpler but will take a longer time as the number of elements in the array increases. A binary search algorithm works by starting the comparison test from the middle of the array. If the element contains the same value as the array, then the search is complete. However, if the compared value is greater than the value in the middle of the array, then the value will be located in the second half of the array. If the compared value is smaller than the value in the middle of the array, then the value will be located in the first half of the array. The search then ignores the part of the array that does not contain the specific value and starts in the middle of the section that contains the value. This cycle repeats until the value is found or until the algorithm decides that the number does not exist in the array. The only issue is that this search requires the values in the array to be sorted (possibly using selection sort) in order for the binary search algorithm to work. ProgramThreeCSeven.java shows an example of the two aforementioned different search algorithms.

When reflecting back on chapter 6's topic of classes, arrays can also be declared as an instance variable. Arrays can also be passed as arguments to methods in a different class, and methods in a different class can also return arrays.