STUDENT LESSON

Single Dimension Arrays

**INTRODUCTION:** A single dimension array is an alternative to the ArrayList studied in Lesson A15.

The key topics for this lesson are:

A. Example of an Array

B. Array Declarations and Memory Allocation

C. Applications of Arrays

D. Arrays as Parameters

E. Arrays and Algorithms

**VOCABULARY:** ALGORITHM ARRAY

**final** INDEX

RANDOM ACCESS SEQUENTIAL

TRAVERSAL

**DISCUSSION:** A. Example of an Array

1. The following program will introduce you to some of the syntax and usage of the *array* class in Java:

Code Sample 16-1:

**int**[] A = **new** **int**[6]; // an array of 6 integers

**int** loop;

**for** (loop = 0; loop < 6; loop++){

A[loop] = loop \* loop;

}

System.out.println("The contents of array A are:");

System.out.println();

**for** (loop = 0; loop < 6; loop++){

System.out.print(" " + A[loop]);

}

System.out.println();

*Run Output:*

The contents of array A are:

0 1 4 9 16 25

2. An array is similar to the ArrayList. It is a linear data structure composed of adjacent memory locations, or “cells”, each holding values of the same type.



3. The variable A is an array, a group of 6 related scalar values. There are six locations in this array referenced by indexes 0 to 5. Note that indexes always start at zero, and count up by one until the last slot of the array. If there are N slots in an array, the indexes will be 0 through N-1 (for example, if N=6, the indexes are 0 through 5 or (N-1)).

4. The variable loop is used in a **for** loop to reference indexes 0 through 5. In this program, the square of each index is stored in the memory location occupied by each cell of the array. The syntax for accessing a memory location of an array requires the use of square brackets [].

5. The square brackets [] are collectively an operator in Java, and are called the index operator. They are similar to the parentheses as they have the highest level of precedence compared to all other operators.

6. The index operator performs automatic bounds checking. Bounds checking makes sure that the index is within the range for the array being referenced. Whenever a reference to an array element is made, the index must be greater than or equal to zero and less than the size of the array. If the index is not valid, the exception ArrayIndexOutOfBoundsException is thrown.

B. Array Declarations and Memory Allocation

1. Array declarations look like this:

***type***[] arrayName;

This tells the compiler that arrayName will be used as the name of an array containing ***type***. However, the actual array is not constructed by this declaration. Often an array is declared and constructed in one statement like this:

***type***[] arrayName = **new** ***type***[length];

This tells the compiler that arrayName will be used as the name of an array containing ***type***, and constructs an array object containing length number of slots.

2. An array is an object, and like any other object in Java is constructed out of main storage as the program is running. The array constructor uses different syntax than most object constructors; ***type***[length] names the type of data in each slot and the number of slots. For example:

**int**[] list = **new** **int**[6];

**double**[] data = **new** **double**[1000];

Student[] school = **new** Student[1250];

Once an array has been constructed, the number of slots it has does not change.

3. The size of an array can be defined by using a **final** value.

**final int** MAX = 200;

**int**[] numb = **new** **int**[MAX];

4. When an array is declared, enough memory is allocated to set up the full size of the array.

C. Application of Arrays

1. Suppose we have a text file *votes.txt* of integer data containing all the votes cast in an election. This election happened to have three candidates and the values in the integer file are 1, 2, or 3, each corresponding to one of the three candidates.

Code Sample 16-2:

FileInput inFile = **new** FileInput("votes.txt");

**int** vote, total = 0, loop;

// sized to 4 boxes, initialized to 0's

**int**[] data = **new** **int**[4];

vote = inFile.readInt();

**while** (inFile.hasMoreTokens()){

data[vote]++;

total++;

vote = inFile.readInt();

}

System.out.println("Total # of votes = " + total);

**for** (loop = 1; loop <= 3; loop++){

System.out.println("Votes for #" + loop +

" = " + data[loop]);

}

a. The array data consists of four cells, each holding an integer value. The first cell, data[0], is allocated but not used in this problem. After processing the entire file, the variable data[n] contains the number of votes for candidate n. We could have stored the information for candidate 1 in position 0, candidate 2 in position 1, and so forth, but the code is easier to follow if we can use a direct correspondence.



b. The value of vote is used to increment the appropriate cell of the array by +1.

2. A second example counts the occurrence of each alphabet letter in a text file.

Code Sample 16-3:

FileInput inFile = **new** FileInput("sample.txt");

**int**[] letters = **new** int[26]; // use positions 0..25

// to count letters

**int** total = 0;

**char** ch;

**while** (inFile.hasMoreLines()){

String line = inFile.readLine().toLowerCase();

**for**(**int** index = 0; index < line.length(); index++){

ch = line.charAt(index);

// line.charAt is from chn.util. It

//extracts the entry.

**if** ('a' <= ch && ch <= 'z') { // if letter

letters[ch – ‘a’]++;

total++;

}

}

}

System.out.println("Count letters");

System.out.println();

ch = 'a';

**for** (**int** loop = 0; loop < 26; loop++){

System.out.println(ch + " : " + letters[loop]);

ch++;

}

System.out.println();

System.out.println("Total letters = " + total);

1. Each line in the text file is read in and then each character in the line is copied into ch. If ch is an uppercase letter, it is converted to its lowercase counterpart.

b. If the character is a letter, the ASCII value of the letter is adjusted to fit the range from 0-25. For example, if ch == 'b', the program calculates ‘b’ – ‘a’ = 1. Then the appropriate cell of the array is incremented by one.

D. Arrays as Parameters

|  |  |
| --- | --- |
| **See Handout 16.1, *Example Program - Arrays as Parameters* ArrayOps.java** | 1. The program *ArrayOps.java*, provides examples of passing arrays as parameters. Notice that the **final** integer constant MAX = 6 is used to size the array in this program. |

2. The main method declares an array named data. The array is initialized with the values 0...5 inside the main method.

3. The parameters of the squareList and printList methods are references to an array object. Any local reference to array list inside the squareList or printList methods is an alias for the array data inside of the main method. Notice that after the call of squareList, the values stored in array data in the main method have been permanently changed.

4. When the rotateList method is called, the copy method of the ArrayOps class is invoked and the local array listCopy is created as a copy of the array data in the main method.

5. The rotateList method rotates the values one cell to the right, with the last value moved to the front of the list. A call to printList is made inside the rotateList method just before leaving the method. After returning to the main method, notice that the array data is unchanged.

E. Arrays and Algorithms

In the following list, we introduce five important algorithms that are quite common in programs that analyze data in arrays. You will meet these again in later lessons and labs.

1. Insertion is a standard problem that must be solved for all data structures. Suppose an array had 10 values and an 11th value was to be added. We are assuming the array can store at least 11 values.

1. If we could place the new value at the end, there would be no problem.

b. But if the new value must be inserted at the beginning of the list in position 0, the other 10 values must be moved one cell over in the list.

2. Deletion of a value creates an empty cell that probably must be dealt with. The most likely solution, after deleting a value, is to move all values that are to the right of the empty spot one cell to the left.

3. A traversal of an array consists of visiting every cell location, probably in order. The visit could involve printing out the array, initializing the array, finding the largest or smallest value in the array, etc.

4. Sorting an array means to organize values in either ascending or descending order. These algorithms will be covered in depth in future lessons.

5. Searching an array means to find a specific value in the array. There are several standard algorithms for searching an array. These will be covered in future lessons.

|  |  |
| --- | --- |
| **SUMMARY/ REVIEW:** | Arrays are useful data structures and you will have many opportunities to program with them. |

**ASSIGNMENT:** Lab Assignment A16.1, *Statistics*

Lab Assignment A16.1, Data File – *numbers.txt*

Lab Assignment A16.2, *Compact*

Lab Assignment A16.2, Data File – *compact.txt*

Worksheet A16.1*, Single Dimension Arrays Review*