COP 2250 - Chapters 7 and 11 – Arrays & ArrayLists

Chapter 7 – Single Dimension Arrays

# NOTE: Chapter 8 will be covered in COP 2251

# Array Basics

* An array is a collection of variables of the same type with one name.
* Put another way, an array is a variable with multiple “slots” for values of the same type.
* The variables in an array are called the elements of the array.
* The elements are indexed numerically in sequence, starting from zero.
* The elements of an array can be of any type, including objects.
* However, in Java, all elements of a particular array must be of the same type.
* Arrays are immutable. They cannot be re-sized. Element values can be re-assigned but elements cannot be deleted, inserted, or added.
* Arrays are declared with [ ] , either before or after (more common) the array name.

double[] prices; // declares array of doubles prices

String[] friends; // declares array of Strings friends

int nums[]; // adopted to please C programmers tsk tsk

* The new keyword is used to allocate space for the array.

prices = new double [25]; // makes space for 25 doubles

nums = new int [12]; // makes space for 12 ints

You can’t use an array until you allocate space for it

* Declaration and space allocation can be combined in one step, if desired:

String friends[] = new String [30]; // C syntax

double[] monthlySales = new double [12];

* Default values are assigned to array elements when space is allocated. Numbers are assigned zero, Booleans are assigned false, objects are set to null, and chars to ‘\u0000’.
* If the values to be stored in an array are known, the array can be declared and initialized with these values when the array is created. Then, new is not needed. Note curly **braces**.

int[] carsSoldThisWeek = {2,5,0,3,2,2,1}; //**array initializer**

# Array Subscripting (indexing)

* The elements of an array are indexed, starting from zero. See Figure 7.1 on page 247.
* The seven element carsSoldThisWeek array above is indexed from 0 - 6 as shown next :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 5 | 0 | 3 | 2 | 2 | 1 |
| indexes | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

* Individual array elements are accessed by specifying the index inside [ ]

System.out.println ( carsSoldThisWeek[2] ); // output is 0

* The value stored in an array element can be reassigned in a similar manner:

monthlySales [3] = 5432.88;

* When the array elements are objects, their methods can be executed:

String[] friends = new String [30];

friends[4] = “Bud Mack”;

System.out.print(friends[4].charAt(4)); // output is M

* Every array has a length property that holds the number of elements in the array.
* The index of the last element in an array is one less than the length of the array.

# Processing Arrays

* Arrays can be easily processed with loops, either while loops or for loops.
* The key is to use the loop control variable inside the [ ] to cycle through an array.
* Note well the nine specific examples commonly required for processing arrays on p 249-251.
* Java also has an enhanced for loop (foreach loop), but it’s one-way only (ascending). See page 251. With it, you can “traverse” an array without using indexes.

int[] numbers = {12,3,34,15,9,8, 24};

for (int n : numbers) {

System.out.print(n + " ");

}

Output is 12 3 34 15 9 8 24

* Note the Caution on page 251. Attempting to access the array element at a non-existent index throws an **ArrayIndexOutOfBoundsException**, crashing the program.

Try AnalyzeNumbers

Challenge Exercise

Write a program that declares a five element array of doubles. In a loop, prompt the user to enter five test scores (percent scores) and assign them to the array. Use the length property of the array. Use a second loop to display the scores again. Use an accumulator to sum up the scores in the second loop. An accumulator is a variable that is initialized to zero and subsequently used as a “bucket” to accumulate other values. End the program by displaying the average percent score.

# Accessing Array Elements Sequentially

* there are many occasions when a Java program will need to access an array sequentially:
  + To search for a particular element.
  + To count the number of occurrences of the same value.
  + To determine the largest and/or smallest element.

# Challenge Exercise

Write a program that prompts the user to enter five integers from the keyboard. Store the integers in an array. Process the array to determine both the smallest and largest integer. Report the results.

Deck of Cards Study

* This is an interesting and instructive study on page 254.

Try DeckOfCards

Copying Arrays (See p 256)

* A simple assignment like array2 = array1; doesn’t copy array elements. It copies the reference (address) of array1 to array2, making array1 and array2 reference the same array. This happens because an array is an object type in Java.
* Use a loop or the **arraycopy( )** method of the **System** class to copy an array. See p 257.
* NOTE: that for arraycopy, the target array must have been declared and assigned space.

Passing Arrays to Methods (p 257)

* A method can receive either a named array or an anonymous array as a parameter. See p 258.
* The method receives a reference to, not a copy of, the array argument. This means that the method will have access to where the array lives in memory (and can change the array element values).
* This is called **pass-by-sharing** by Liang.

Returning an Array from a Method

* A method can return an array. The returned array must be “caught” in an array variable of the correct type or used as the argument to another method.

Try CountLettersInArray

Variable-Length Argument Lists (for a method)

* A variable number of arguments of the same type can be passed to method with the syntax **type… paramName**.
* Only **one** variable length argument is permitted, and it must be the **final** one in the list.
* Inside the method, Java creates an array for the variable length parameter.

Try VarArgsDemo

Searching Arrays

* A common programming task is to search an array for a specific value. Liang provides sample code for two algorithms for searching. See page 216.
  1. Linear Search
* Searches the array from start to finish until a **match** is found for the **key**.
* This inefficient approach can be time-consuming for a large array.
  1. Binary Search
* Assumes an **ordered** array and starts search with a **comparison** to the **middle** element.
* Each successive trial eliminates half of the remaining elements, a more efficient approach.

Sorting Arrays

* This is another common task in programs that use arrays.

Selection Sort

* To sort in ascending order, this method repeatedly selects the largest value and swaps it with the last value in the remaining values.
* See page 269 and watch the **Video Note**.

Try SelectionSort

The Arrays Class

* This utility class is in the **java.util** package and must be imported to your programs.
* The class has static methods that simplify some array operations, including sorting.
* See page 270-272.

Command Line Arguments

* It’s time to finally learn what the **String args[]** thing is all about in main.
* The **args** array is used to read strings passed into main from the command line.
* Command line args can be easily emulated in Eclipse for convenience.
* To add arguments, select the desired main class in the Package Explorer and then, on the Eclipse menu, select **Run | Run Configurations…**
* In the resulting dialog box, click the **Arguments** tab.
* Enter the desired arguments in the textbox.
* There is a short video for this, too. Watch it back in MyCourses.
* Test your Eclipse command line arguments skill with the **Calculator** program below.

Try Calculator

Class Arraylist

This important class is tested by the 1Z0-811 exam. ArrayList is covered in two locations in our text:

* Chapter 11: page 432 – 440
* Chapter 20: page 766 – 771
  + The important part of this section is the use of iterators with ArrayList.

This API class solves the issue that an array cannot be re-sized. An ArrayList is a data structure with methods for adding, deleting, inserting, and manipulating elements. See the UML diagram on p433.

* However, an ArrayList cannot store primitives like int or double. It’s for storing objects of the same type, not primitives. That doesn’t mean that you cannot store numeric data in an ArrayList. You just have to use a wrapper type like Integer or Double to store numbers.
* The object type to be stored in an ArrayList is specified inside < > when it is declared.

ArrayList<Integer> amounts = new ArrayList<>(); // array list of Integers

ArrayList<Double> prices = new ArrayList<>(); // array list of Doubles

ArrayList<String> names = new ArrayList<>(); // array list of Strings

Although an ArrayList cannot hold primitives, autoboxing and autounboxing is allowed to simplify the syntax. See pages 383 – 384.

amounts.add(new Integer(15)); // adding an element with the wrapper class

amounts.add(33); // adding an element with autoboxing

Try TestArrayList

* Table 11.1 on page 436 compares and contrasts arrays and ArrayLists. Study it well.

Try DistinctNumbers

* Like an array, an ArrayList can be processed with an enhanced for (foreach) loop. See the code snippets in the middle of page 437.

Useful Methods for Lists

* Refer to page 438-439 for some important skills for ArrayLists.

Chapter 20

* Read pages 766-771 to learn about iterators and their usage with ArrayList.

Try TestArrayAndLinkedList.java from the zip (but ignore the LinkedList code).

Try ArrayListOne from the zip.