Java Chapter 7 Part 1

* Arrays: Declaring, Processing, Using with Methods
* CIS 255 • Shelby-Hoover Campus

Arrays

* A primitive variable can only hold one value at a time
* An object variable can only hold the fields of one object at a time
* To store multiple values of the same type, a program could include several individual variables each with a different name, but this becomes difficult as the number of variables becomes larger
* An **array** is a set of variables declared with the same name and type
  + The declaration of an array includes a specific size (known as the **size declarator**)
  + A statement that uses a single variable from an array must include the position of the variable within the array

Declaring an Array

* The syntax for declaring an array involves the type, name, and size:  
    
  type[] arrayName = new type[size];  
    
  int[] numbers = new int[6];
* The size must be a non-negative int and can be a literal value, a variable, or a constant:  
    
  final int NUM\_ELEMENTS = 6;  
  int[] numbers = new int[NUM\_ELEMENTS];
* Once an array’s size is set, the size cannot be changed

Array Elements

* Each variable in an array is called an element
* To access a specific element, use its **subscript** (also called its **index**), which indicates the position of the element within the array
  + The lowest subscript in an array is 0
  + The highest subscript in an array is one less than its size
  + A ten-element array has subscripts from 0 to 9
* Write the subscript of the element to be accessed in square brackets after the array name (not the same as specifying the size in the declaration statement)
* A program uses each element in an array in the same way it uses a single variable of that type:
  + numbers[0] = 25;
  + numbers[5] = keyboard.nextInt();
  + numbers[1] = numbers[5] + 10;
  + System.out.println(numbers[1]);
* Example: ArrayDemo1.java (Code Listing 7-1)

Arrays and Loops

* The subscript used to select a particular element can be an int variable
* Many programs use a loop to apply the same processing to each element in a loop: the loop control variable (counter) is also used as the subscript
* Example: ArrayDemo2.java (Code Listing 7-2)
  + The first loop allows the user to enter a value for each employee into one of the array’s three elements (note that the label for the employee number is one greater than the subscript)
  + The second loop displays the values entered by the user one at a time

Array Bounds

* Remember: the legal subscripts for an array range from 0 to (size – 1)
* Java performs array bounds checking, testing a subscript to see if it is within the range of valid subscripts for the array, when a program executes
* Any attempt to access a subscript outside this range will result in a runtime exception (InvalidSubscript.java, Code Listing 7-3)
* Be careful not to commit an off-by-one error
  + Don’t accidentally skip the first element by using an initial subscript of 1
  + Don’t accidentally go beyond the end of the array by making the comparison <= the size

Initialization Lists

* A program may initialize an array using a list of values in curly brackets:  
    
  int[] courses = {146, 150, 251, 255};
* Java will automatically size the array to be large enough to fit the set of values
* Values will be assigned to the array elements in the order in which they are listed
* This can only be done at the time of initialization, not afterwards (assigning a new initialization list to an array variable creates a different array in memory)
* Example: ArrayInitialization.java (Code Listing 7-4)

Alternative Declaration Syntax

* The empty brackets may be placed after the array name instead of before:  
    
  int numbers[] = new int[6];
* This can be useful when declaring arrays and individual variables of the same type in a single statement:  
    
  int numbers[] = new int[6], highNum, lowNum;

Array length Member

* Java maintains the size of an array in a special member, length
* Note that this is a field, not a method (the class String has a method named length())
* A program accesses this member using the dot notation:  
    
  int[] days = new int[31];  
  int i;  
  for (i = 0; i < days.length; i++)  
   days[i] = keyboard.nextInt();

The Enhanced for Loop

* Java provides another form of the for loop, called the **enhanced for loop**, specifically for applying the same statements to the elements in an array one at a time
* Syntax:  
    
  for (type varName : arrayName)  
   statement\_to\_process\_one\_element;  
    
  for (int oneGrade : gradeArray)  
   System.out.println(oneGrade);
* Limitations
  + Array elements cannot be changed
  + Must process every element in forward order
  + Only one array at a time
  + Cannot obtain the subscript

Arrays as Objects

* An array variable stores the location of the first array element in memory
* You can assign a new array to an array variable, but the original array will be lost
* Assigning one array name to another does not copy the array elements; it causes both array variables to reference the same memory location
  + Example: SameArray.java (Code Listing 7-7)
  + To copy the contents of one array to another, use a loop to copy one element at a time:   
      
    int[] firstArray = { 5, 10, 15, 20, 25 };  
    int[] secondArray = new int[firstArray.length];  
      
    for (int index = 0; index < firstArray.length; index++)  
     secondArray[index] = firstArray[index];
* Using a comparison operator between two array names compares their memory locations, not their contents (comparing the contents is covered later in this lecture)

Array Elements as Arguments

* Each array element is an individual variable of the base type of the array
* These elements can be passed individually as arguments to a parameter of the array’s base type
  + No additional syntax is required in the method header
  + The method call must include the subscript of the element to be sent
* Primitive array elements are passed by value; object array elements are passed by reference
* Example: Using a method to display one element at a time on a single line of output (PassElements.java, Code Listing 7-8)
  + The loop calls the method showValue for each element in the array
  + The parameter n receives a copy of each element
  + Note the use of print rather than println in the method to ensure that the values appear on the same line separated by spaces

Entire Arrays as Arguments

* To send an entire array to a method, the method’s parameter must be an array parameter (with empty square brackets after the array type)
* The argument in the method call is the array name by itself (no subscript or brackets)
* Since an array is an object (the variable stores the address), array arguments are pass by reference (changes to the array parameter in the called method will also be applied to the original array argument in the calling method)
* Example: Using methods to obtain and display values in an entire array (PassArray.java, Code Listing 7-9)
  + Both methods, getValues and showArray, contain an array parameter named array (this name can vary)
  + Each method loops through the entire array
  + When main calls each method (lines 18 and 24), the argument is the array name numbers by itself (no brackets)

Comparing Arrays

* To compare two arrays’ contents (not the addresses where they are stored) requires a complex algorithm
* The code in CompareArrays.java (from section 7.4) attempts to compare two arrays of type int
  + A boolean flag variable is set to true to indicate the code has not found any evidence to indicate that the two arrays are not equal
  + The code examine the lengths of the arrays: if they differ, the arrays are not the same, so the code sets the flag to false
  + Then, the code uses a loop to examine each element in the array
    - The loop stops once the flag variable changes to false or the counter reaches the end of the array
    - If the array lengths differ, the flag variable is already false, so the code skips the loop
    - If the code finds two elements that don’t match, it changes the flag to false
* At the end of the program, if the flag variable is still true, the array contents are the same; if not, the array contents differ in some way

Array Statistics

* To get the sum of the elements in the array, set the sum variable equal to 0, then loop through the array elements and add each element’s value to the sum variable
* To find the average, divide the sum by the size of the array
* To find the maximum or minimum value:
  + Assume that the first element in the array (at subscript 0) could be the maximum / minimum
  + Loop through the remaining elements in the array, comparing each one to the maximum / minimum found so far
  + If the current element is higher / lower than the maximum / minimum found so far, save it as the new maximum / minimum
* Array processing can be packaged in a class
  + SalesData.java (Code Listing 7-10), instantiated in Sales.java (Code Listing 7-11)
  + Grader.java (Code Listing 7-12), instantiated in CalcAverage.java (Code Listing 7-13)

Partially-Filled Arrays

* It may not be possible to know how many values are coming in from input ahead of time
* The array size needs to be big enough to store the largest amount of data the program expects
* When looping to read values from input into an array, the program should count how many values are actually being read into the array
* The input loop must also check to see if there’s still room for more elements in the array
* After the input loop, the program uses the count, not the array’s length, as the loop limit when processing the array elements

Partially-Filled Examples

* User Input: SentinelArray.java (from section 7.4)
  + The input loop stops either when the user enters the sentinel value or when the array is full
  + The loop to display these values stops at count, not at the length of the array
* File Input: FileArray.java (from section 7.4)
  + Recall that a program determines whether or not a file contains additional unread data using the Scanner method hasNext()
  + The input loop terminates either when hasNext() returns false or when the array is full

Array Miscellany

* A method may return an array
  + The return type should include empty square brackets
  + Example: ReturnArray.java (Code Listing 7-14)
* Multiple sets of values can be stored in parallel arrays
  + The subscript ties the information together between different arrays
  + Example: MonthDays.java (Code Listing 7-15)
* When you declare an array of objects, each array element is an object variable
  + Use a loop to call a constructor for each array element
  + Invoke a method using the dot operator after the subscript
  + Example: ObjectArray.java (Code Listing 7-16)