Java Chapter 08 – Arrays

**Arrays:**

* A named list of data items that all have the same data type
* Each data item is an **element** of the array

\*Can use individual Array Elements in the same manner as you use any single variable of the same type

* Useful when storing just 1 value at a time in memory does not satisfy your needs

**Declaring Arrays:**

* Array Declaration:
* Same as declaring any simple variable, but w/ ‘[ ]’ after the type

int [] idNums;

* Conventional naming for Identifiers 🡪
* Like variables, Lower Camel Case 🡪 idNums
* Often named using a plural noun 🡪 salesFigures
* Often named by adding a final word implying a group 🡪 salesList, salesArray
* Reserve Memory Space:
* idNums = new int [10];
* \*Combine Declaration & Reservation 🡪 int [] idNums = new int[10];

\*\*This statement reserves 10 memory locations for 10 int values

* Subscript:
* Distinguish each intNums item from the others in the list w/ subscript aka ‘sub’

intNums[0] 🡪 1st element in the Array 🡪 always 0

intNums[9] 🡪 10th element in the Array 🡪 always 1 less than the Array’s size

* \*\*When declaring or accessing Array, can use any expression to represent the size as long as the expression is an integer
* Java Allows: Variables, Arithmetic Expressions, Method Return Values to be used as Array sizes
* Literal Integer Constant 🡪 double[] moneyValues = new double[10];
* Named Integer Constant 🡪 double[] moneyValues = new double[NUMBER\_ELS];
* Integer Variable 🡪 double[] moneyValues = new double[numberOfEls];
* Calculation 🡪 double[] moneyValues = new double[x + y \* z];
* Method’s Return Value 🡪 double[] moneyValues = new double[getElements()];

**Initializing Arrays:**

* Assign nondefault values to array elements upon creation 🡪 use an **initializing list**
* Populate the Array 🡪 int idNums = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
* Do not give Array size 🡪 size is based on the # of values placed in the list
* Do not use ‘new’ 🡪 new memory is assigned based on the length of the list
* \*\*Cannot directly initialize part of an Array – initialize every element or none of them

**Using Variable Subscripts w/ Arrays:**

\*\*Power of Arrays become apparent when you begin to use subscripts that are variables rather than subscripts that are constant values

Examples use 🡪 **int[] scoreArray = {1, 2, 3, 4, 5 };**

* Performing the Same operation on each Array Element:
* Common to perform loops that vary the LCV from 0 to one less than the Array size

Final int **INCREASE = 3**;

for(sub = 0; sub < 5; ++sub)

scoreArray[sub] += **INCREASE**;

* Convenient to declare a named constant (equal to the Array size) & use it as the loops limiting value

final int **NUMBER\_OF\_SCORES = 5**;

for(sub = 0; sub < **NUMBER\_OF\_SCORES**; ++sub)

scoreArray[sub] += INCREASE;

* MORE Convenient to use a field (instance variable) that’s automatically assigned a value for every Array you create

🡪 **length field**: contains the number of elements in the Array

for(sub = 0; sub < **scoreArray.length**; ++sub)

scoreArray[sub] += INCREASE;

\*\*Don’t confuse the length property with the String method length()

* Enhanced For Loop:
* Allows you to cycle through an Array w/out specifying the starting & ending points for LCV
* Both Loop Statements display every element the Array

for(int sub = 0; sub < scoreArray.length; ++sub)

System.out.println(scoreArray[sub]); // for loop

for(int val : scoreArray)

System.out.println(val); // for each loop

* Val 🡪 same type as the Array following the colon
* Val 🡪 takes on each value in the Array
* Reads “for each val in scoreArray, display val”
* \*\*Using part of an Array:
* When you don’t want to use every value in an Array

**Arrays of Objects:**

* Can declare Arrays that hold elements of Objects

public class Employee

{

private in empNum;

private double empSal;

Employee(int e, double s)

{

empNum = e;

empSal = s;

}

Public in getEmpNum() { return empNum; }

Public double getSalary() { return empSal; }

}

* More Convenient to create an Array of Employee Objects, vs creating separate Employee Objects
* Declare an Array that holds & reserves computer memory for 7 Employee Objects:

Employee[] emps = new Employee[7];

* Call the 7 individual constructors to construct the Employee Objects:

\*Constructs each Employee w/ an ID # & a salary of $15,000

final in START\_NUM = 101;

final double STARTING/SALARY = 15\_000;

for(int x = 0; x < emps.length; ++x)

emps[x] = new Employee(START\_NUM + x, STARTING\_SALARY);

* Construct Array of Objects using default Constructor:

Final int NUM\_ITEMS = 1000;

InventoryItem[] items = new InventoryItem[NUM\_ITEMS];

For(int x = 0; x < NUM\_ITEMS; ++x)

Items[x] = new InventoryItem();

* Using Methods of an Object in the Array:

for(int x = 0; x < emps.length; ++x)

System.out.println(**emps[x].getEmpNum()** + “ ” + **emps[x].getSalary()**);

* Using Enhanced for Loops w/ Objects:

**\***Eliminates the need to use a limiting value for the loop

**\***Eliminates the need for a subscript following each element

for(Employee **worker** : emps)

System.out.println(**worker.getEmpNum()** + “ ” + **worker.getSalary()**);

* Manipulating Array of Strings:
* Can create an Array of String Objects:

String[] deptNames = { “Accounting”, “Human Resources”, “Sales” };

* Display the list of Strings stored in this Array:

for(int a = 0; a < deptNames.length; ++a)

System.out.println(“deptNames[a]);

* deptNames.length 🡪 Refers to length of Array (3 Elements)
* deptNames[0].length() 🡪 Refers to the length of the String (‘Accounting’ – 10)

**Searching Arrays:**

* When you want to determine whether a variable holds one of many valid values, one option is to use a series of if statements to compare the variable to a series of valid values 🡪 Uses a lot of code
* Better solution is to compare the variable to a list of valid values in an Array 🡪 **Searching the Array**
* Initialize the array with the valid values:

int[] validValues = { 2, 4, 6, 8, 10 };

* Use a for loop to loop through the Array & set a bool variable to true if a match is found:

for(int x = 0; x < validValues.length; ++x){

if(itemsOrdered = validValues[x])

validItem = true; // this variable is a flag

} 🡪 Checks the itemsOrdered value against each of the 5 Array values

**Using Parallel Arrays:**

* Set up another Array w/ the same # of elements & corresponding data, & can use the same subscript to access additional info
* Say the 5 itemsOrdered items have 5 different prices, this Array holds those prices:
* \*Prices must appear in same order as their corresponding spot in the validValues Array

double[] prices = { 0.20, 0.40, 0.60, 0.80, 1.00 };

* Now, the same for loop that finds valid item numbers also finds the price:

for(int x = 0; x < validValues.length; ++x){

if(itemsOrdered = validValues[x])

validItem = true;

itemPrice = prices[x];

} 🡪 When the ordered item’s # is found in the validValues Array, the itemPrice value is “pulled” from the prices Array

\*As soon as a match for an itemOrdered is found 🡪 it’s most efficient to end loop early to save time

* A for loop w/ an early exit 🡪 changing LCV in loop body not always best
* A for loop that uses a compound test for termination 🡪 better solution

for(int x = 0; x < validValues.length && !validItem; ++x){...

**Range Match:**

* Searching an Array for an exact match is not always practical
* A better option might be to create 2 corresponding Arrays & perform a **Range Match**
* Compare a value to the endpoints of numerical ranges to find the category in which a value belongs
* One Array holds 5 discount rates:

int[] discountRangeLimits = { 1, 12, 50, 100, 200 };

* The other Array holds 5 discount range limits:

double[] discountRates = { 0, 0.10, 0.14, 0.18, 0.20 };

**Passing Arrays:** (To Methods)

* Can pass a single Array Element to a method in same manner as you pass any variable
* \*In Java – an Array is not passed by reference, but a reference to an Array is passed by value

**Returning Arrays:** (From Methods)

* Methods can return Array References
* Include ‘[ ]’ w/ return type in the Method header:

public static int[] getArray()

{

Int[] scores = { 90, 80, 70, 60 };

Return scores;

} 🡪 Can store returned value in any integer Array reference:

int[] scoresFromMethods = getArray();