NCP2103: Object-Oriented Programming (Java Programming)

Arrays

Module 8

Errol John M. Antonio
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

Objectives

- Declare and initialize an array
- Use subscripts with an array
- Declare and use arrays of objects
- Search an array
- Pass arrays to and return arrays from methods

Declaring and Initializing an Array

- Array
  - Named list of data items
  - All have same type
- Declare array variable
  - Same way as declaring any simple variable
  - Insert pair of square brackets after type

Declaring and Initializing an Array (cont'd.)

- `double[] salesFigure;`
- `int[] idNum;`
- Still need to reserve memory space
  - `sale = new double[20];`
  - `double[] sale = new double[20];`
- Subscript
  - Integer contained within square brackets
  - Indicates one of array's variables or elements

Declaring and Initializing an Array (cont'd.)

- Array's elements numbered beginning with zero
  - Can legally use any subscript from 0 through 19
  - When working with array that has 20 elements
- Work with any individual array element
  - Treat no differently than single variable of same type
  - Example: `sale[0] = 2100.00;`

Declaring and Initializing an Array (cont'd.)

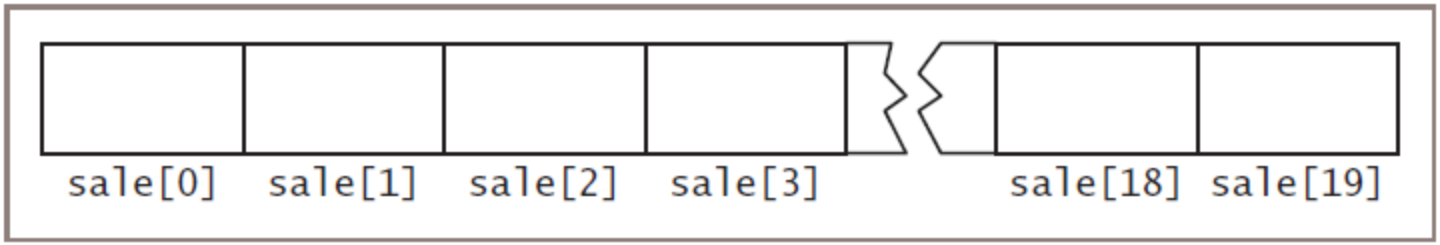

Figure 8-1 An array of 20 `sale` items in memory

Initializing an Array

- Variable with reference type
  - Such as array
  - Holds memory address where value stored
- Array names
  - Represent computer memory addresses
  - Contain references
- Declare array name
  - No computer memory address assigned
  - Has special value `null`
    - Unicode value `'\u0000'`

Initializing an Array (cont'd.)

- Use keyword `new` to define array
  - Array name acquires actual memory address value
- `int[] someNums = new int[10];`
  - Each element of `someNums` has value of 0
- `char` array elements
  - Assigned `'\u0000'`
- `boolean` array elements
  - Automatically assigned value `false`

Initializing an Array (cont'd.)

- Assign nondefault values to array elements upon creation

```
int[] tenMult = {10, 20, 30, 40, 50,
60};
```

- Populating the array
  - Providing values for all the elements in an array

Using Subscripts with an Array

- Scalar
  - Primitive variable
- Power of arrays
  - Use subscripts that are variables
  - Rather than constant subscripts
  - Use a loop to perform array operations

```
for (sub = 0; sub < 5; ++sub)
  scoreArray[sub] += 3;
```

Using Subscripts with an Array (cont'd.)

- Application contains array
  - Use every element of array in some task
  - Perform loops that vary loop control variable
    - Start at 0
    - End at one less than size of array
- Convenient to declare symbolic constant equal to size of array

```
final int NUMBER_OF_SCORES = 5;
```

Using Subscripts with an Array (cont'd.)

- Field
  - Instance variable
  - Automatically assigned value for every array created
- `length` field
  - Contains number of elements in array

```
for(sub = 0; sub <
scoreArray.length;    ++sub)
   scoreArray[sub] += 3;
```

Using Subscripts with an Array (cont'd.)

- Enhanced `for` loop
  - Cycle through array
  - Without specifying starting and ending points for loop control variable

```
for(int val : scoreArray)
   System.out.println(val);
```

Declaring and Using Arrays of Objects

- Create array of `Employee` objects

```
Employee[] emp = new Employee[7];
```

  - Must call seven individual constructors

```
final double PAYRATE = 6.35;
for(int x = 0; x < NUM_EMPLOYEES;
++x)
   emp[x] = new Employee(101 + x,
   PAYRATE);
```

Using the Enhanced `for` Loop

- Use the enhanced `for` loop to cycle through an array of objects
  - Eliminates the need to use a limiting value
  - Eliminates the need for a subscript following each element

```
for(Employee worker : emp)
   System.out.println(worker.getEmpNu
   m() + " " + worker.getSalary();
```

Manipulating Arrays of `Strings`

- Create array of `Strings`

```
String[] deptName = {"Accounting",
"Human Resources", "Sales"};
for(int a = 0; a < deptName.length;
++a)
  System.out.println(deptName[a]);
```

Manipulating Arrays of `Strings` (cont'd.)


```
import javax.swing.*;
public class SearchList
{
   public static void main(String[] args)
   {
      String[] deptName = {"Accounting", "Human Resources", "Sales"};
      String dept;
      int x;
      boolean deptWasFound = false;
      dept = JOptionPane.showInputDialog(null,
         "Enter a department name");
      for(x = 0; x < deptName.length; ++x)
         if(dept.equals(deptName[x]))
           deptWasFound = true;
      if(deptWasFound)
         JOptionPane.showMessageDialog(null, dept +
            " was found in the list");
      else
         JOptionPane.showMessageDialog(null, dept +
            " was not found in the list");
   }
}
```

Figure 8-3 The SearchList class

Searching an Array

- Determine whether variable holds one of many valid values
  - Use series of `if` statements
  - Compare variable to series of valid values

Searching an Array (cont'd.)

- Searching an array
  - Compare variable to list of values in array

```
for(int x = 0; x <
validValues.length; ++x)
{
   if(itemOrdered == validValues[x])
    validItem = true;
}
```

Searching an Array (cont'd.)

- Parallel array
  - One with same number of elements as another
  - Values in corresponding elements related
- Alternative for searching
  - Use `while` loop

```
import javax.swing.*;
public class FindPrice
{
   public static void main(String[] args)
   {
      final int NUMBER_OF_ITEMS = 10;
      int[] validValues = {101, 108, 201, 213, 266,
         304, 311, 409, 411, 412};
      double[] prices = {0.29, 1.23, 3.50, 0.69, 6.79,
         3.19, 0.99, 0.89, 1.26, 8.00};
      String strItem;
      int itemOrdered;
      double itemPrice = 0.0;
      boolean validItem = false;
      strItem = JOptionPane.showInputDialog(null,
         "Enter the item number you want to order");
      itemOrdered = Integer.parseInt(strItem);
      for(int x = 0; x < NUMBER_OF_ITEMS; ++x)
      {
         if(itemOrdered == validValues[x])
         {
            validItem = true;
            itemPrice = prices[x];
         }
      }
      if(validItem)
         JOptionPane.showMessageDialog(null, "The price for item " +
            itemOrdered + " is $" + itemPrice);
      else
         JOptionPane.showMessageDialog(null,
            "Sorry - invalid item entered");
   }
}
```

Figure 8-5 The FindPrice application that accesses information in parallel arrays

Searching an Array (cont'd.)

```
x = 0;
while(x < NUMBER_OF_ITEMS && itemOrdered != validValues[x])
   ++x;
if(x != NUMBER_OF_ITEMS)
{
   validItem = true;
   itemPrice = prices[x];
}
```

Figure 8-8 A while loop with an early exit

Searching an Array For a Range Match

- Searching array for exact match
  - Not always practical
- Range match
  - Compare value to endpoints of numerical ranges
  - Find category in which value belongs

```
import javax.swing.*;
public class FindDiscount
{
   public static void main(String[] args)
   {
      final int NUM_RANGES = 5;
      int[] discountRangeLimit = {   1,   13,   50,  100, 200};
      double[] discountRate =    {0.00, 0.10, 0.14, 0.18, 0.20};
      double customerDiscount;
      String strNumOrdered;
      int numOrdered;
      int sub = NUM_RANGES - 1;
      strNumOrdered = JOptionPane.showInputDialog(null,
         "How many items are ordered?");
      numOrdered = Integer.parseInt(strNumOrdered);
      while(sub >= 0 && numOrdered < discountRangeLimit[sub])
         --sub;
      customerDiscount = discountRate[sub];
      JOptionPane.showMessageDialog(null, "Discount rate for " +
         numOrdered + " items is " + customerDiscount);
   }
}
```

Figure 8-9 The FindDiscount class

Passing Arrays to and Returning Arrays from Methods

- Pass single array element to method
  - Same as passing variable
- Passed by value
  - Copy of value made and used in receiving method
  - All primitive types passed this way

Passing Arrays to Methods (cont'd.)

- Reference types
  - Object holds memory address where values stored
  - Receiving method gets copy of array's actual memory address
  - Receiving method has ability to alter original values in array elements

```
public class PassArray
{
   public static void main(String[] args)
   {
      final int NUM_ELEMENTS = 4;
      int[] someNums = {5, 10, 15, 20};
      int x;
      System.out.print("At start of main: ");
      for(x = 0; x < NUM_ELEMENTS; ++x)
         System.out.print(" " + someNums[x] );
      System.out.println();
      methodGetsArray(someNums);
      System.out.print("At end of main: ");
      for(x = 0; x < NUM_ELEMENTS; ++x)
         System.out.print(" " + someNums[x]);
      System.out.println();
   }
   public static void methodGetsArray(int[] arr)
   {
      int x;
      System.out.print("At start of method arr holds: ");
      for(x = 0; x < arr.length; ++x)
         System.out.print(" " + arr[x] );
      System.out.println();
      for(x = 0; x < arr.length; ++x)
         arr[x] = 888;
      System.out.print(" and at end of method arr holds: ");
      for(x = 0; x < arr.length; ++x)
         System.out.print(" " + arr[x] );
      System.out.println();
   }
}
```

Figure 8-13 The PassArray class

Returning an Array from a Method

- Method can return an array reference
- Include square brackets with the return type
  - In the method header

You Do It

- Creating and populating an array
- Initializing an array
- Using a `for` loop to access array elements
- Creating parallel arrays to eliminate nested `if` statements
- Creating an application with an array of objects

You Do It (cont'd.)

- Creating an interactive application that creates an array of objects
- Passing an array to a method

Don't Do It

- Don't forget that the lowest array subscript is 0
- Don't forget that the highest array subscript is one less than the length
- Don't forget that length is an array property and not a method
- Don't place a subscript after an object's field or method name when accessing an array of objects
- Don't assume that an array of characters is a string

Don’t Do It (cont'd.)

- Don’t forget that array names are references
- Don’t use brackets with an array name when you pass it to a method

Summary

- Array
  - Named list of data items
  - All have same type
- Array names
  - Represent computer memory addresses
- Shorten many array-based tasks
  - Use variable as subscript
- `length` field
  - Contains number of elements in array

Summary (cont'd.)

- You can declare arrays that hold elements of any type, including `String`s and other objects
- Search array to find match to value
- Perform range match
- Pass single array element to method

End of Module.

REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.