Java Programming: Guided Learning with Early Objects

Chapter 7 - Arrays

Objectives

- Learn about arrays
- Explore how to declare and manipulate data in arrays
- Learn about the instance variable `length`
- Understand the meaning of "array index out of bounds"
- Discover how to pass an array as a parameter to a method
- Discover how to manipulate data in a two-dimensional array
- Learn about multidimensional arrays

Why Do We Need Arrays?

- Example:
  - Read five numbers from a user, print in reverse order
  - Must read and store each number in a separate variable
  - After reading, print in reverse order
- More convenient to specify a number of variables with their data type in one structure
- In Java, the structure is an array

Arrays

- Collection of fixed number of elements (components)
  - All elements are the same data type
- One-dimensional array: elements arranged in a list
- Syntax:
  `dataType[] arrayName = new dataType[intExp];`
- Arrays are objects
  - `arrayName` is a reference variable

Alternate Ways to Declare an Array

- Declare array as: `int list[];`
- Consider the following declarations:

```
int alpha[], beta;
int[] gamma, delta;
```

  - Arrays: `alpha`, `gamma`, `delta`
  - `int` variable: `beta`

Accessing Array Elements

- Syntax:

```
arrayName[indexExp]
```

- **Array subscripting operator**: `[]`
- Declare an array list of 10 elements:

```
int[] list = new int[10];
```

- Assignment statement
  - Store 34 in 6th element:

```
int[5] = 34;
```

Specifying Array Size During Program Execution

- Arrays instantiated with their size
  - Can be determined from user input at run time

```
int arraySize;
System.out.print("Enter array size:");
arraySize = console.nextInt();
System.out.println();
int[] list = new int[arraySize];
```

Array Initialization During Declaration

- Array initialized with specific values when declared

```
double[] sales = {12.25, 32.50, 16.90,
        23, 45.68};
```

- **Initializer list**:
  - **Initial values** placed between braces
  - Separated by commas
- Size of array determined by number of values in initializer list

- If initialized when declared, do not use `new`

Arrays and the Instance Variable `length`

- Array is an object
- Instance variable `length` associated with instantiated array
  - Variable `length` `public`
  - Accessed with array name and dot operator
  - Example: `int[] numList = new int[10];`
    - `numList.length` is 10

Processing One-Dimensional Arrays

- Stepping through an array accomplished with a loop
- If number of elements is known, use a `for` loop:

```
for (int i = 0; i < list.length; i++)
    System.out.print(list[i] + " ");
```

- Use of variable `length` is convenient and safe
  - Avoids use of integer literal representing array length
  - Protects array out of bounds exception

Array Index Out of Bounds Exception

- Example:

```
double[] num = double[10];
```

  - Valid indices are 0 through 9
  - Array is **in bounds** if index >= 0 and index <= 9
  - Array is **out of bounds** if index < 0 or index > arraySize – 1
- Java throws `ArrayIndexOutOfBoundsException` if index goes out of bounds

Base Address of an Array

- Base address of array is memory location of first element
- If `list` is a one-dimensional array, base address of `list` is address of `list[0]`
- Value of variable `list` is base address

Declaring Arrays as Formal Parameters to Methods

- Arrays can be formal parameters to methods

- Syntax to declare method:
```
public static void arrayPar
       (int[] listA){//…}
```
- Syntax to call method with actual parameters:
```
arrayPar(intList);
```

Arrays as Parameters to Methods

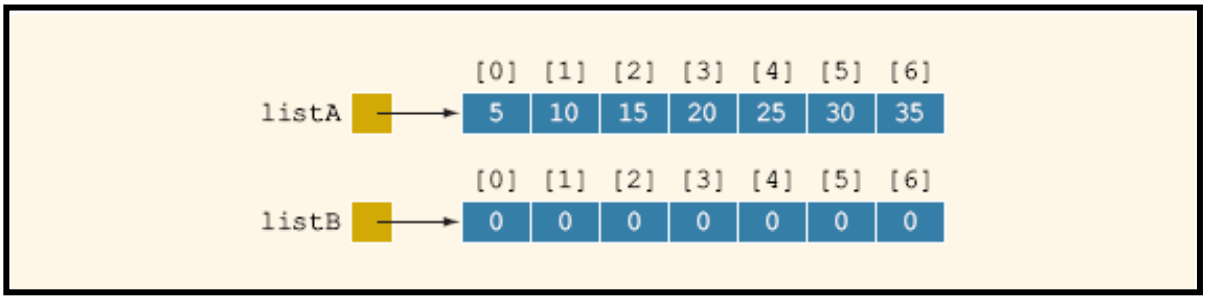
- Arrays can be passed as parameters to methods
  - List other objects
- Number of elements in array might be less than array length
- Process only elements that hold valid data
- Declare additional formal parameter specifying number of valid elements
```
void printArray (int[]list, int numElems)
```
- Base address passed to formal parameter

Searching an Array (List) for a Specific Item

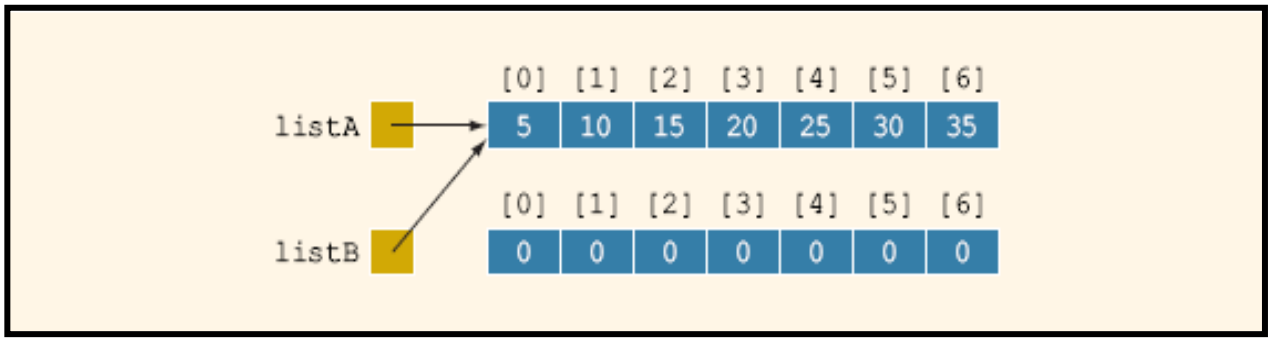
- Starting with first element, sequential search compares `searchItem` with elements in list
- Search continues until:
  - Element found
  - List exhausted
- If element found, return index
  - Otherwise, return -1

Assignment Operator, Relational Operators, and Arrays: A Precaution

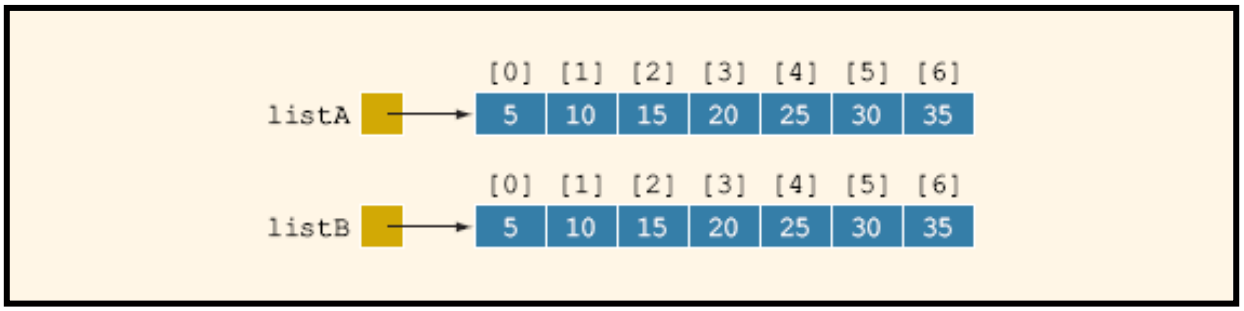
- Consider the statement:
```
listB = listA;
```
  - `listA` is a memory address
  - Statement makes a shallow copy
- Make a deep copy element by element
- Same applies to comparison
  - `listA == listB` compares memory addresses
  - Compare elements by iterating through the array



Arrays `listA` and `listB`



Arrays after the statement `listB = listA;` executes



`listA` and `listB` after the `for` loop executes

Designing a Class to Process `int` Arrays (Optional)

- Design a class to process arrays
- Class uses an instance variable to store array elements
  - Uses instance variable to track number of elements in the array

Arrays of Objects

- Previous sections learned how to use arrays to store and manipulate values of primitive types
- Arrays can also store object references and manipulate objects

Arrays of `String` Objects

- Create an array of strings:
```
String[] nList = new String[5];
```
- Assign a value to an array element:
```
nList[0] = "Amanda Green";
```
  – Element 0 is a reference to a `String` object
- Output names using `for` loop:
```
for (int i = 0; i < nList.length; i++)
   System.out.println(nList[i]);
```

Arrays of Objects of Other Classes

- Recall `class Clock`
- Declare array of 100 elements of type `Clock`:
```
Clock[] arrivalTime = new Clock[100];
```
- Instantiate `Clock` objects for each array element:
```
for (int j=0; j <arrivalTime.length;j++)
   arrivalTime[j] = new Clock();
```
- Use methods of `class Clock` to manipulate array elements:
```
arrivalTime[49].setTime(8,5,10);
```

Arrays and Variable Length Parameter List (Optional)

- Variable length parameter list simplifies code
- Syntax (ellipsis part of the syntax):
`dataType … identifier`
- Formal parameter list of variable length
  – Can specify any number of actual parameters
  – Can specify an array as a parameter
  – Number of actual parameters may be zero
- Method can have both variable-length formal parameter and other formal parameters

- At most, one variable-length formal parameter
- Variable-length formal parameter must be last
- Access elements in an array with **foreach** loop
- Syntax:
  `for (dataType ident : arrayName)`

Example :-

This Class contains methods to manipulate data in a one-dimensional array :-

```
package ch7arrays;

/**
 *
 * @author Husain Gholoom
 */
public class ArrayListClass {

   int[] list;
   int numOfElements;

    // Default constructor.
    // Instantiates an array of size 5.
    // Postcondition: list points to an array of size 5.
    //          numOfElements = 0;

   public ArrayListClass()
   {
      list = new int[5];
      numOfElements = 0;
   }

    // Constructor.
    // Instantiates an array of the size specified by the user.
    // Postcondition: list points to an array of size
    //          arraySize.
    //          numOfElements = 0;

   public ArrayListClass(int arraySize)
   {
      list = new int[arraySize];
      numOfElements = 0;
   }
```

```
 // Method to return the length of the array.
 // Postcondition: The value of list.length is
 //            returned.

public int listSize()
{
   return list.length;
}

 // Method to insert an element at the end of the list.
 // Postcondition: If numOfElements < list.length - 1
 //              list[numOfElements++] = elem;
 //            otherwise output an appropriate error
 //            message.

public void insertEnd(int elem)
{
   if (numOfElements < list.length)
      list[numOfElements++] = elem;
   else
      System.out.println("Cannot insert because the "
                  + "list is full.");
}

 // Method to return the element at the position specified
 // by index.
 // This method does not determine if index is out-of-bound
 // Postcondition: retrun list[index];

public int elementAt(int index)
{
   return list[index];
}
```

```
   // Method to return the elements of list as a string.
   // Elements are separated by commas and spaces.
   // Postcondition: Elements of list are returned as a
   //                string.

public String toString()
{
   String str = "[";

   for (int index = 0; index < numOfElements - 1; index++)
      str = str + list[index] + ", ";

   if (numOfElements > 0)
      str = str + list[numOfElements - 1] + "]";
   else
      str = str + "]";

   return str;
}

   // Method to return the sum of the elements of list
   // Postcondition: Sum of the elements of list is returned.

public int sumArray()
{
   int sum = 0;

   for (int index = 0; index < numOfElements; index++)
       sum = sum + list[index];

   return sum;
}
```

```
  // Method to find and return the index of the first
  // occurrence of the largest element, if it repeats,
  // in an int array.
  // This method does not check if list is empty.
  // Postcondition: The index of the first occurence of the
  //                largest element is returned.

public int indexLargestElement()
{
   int maxIndex = 0; //Assume first element is the largest

   for (int index = 1; index < numOfElements; index++)
       if (list[maxIndex] < list[index])
          maxIndex = index;

   return maxIndex;
}

  // Method to copy one array into another array. The
  // elements of otherList are copied into list. The
  // array list must be at least as large as the number
  // of elements to be copied.
  // Postcondition: The parameter numOfElements specifies
  //                the number of elements of list1 to be
  //                copied into list2.

public void makeCopy(ArrayListClass otherList)
{
   numOfElements = otherList.numOfElements;

   for (int index = 0; index < otherList.numOfElements;
                     index++)
       list[index] = otherList.list[index];
}
```

```
    // Method to sort list using selection sort algorithm.
    // Postcondition: The array list is sorted

  public void selectionSort()
  {
     int smallestIndex;
     int minIndex;
     int temp;

     for (int index = 0; index < numOfElements - 1; index++)
     {
           //Step a
        smallestIndex = index;

        for (minIndex = index + 1; minIndex < numOfElements;
                              minIndex++)
           if (list[minIndex] < list[smallestIndex])
              smallestIndex = minIndex;

           //Step b
        temp = list[smallestIndex];
        list[smallestIndex] = list[index];
        list[index] = temp;
     }
  } //end selectionSort

}
```

This main program illustrates how to use selection sort Method of the class ArrayListClass in the program

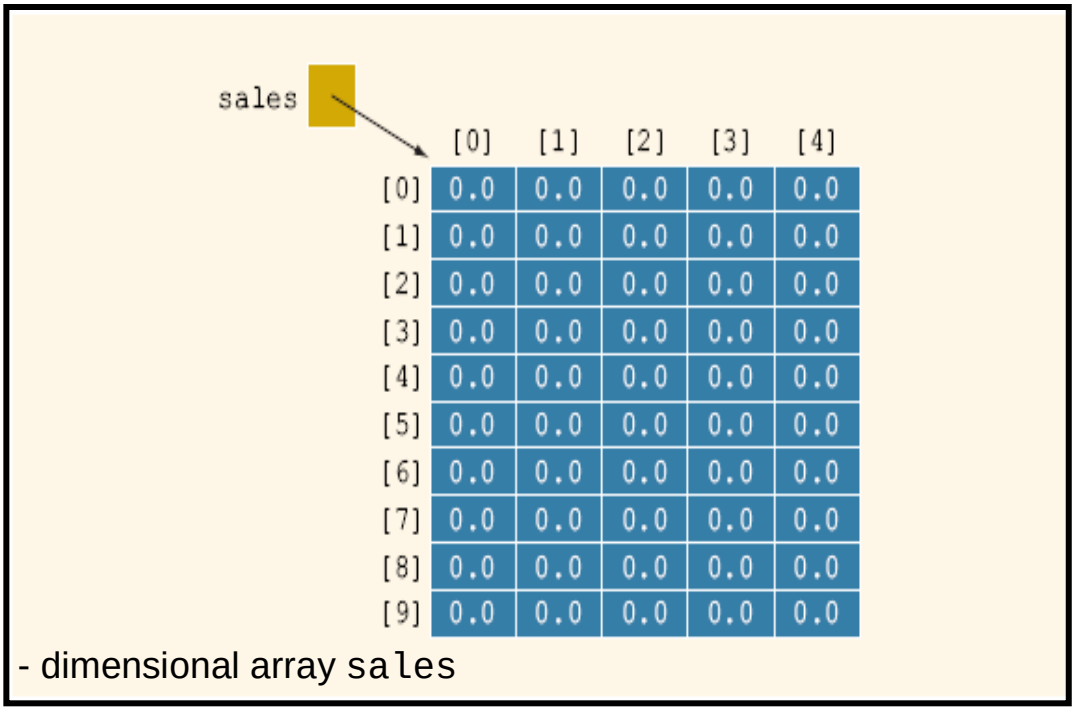
```
package ch7arrays;

/**
 *
 * @author Husain Gholoom
 */

import java.util.*;
public class Main {

    /**
     * @param args the command line arguments
     */
    static Scanner console = new Scanner(System.in);
    public static void main(String[] args) {

        ArrayListClass myList =       new ArrayListClass(10);

        int num;

        System.out.print("Enter "
                    + myList.listSize()
                    + " integers: ");

        for (int index = 0;  index < myList.listSize();   index++)
        {
           num = console.nextInt();
           myList.insertEnd(num);
        }

        num = myList.listSize();
        System.out.println();
        System.out.println("Size of the List Array "
                        + num);
```

```
        num = myList.sumArray();
        System.out.println();
        System.out.println("Sum  of elements of the List Array "
                        + num);


       num = myList.indexLargestElement();
        System.out.println();
        System.out.println("Index of the first occurrence of "
                          +"largest elements of the List Array "
                          +" if it repeats in the int array   "
                        + num);

        System.out.println();

        System.out.println("Before sorting "
                        + "myList:\n" + myList);

        myList.selectionSort();

        System.out.println("After sorting "
                        + "myList: " + myList);

    }           }
```

Two-Dimensional Arrays

- **Two-dimensional array**: collection of a fixed number of elements
  - Arranged in rows and columns
  - All elements of the same type
- Syntax:

```
dataType [][] arrayName;
arrayName = new dataType[intA][intB];
```



Two - dimensional array `sales`

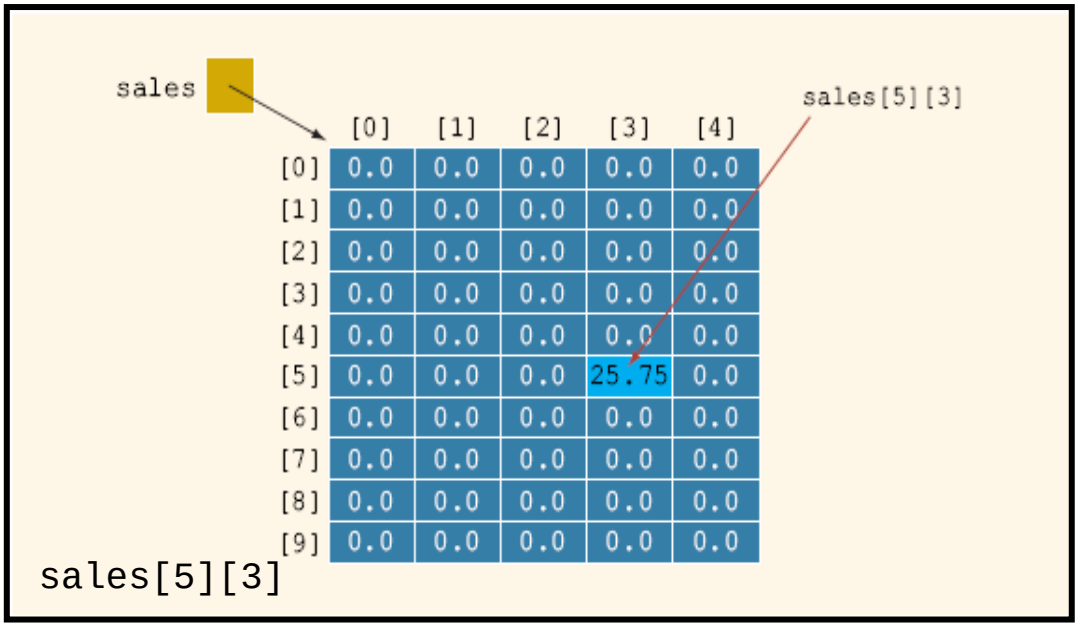
Accessing Array Elements

- Access a two-dimensional array with a pair of indices
  - One for row, one for column
- Syntax:

```
arrayName[indexExp1][indexExp2]
```

- Example:

```
sales[5][3] = 25.75;
```



Two-Dimensional Arrays and the Instance Variable `length`

- Instance variable length determines number of rows as well as number of columns
- Example:

```
int[][] matrix = new int[20][15];
```

  - Value of `matrix.length` is 20
  - Value of `matrix[0].length` is 15

Two-Dimensional Ragged Arrays

- Java allows different number of columns for each row
- Rows must be instantiated separately
- **Ragged arrays**: different number of columns for each row

Two-Dimensional Array Initialization During Declaration

- Two-dimensional arrays can be initialized when declared
- Example:

```
int[][] board = { { 2, 3, 1},
                  {15,25,13},
                  {20, 4, 7},
                  {11, 18, 14}};
```

Processing Two-Dimensional Arrays

- Three ways to process a two-dimensional array:
  - Process entire array
  - **Row processing**
  - **Column processing**
- Each row or column can be processed as a one-dimensional array

Initialization

- `for` loop initializes each element of a row:

```
int row = 4;
for (int col = 0;
     col < matrix[row].length; col++)
   matrix[row][col] = 10;
```

- Nested `for` loops initialize each element:

```
for (int row = 0; row < matrix.length;       row++)
   for (int col = 0;
        col < matrix[row].length; col++)
       matrix[row][col] = 10;
```

Print

- Use a nested `for` loop to output elements of matrix:

```
for (int row = 0; row < matrix.length;       row++){
  for (int col = 0;
       col <      matrix[row].length; col++)
     System.out.print(matrix[row][col]);
  }
  System.out.println();  }
```

Input

- Use `for` loop to input data into row number 4:

```
int row = 4;
for (int col = 0;
     col < matrix[row].length; col++)
   matrix[row][col] = console.nextInt();
```

- Input data into each element of array:

```
for (int row = 0; row < matrix.length;   row++)
  for (int col = 0;
       col <      matrix[row].length; col++)
     matrix[row][col]=console.nextInt();
```

Sum By Row

- Use `for` loop to sum elements in row 4:

```
int row = 4;
for (int col = 0;
     col < matrix[row].length; col++)
   sum += matrix[row][col];
```

- Sum each element of array:

```
for (int row = 0; row < matrix.length;   row++)
  for (int col = 0;
       col <      matrix[row].length; col++)
     sum += matrix[row][col];
```

Sum By Column

- Use `for` loop to sum elements in columns:

```
for (int col = 0;
     col < matrix[0].length; col++)
   sum = 0;
   for (int row = 0; row < matrix.length;  row++)
      sum += matrix[row][col];
```

Largest Element in Each Row and Each Column

- Largest element in each row:

```
for (int row = 0; row < matrix.length;  row++){
   largest = matrix[row][0];
   for (int col = 1;
        col < matrix[row].length; col++)
      if (largest < matrix[row][col])
        largest = matrix[row][col];   }
```

Passing Two-Dimensional Arrays as Parameters to Methods

- References to two-dimensional arrays can be passed as parameters to a method
- Methods to process arrays can be placed in a class
- When storing a two-dimensional array in computer memory, Java uses **row order form**
  - First row stored first, followed by second, etc.

Multidimensional Arrays

- Arrays may be three or more dimensions
- ***n*-dimensional array**: collection of a fixed number of variables arranged in *n* dimensions
- Syntax:

```
dataType[][]…[] arrayName =
   new dataType[intA][intB]…[intN];
```

- Example:

```
double[][][] carDealers =
       new double[3][7][5];
```

Summary

- Array: collection of a fixed number of elements of the same data type
  - One-dimensional arrays
  - Two-dimensional arrays
  - Multidimensional arrays
- Array index: any expression that evaluates to nonnegative integer
  - First array index is 0
  - When array instantiated, elements initialized to default value

- Variable `length` contains size of the array
- Base address of an array is address of first element
  - Can be passed as parameters to methods
  - Individual array elements can be parameters
- Can create an array of objects
- Syntax for variable-length formal parameter: `dataType … identifier`
- The foreach loop processes elements in array