7

Arrays & ArrayList

Objectives

- test 1 results / midterm information
- arrays in Java
- multi-dimensional arrays
- passing arrays to methods
- dynamically resizing arrays: ArrayList class
- variable length argument lists

Test 1 results

- great performance
- mean = 85 (min = 12, max = 100)
- test solutions on the website

Midterm

- Monday, April 20
- 20% of the total grade
- material from Chapters 1 – 7
- practice midterm will be available later today

Outline

Chapter 7 overview

- **Arrays**
  - **Data structures**
  - **Related data items of same type**
  - **Remain same size once created (fixed-length entries)**

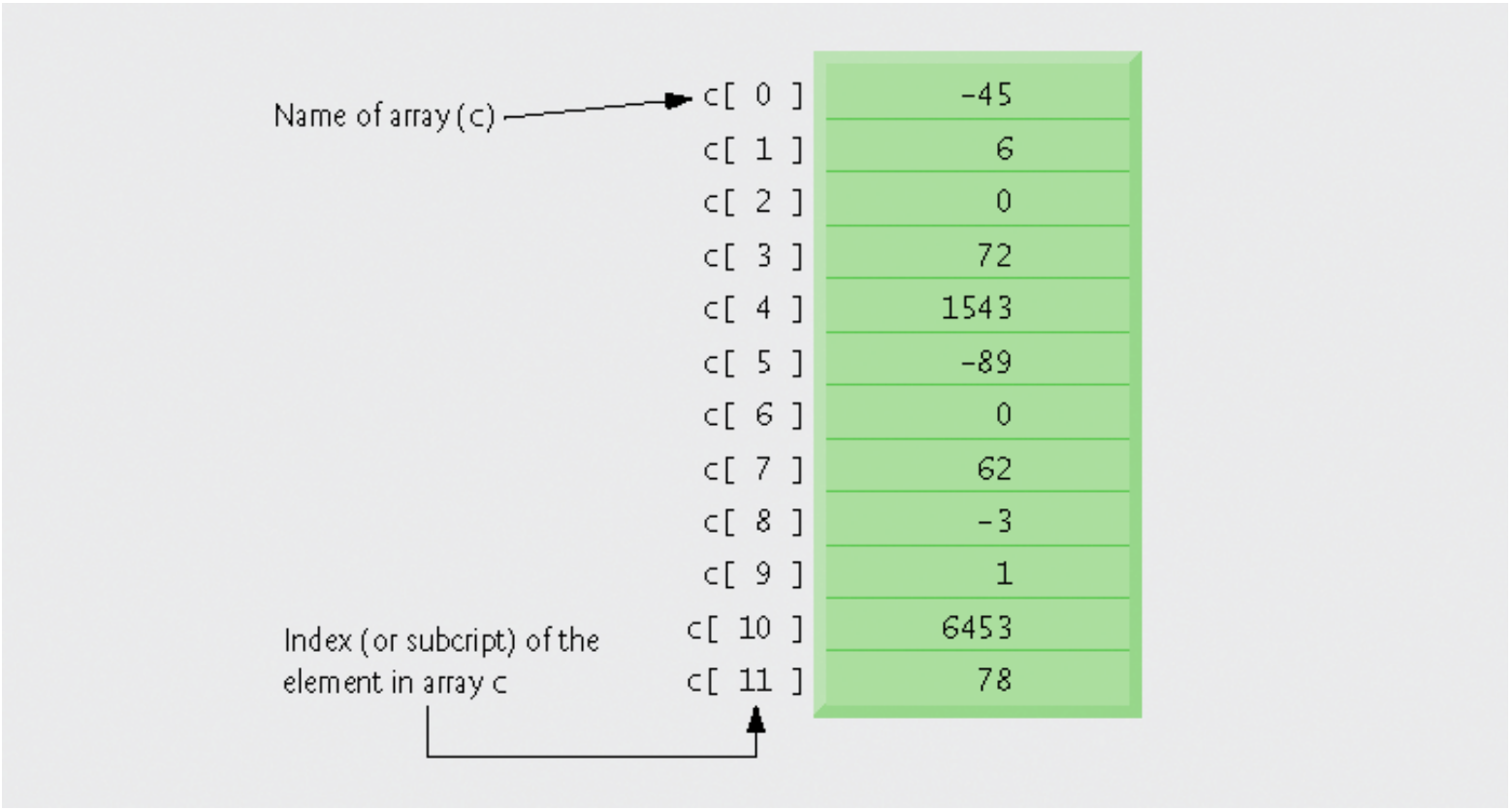

Fig. 7.1 | A 12-element array.

Example array

- **Example array `c`**
  - **`c` is the array *name***
  - **`c.length` accesses array `c`'s *length***
  - **`c` has 12 *elements* (`c[0]`, `c[1]`, … `c[11]`)**
    - **The *value* of `c[0]` is `-45`**
  - **0-based indexing**

Declaring and Creating Arrays

- **Declaring and Creating arrays**
  - **Created dynamically with keyword new**

```
int c[] = new int[ 12 ];
```

    - **Equivalent to**

```
int c[];  // declare array variable
c = new int[ 12 ]; // create array
```

  - **We can create arrays of objects too**

```
String b[] = new String[ 100 ];
```

Array initializer

- **Using an array initializer**
  - **Use *initializer list***
    - **Items enclosed in braces ({})**
    - **Items in list separated by commas**

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

      - **Creates a five-element array**
      - **Index values of 0, 1, 2, 3, 4**
  - **Do not need keyword new**

Enhanced for Statement

- **Enhanced `for` statement**
  - **Iterates through elements of an array or a collection without using a counter**
  - **Syntax**

```
for ( parameter : arrayName )
   statement
```

Outline

EnhancedForTest.java

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main( String args[] )
7     {
8        int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int number : array )
13          total += number;
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class EnhancedForTest
```

```
Total of array elements: 849
```

For each iteration, assign the next element of `array` to `int` variable `number`, then add it to `total`

Enhanced for Statement (Cont.)

- **Lines 12-13 are equivalent to**

```
for ( int counter = 0; counter < array.length; counter++ )
    total += array[ counter ];
```

- **Usage of enhanced for loop**
  - **Can access array elements**
  - **Cannot modify array elements**
  - **Cannot access the counter indicating the index**

Multidimensional Arrays

- **Multidimensional arrays**
  - **Tables with rows and columns**
  - **Most common are two-dimensional arrays (matrices)**
    - **m-by-n array**

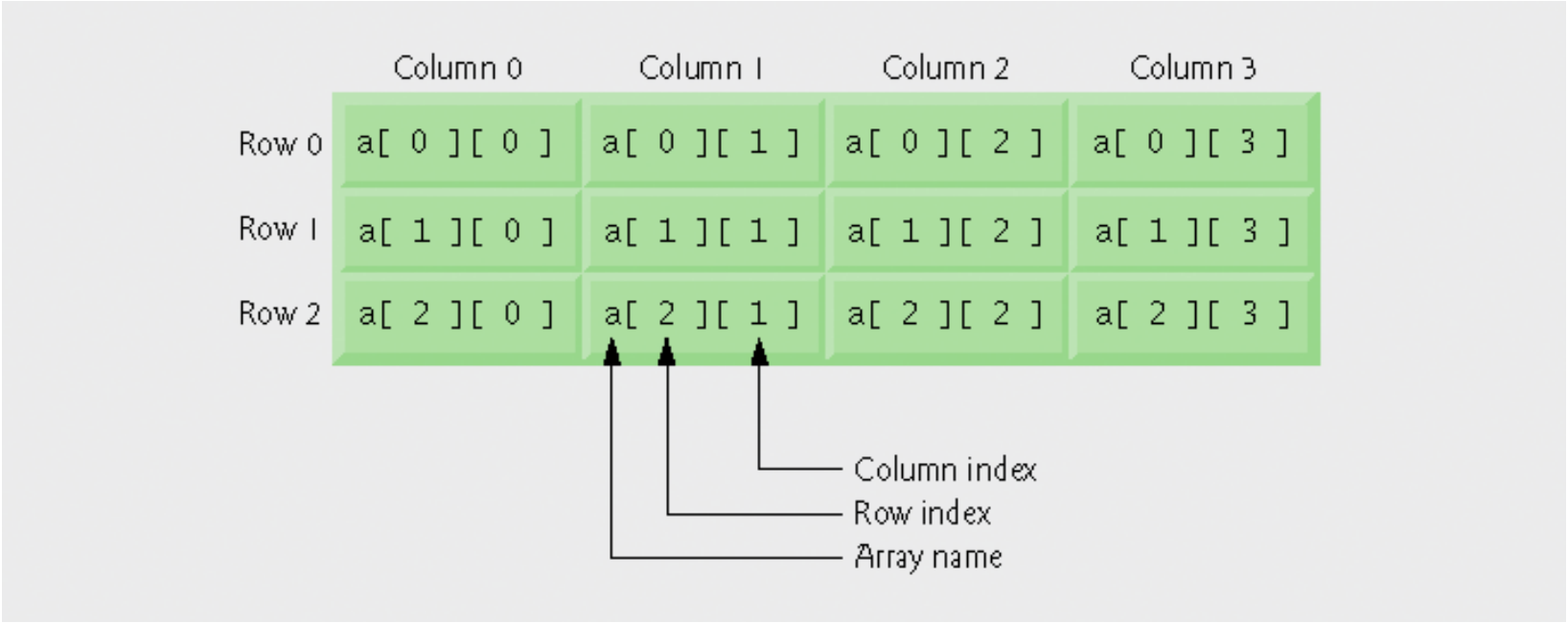

Fig. 7.16 | Two-dimensional array with three rows and four columns.

Multidimensional Arrays (Cont.)

- **Arrays of one-dimensional array**
  - **Declaring two-dimensional array `b[2][2]`**

```
int b[][] = { { 1, 2 }, { 3, 4 } };
```

    - **`1` and `2` initialize `b[0][0]` and `b[0][1]`**
    - **`3` and `4` initialize `b[1][0]` and `b[1][1]`**

```
int b[][] = { { 1, 2 }, { 3, 4, 5 } };
```

    - **row `0` contains elements `1` and `2`**
    - **row `1` contains elements `3`, `4` and `5`**

Multidimensional Arrays (Cont.)

- **Two-dimensional arrays with rows of different lengths**
  - **Lengths of rows in array are not required to be the same**
    - **E.g.,** `int b[][] = { { 1, 2 }, { 3, 4, 5 } };`

Multidimensional Arrays (Cont.)

- **Creating two-dimensional arrays with array-creation expressions**
  - **3-by-4 array**

```
int b[][];
b = new int[ 3 ][ 4 ];
```

  - **Rows can have different number of columns**

```
int b[][];

b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 columns for row 0
b[ 1 ] = new int[ 3 ]; // create 3 columns for row 1
```

```
1  // Fig. 7.17: InitArray.java
2  // Initializing two-dimensional arrays.
3
4  public class InitArray
5  {
6     // create and output two-dimensional arrays
7     public static void main( String args[] )
8     {
9        int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
10       int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12       System.out.println( "Values in array1 by row are" );
13       outputArray( array1 ); // displays array1 by row
14
15       System.out.println( "\nValues in array2 by row are" );
16       outputArray( array2 ); // displays array2 by row
17    } // end main
18
```

Use nested array initializers to initialize `array1`

Use nested array initializers of different lengths to initialize `array2`

Outline

`InitArray.java`

(1 of 2)

Line 9

Line 10

```
19    // output rows and columns of a two-dimensional array
20    public static void outputArray( int array[][] )
21    {
22       // loop through array's rows
23       for ( int row = 0; row < array.length; row++ )
24       {
25          // loop through columns of current row
26          for ( int column = 0; column < array[ row ].length; column++ )
27             System.out.printf( "%d  ", array[ row ][ column ] );
28
29          System.out.println(); // start new line of output
30       } // end outer for
31    } // end method outputArray
32 } // end class InitArray
```

`array[row].length` returns number of columns associated with `row` subscript

Use double-bracket notation to access two-dimensional array values

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```

Outline

`InitArray.java`

(2 of 2)

Line 26

Line 27

Program output

Multidimensional Arrays (Cont.)

- **Common multidimensional-array manipulations performed with `for` statements**
  - **Many common array manipulations use `for` statements**

E.g.,

```
for ( int column = 0; column < a[ 2 ].length; column++ )
      a[ 2 ][ column ] = 0;
```

Exercise: Sum of matrix elements

Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - Array `hourlyTemps`

```
int hourlyTemps[] = new int[ 24 ];
```

    - The method call

```
modifyArray( hourlyTemps );
```

    - Passes array `hourlyTemps` to method `modifyArray`

Passing Arguments to Methods

- **Two ways to pass arguments to methods**
  - **Pass-by-value**
    - **Copy of argument's value is passed to called method**
    - **Every primitive type is passed-by-value**
  - **Pass-by-reference**
    - **Caller gives called method direct access to caller's data**
    - **Called method can manipulate this data**
    - **Improved performance over pass-by-value**
    - **Every object (includes arrays) is passed-by-reference**

Exercise: Reverse array (as a new array)

Implement a static method **`reverseArray`** that returns a reversed input integer array as a new array.

Use the following method prototype, where **`array`** is the input integer array.

```
static int[] reverseArray( int[] array )
```

Using Command-Line Arguments

- **Command-line arguments**
  - **Pass arguments from the command line**
    - `String args[]`
  - **Appear after the class name in the `java` command**
    - `java MyClass a b`
  - **Number of arguments passed in from command line**
    - `args.length`
  - **First command-line argument**
    - `args[ 0 ]`

Lists

- **List**
  - **Ordered `Collection` that can contain duplicate elements**
  - **Sometimes called a *sequence***
  - **Implemented via interface `List`**
    - `ArrayList`
    - `LinkedList`
    - `Vector`

Class `ArrayList`

- **ArrayList is a dynamically sized array**
  - **Generic array of elements**
  - **Specify item type using `<...>` , for example:**

```
ArrayList<String>, ArrayList<Object>
```

- **Instance methods**
  - `get`
  - `size`
  - `add`
  - `contains`
  - **and others ...**

Question

What are main differences between an `int[]` array and `ArrayList<Integer>`?