Arrays

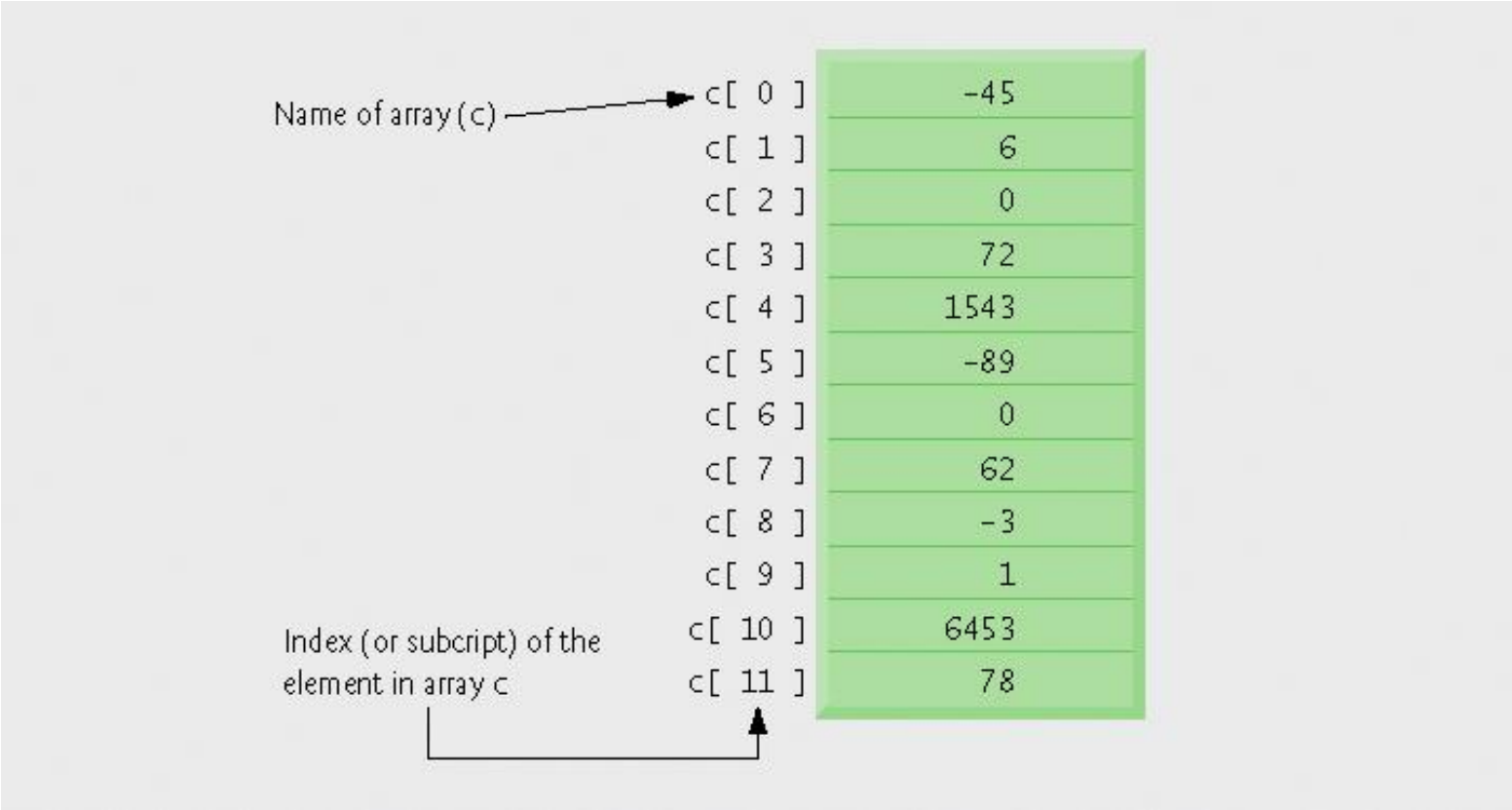
OBJECTIVES

In this chapter you will learn:

- What arrays are.
- To use arrays to store data in and retrieve data from lists and tables of values.
- To declare an array, initialize an array and refer to individual elements of an array.
- To use the enhanced `for` statement to iterate through arrays.
- To pass arrays to methods.
- To declare and manipulate multidimensional arrays.

Introduction

- Arrays
  - Data structures
  - Related data items of same type
  - Remain same size once created
    - Fixed-length entries
  - Group of variables
    - Have same type



Example: 12-element array.

Arrays (Cont.)

- Index
  - Also called subscript
  - Position number in square brackets
  - Must be positive integer or integer expression
  - First element has index zero

```
a = 5;
b = 6;
c[ a + b ] += 2;
```

  - Adds `2` to `c[ 11 ]`

- Examine 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

Declaring and Creating Arrays

- Declaring and Creating arrays
  - Arrays are objects that occupy memory
  - Created dynamically with keyword new

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

    - Equivalent to

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

    - Can also be written as: int [ ] c = new int [12];

  - We can create arrays of objects too

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

Note: The **java**.lang.**String** class is used to create **string** object.

Common Programming Error

- Using a value of type `long` as an array index results in a compilation error. An index must be an `int` value or a value of a type that can be promoted to int—namely, `byte`, `short` or `char`, but not `long`.

- Declaring multiple array variables in a single declaration can lead to subtle errors. Consider the declaration `int[] a, b, c;`. If `a, b` and `c` should be declared as array variables, then this declaration is correct—placing square brackets directly following the type indicates that all the identifiers in the declaration are array variables. However, if only `a` is intended to be an array variable, and `b` and `c` are intended to be individual int variables, then this declaration is incorrect—the declaration `int a[], b, c;` would achieve the desired result.

Examples Using Arrays

- Creating and initializing an array
  - Declare array
  - Create array
  - Initialize array elements

```java
// Fig. 7.2: InitArray.java
// Creating an array.

public class InitArray
{
   public static void main( String [] args )
   {
      int array[]; // declare array named array

      array = new int[ 10 ]; // create the space for array

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

Declare `array` as an array of `int`s

Create 10 `int`s for `array`; each `int` is initialized to 0 by default

`array.length` returns length of `array`

```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```

Each `int` is initialized to 0 by default

`array[counter]` returns `int` associated with index in `array`

Examples Using Arrays (Cont.)

- 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`

```java
// Fig. 7.3: InitArray.java
// Initializing the elements of an array with an array initializer

public class InitArray
{
   public static void main( String [] args )
   {
      // initializer list specifies the value for each element
      int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

Declare `array` as an array of `int`s

Compiler uses initializer list to allocate array

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

Examples Using Arrays (Cont.)

- Calculating a value to store in each array element
  - Initialize elements of 10-element array to even integers

```java
// Fig. 7.4: InitArray.java
// Calculating values to be placed into elements of an array.

public class InitArray
{
   public static void main( String [] args )
   {
      final int ARRAY_LENGTH = 10; // declare constant
      int array[] = new int[ ARRAY_LENGTH ]; // create array

      // calculate value for each array element
      for ( int counter = 0; counter < array.length; counter++ )
         array[ counter ] = 2 + 2 * counter;

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

final for a variable indicates a constant variable

```
Index   Value
    0       2
    1       4
    2       6
    3       8
    4      10
    5      12
    6      14
    7      16
    8      18
    9      20
```

Common Programming Error

- Assigning a value to a constant after the variable has been initialized is a compilation error.
- Attempting to use a constant before it is initialized is a compilation error.

Examples Using Arrays (Cont.)

- Summing the elements of an array
  - Array elements can represent a series of values
    - We can sum these values

```java
// Fig. 7.5: SumArray.java
// Computing the sum of the elements of an array.

public class SumArray
{
   public static void main( String [] args )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int counter = 0; counter < array.length; counter++ )
         total += array[ counter ];

      System.out.printf( "Total of array elements: %d\n",   total );
   } // end main
} // end class SumArray
```

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

Examples Using Arrays (Cont.)

- Using arrays to analyze survey results
  - 40 students rate the quality of food
    - 1–10 Rating scale: 1 means awful, 10 means excellent
  - Place 40 responses in array of integers
  - Summarize results

```java
// Fig. 7.8: StudentPoll.java
// Poll analysis program.

public class StudentPoll
{
   public static void main( String [] args )
   {
      // array of survey responses
      int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3, 8, 6,
         10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6,
         4, 8, 6, 8, 10 };
      int frequency[] = new int[ 11 ]; // array of frequency counters

      // for each answer, select responses element and use that value
      // as frequency index to determine element to increment
      for ( int answer = 0; answer < responses.length; answer++ )
         ++frequency[ responses[ answer ] ];

      System.out.printf( "%s%10s", "Rating", "Frequency" );

      // output each array element's value
      for ( int rating = 1; rating < frequency.length; rating++ )
         System.out.printf( "%d%10d", rating, frequency[ rating ] );
   } // end main
} // end class StudentPoll
```

Declare `responses` as array to store 40 responses

Declare `frequency` as array of 11 `int` and ignore the first element

For each response, increment `frequency` values at index associated with that response

Output:

```
Rating Frequency
     1         2
     2         2
     3         2
     4         2
     5         5
     6        11
     7         5
     8         7
     9         1
    10         3
```

Error-Prevention Tip

- When writing code to loop through an array, ensure that the array index is always greater than or equal to 0 and less than the length of the array. The loop-continuation condition should prevent the accessing of elements outside this range.

Enhanced `for` Statement

Enhanced `for` statement:

- Iterates through elements of an array or a collection without using a counter

Syntax

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

```java
// Fig. 7.12: EnhancedForTest.java
// Using enhanced for statement to total integers in an array.

public class EnhancedForTest
{
   public static void main( String [] args )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int number : array )
         total += number;

      System.out.printf( "Total of array elements: %d\n", total );
   } // end main
} // 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`

Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - Array `hourlyTemperatures` is declared as

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

    - The method call

```
modifyArray( hourlyTemperatures );
```

    - Passes array `hourlyTemperatures` to method `modifyArray`

```java
// Fig. 7.13: PassArray.java
// Passing arrays and individual array elements to methods.

public class PassArray
{
   // main creates array and calls modifyArray and modifyE
   public static void main( String [] args )
   {
      int array[] = { 1, 2, 3, 4, 5 };

      System.out.println(
         "Effects of passing reference to entire array:\n" +
         "The values of the original array are:" );

      // output original array elements
      for ( int value : array )
         System.out.printf( "   %d", value );

      modifyArray( array ); // pass array reference
      System.out.println( "\n\nThe values of the modified array are:" );

      // output modified array elements
      for ( int value : array )
         System.out.printf( "   %d", value );

      System.out.printf(
         "\n\nEffects of passing array element value:\n" +
         "array[3] before modifyElement: %d\n", array[ 3 ] );
```

Declare 5-`int` array with initializer list

Pass entire array to method `modifyArray`

```java

      modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
      System.out.printf(
         "array[3] after modifyElement: %d\n", array[ 3 ] );
   } // end main

   // multiply each element of an array by 2
   public static void modifyArray( int array2[] )
   {
      for ( int counter = 0; counter < array2.length; counter++ )
         array2[ counter ] *= 2;
   } // end method modifyArray

   // multiply argument by 2
   public static void modifyElement( int element )
   {
      element *= 2;
      System.out.printf(
         "Value of element in modifyElement: %d\n", element );
   } // end method modifyElement
} // end class PassArray
```

Pass array element `array[3]` to method `modifyElement`

Method `modifyArray` manipulates the array directly

Method `modifyElement` manipulates a primitive's copy

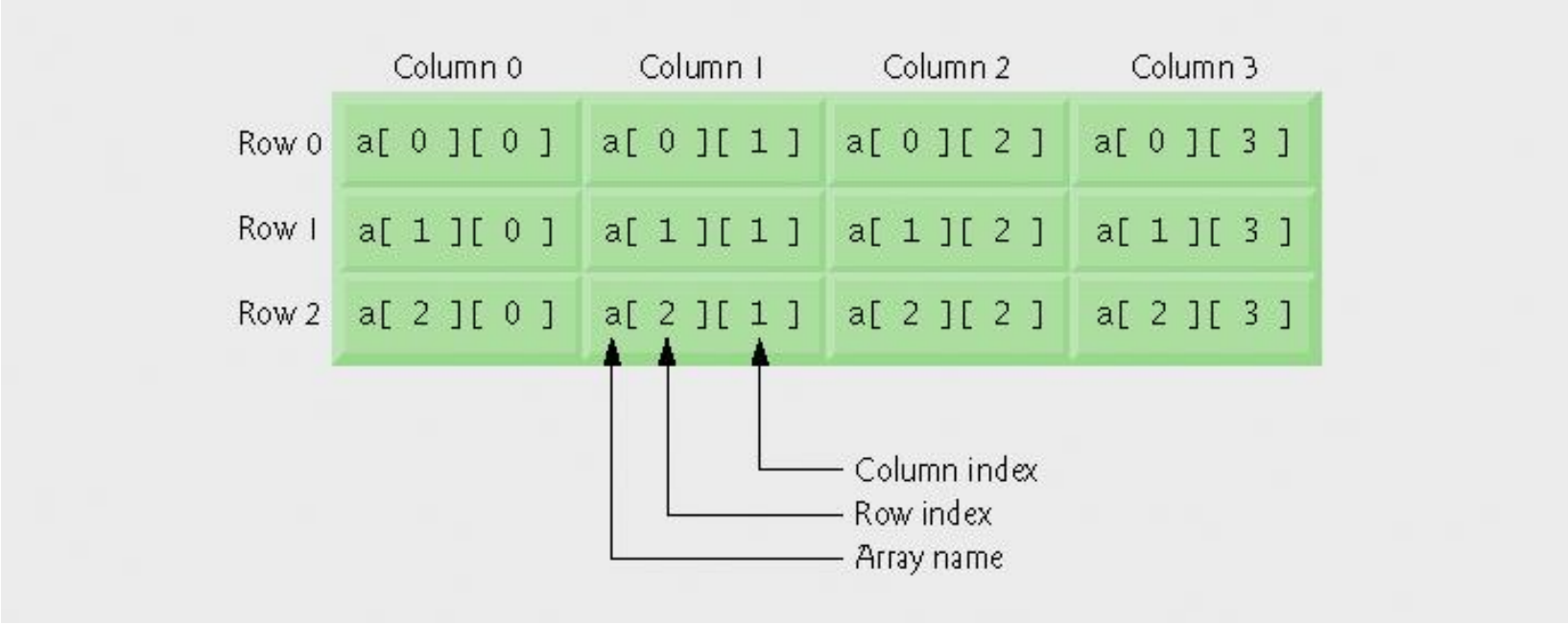
```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Passing Arrays to Methods (Cont.)

- Notes on 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 is passed-by-reference
        - In java, an object is a class instance or an array.
        - Therefore, arrays are objects and they are passed by reference

Multidimensional Arrays

- Multidimensional arrays
  - Tables with rows and columns
    - Two-dimensional array
    - m-by-n array



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
```

```java
// Fig. 7.17: InitArray.java
// Initializing two-dimensional arrays.

public class InitArray
{
   // create and output two-dimensional arrays
   public static void main( String [] args )
   {
      int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
      int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };

      System.out.println( "Values in array1 by row are" );
      outputArray( array1 ); // displays array1 by row

      System.out.println( "\nValues in array2 by row are" );
      outputArray( array2 ); // displays array2 by row
   } // end main

```

Use nested array initializers to initialize `array1`

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

```java
   // output rows and columns of a two-dimensional array
   public static void outputArray( int array[][] )
   {
      // loop through array's rows
      for ( int row = 0; row < array.length; row++ )
      {
         // loop through columns of current row
         for ( int column = 0; column < array[ row ].length; column++ )
            System.out.printf( "%d  ", array[ row ][ column ] );

         System.out.println(); // start new line of output
      } // end outer for
   } // end method outputArray
} // 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
```

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;
```