CMPE 101
Object Oriented Programming

İstanbul
Bilgi University

Dr. Emel Küpçü

Department of Computer Engineering

İstanbul Bilgi University

Week-6: Programming with Arrays

7.2 Arrays

- Array
  - Group of variables (called elements) containing values of the same type.
  - Arrays are objects so they are reference types.
  - Elements can be either primitive or reference types.
- Access a specific element in an array using its index
  - Use the element's index.
  - Array-access expression—the name of the array followed by the index of the particular element in square brackets, [ ].
- The first element in every array has index zero.
- The highest index in an array is one less than the number of elements in the array.
- Array names follow the same conventions as other variable names.

Arrays (Cont.)

- An index must be a nonnegative integer.
- An indexed array name is an array-access expression.
  - Can be used on the left side of an assignment to place a new value into an array element.
- Every array object knows its own length and stores it in a `length` instance variable.
  - `length` cannot be changed because it's a `final` variable.

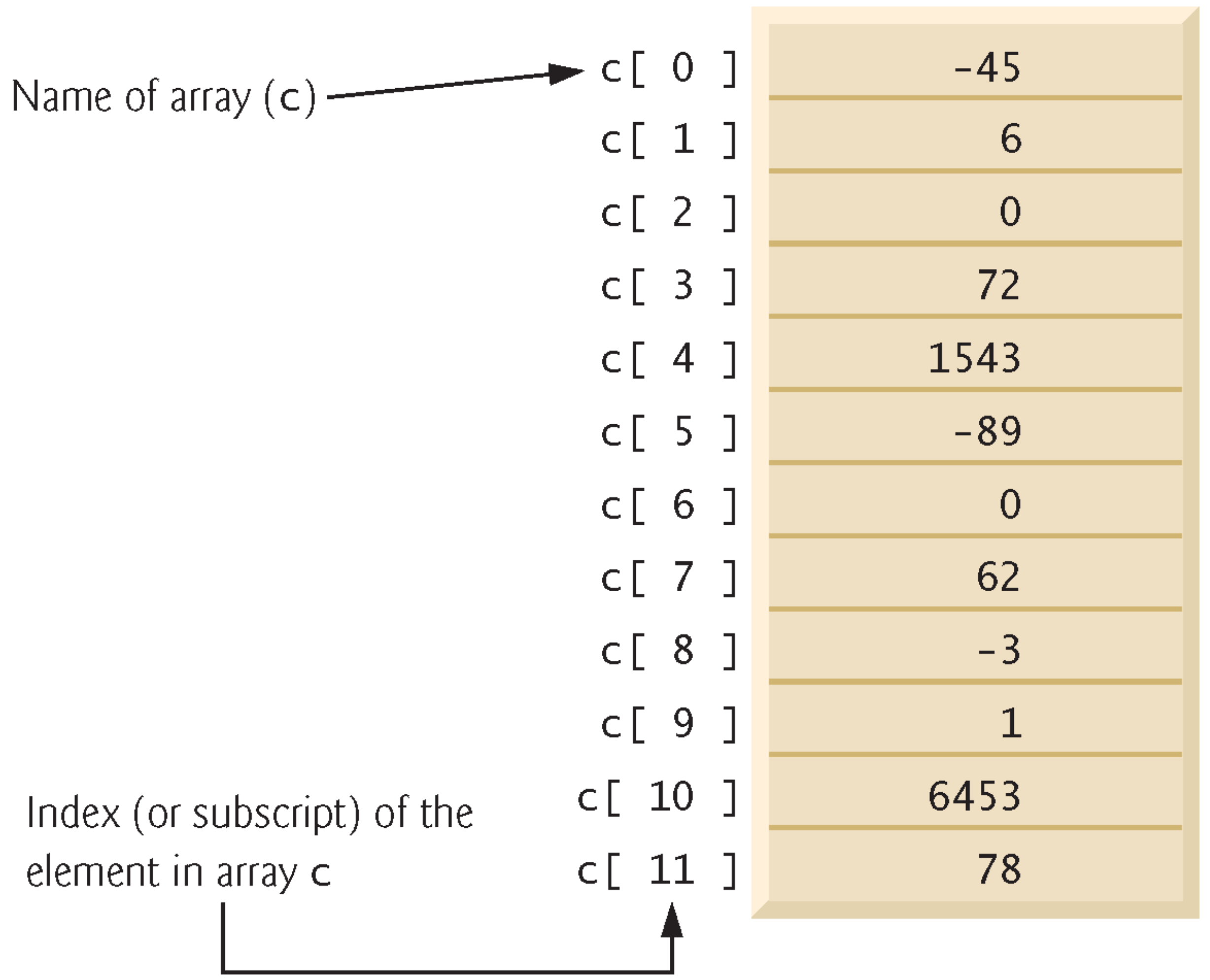

Fig. 7.1 | A 12-element array.

7.3 Declaring and Creating Arrays

- Array objects
  - Created with keyword **new**.
  - You specify the element type and the number of elements in an array-creation expression, which returns a reference that can be stored in an array variable.
- Declaration and array-creation expression for an array of 12 `int` elements

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

- Can be performed in two steps as follows:

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

7.3 Declaring and Creating Arrays (Cont.)

- In a declaration, *square brackets* following a type indicate that a variable will refer to an array (i.e., store an array *reference*).
- When an array is created, each element of the array receives a default value
  - Zero for the numeric primitive-type elements, `false` for `boolean` elements and `null` for references.

Common Programming Error 7.2

In an array declaration, specifying the number of elements in the square brackets of the declaration (e.g., `int[12] c;`) is a syntax error.

7. Declaring and Creating Arrays (Cont.)

- When the element type and the square brackets are combined at the beginning of the declaration, all the identifiers in the declaration are array variables.
  - `int[] numbers, values; // Both are array variables`

- For readability, declare only one variable per declaration.

  -
    ```
    int[] numbers;
    int[] values;
    ```

Common Programming Error 7.3

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.

7.3 Declaring and Creating Arrays (Cont.)

- Every element of a primitive-type array contains a value of the array’s declared element type.
  - Every element of an `int` array is an `int` value.
- Every element of a reference-type array is a reference to an object of the array’s declared element type.
  - Every element of a `String` array is a reference to a `String` object.

Creating and Initializing an Array

```java
// Fig. 7.2: InitArray.java
// Initializing the elements of an array to default values of zero.

public class InitArray {
   public static void main(String[] args) {
      // declare variable array and initialize it with an array object
      int[] array = new int[10]; // create the array object

      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]);
      }
   }
}
```

Fig. 7.2 | Initializing the elements of an array to default values of zero. (Part 1 of 2.)

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

Fig. 7.2 | Initializing the elements of an array to default values of zero. (Part 2 of 2.)

7.4.2 Using an Array Initializer

- Array initializer
  - A comma-separated list of expressions (called an initializer list) enclosed in braces.
  - Used to create an array and initialize its elements.
  - Array length is determined by the number of elements in the initializer list.

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

    - Creates a five-element array with index values `0`–`4`.
- Compiler counts the number of initializers in the list to determine the size of the array
  - Sets up the appropriate new operation "behind the scenes."

```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 initial 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]);
      }
   }
}
```

Fig. 7.3 | Initializing the elements of an array with an array initializer. (Part 1 of 2.)

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

Fig. 7.3 | Initializing the elements of an array with an array initializer. (Part 2 of 2.)

Calculating the Values to Store in an Array

```java
// Fig. 7.4: InitArray.java
// Calculating the values to be placed into the 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]);
      }
   }
}
```

Fig. 7.4 | Calculating the values to be placed into the elements of an array. (Part 1 of 2.)

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

Fig. 7.4 | Calculating the values to be placed into the elements of an array. (Part 2 of 2.)

Examples Using Arrays (Cont.)

- `final` variables must be initialized before they are used and cannot be modified thereafter.
- An attempt to modify a `final` variable **after it's initialized** causes a compilation error
- An attempt to access the value of a `final` variable before it's initialized causes a compilation error

Summing the Elements of an Array

```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);
   }
}
```

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

Fig. 7.5 | Computing the sum of the elements of an array.

7.7 Enhanced for Statement

- Enhanced `for` statement
  - Iterates through the elements of an array without using a counter.
  - Avoids the possibility of “stepping outside” the array.
- Syntax:

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

  where *parameter* has a type and an identifier and *arrayName* is the array through which to iterate.
- Parameter type must be consistent with the array’s element type.
- The enhanced `for` statement simplifies the code for iterating through an array.

```java
// Fig. 7.12: EnhancedForTest.java
// Using the 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);
   }
}
```

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

Fig. 7.12 | Using the enhanced `for` statement to total integers in an array.

Enhanced for Statement (Cont.)

- The enhanced **for** statement can be used *only* to obtain array elements
  - It *cannot* be used to *modify* elements.
  - To modify elements, use the traditional counter-controlled for statement.
- Can be used in place of the counter-controlled for statement if you don't need to access the index of the element.

Output:
```
10
20
30
40
```

```java
public class ModifyArray {
   public static void main(String[] args) {
      int[] numbers = {10, 20, 30, 40};

      // Attempting to modify elements (DOESN'T WORK)
      for (int num : numbers) {
         num = num * 2;  // This changes only 'num', NOT the array
      }

      // Printing array to check if elements were modified
      for (int num : numbers) {
         System.out.println(num);
      }
   }
}
```

```java
// Using a traditional for loop to modify elements
for (int i = 0; i < numbers.length; i++) {
   numbers[i] *= 2;  // Modifies the actual array elements
}
```

Output:
```
20
40
60
80
```

7.8 Passing Arrays to Methods

- To pass an array argument to a method, specify the name of the array without any brackets.
  - Since every array object "knows" its own length, we need not pass the array length as an additional argument.
- When an argument to a method is an entire array or an individual array element of a reference type, the called method receives a copy of the reference.
- When an argument to a method is an individual array element of a primitive type, the called method receives a copy of the element's value.
  - Such primitive values are called scalars or scalar quantities.

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

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

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

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

```

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 1 of 4.)

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

        // 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]);

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

```

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 2 of 4.)

```java
   // 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;
      }
   }

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

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 3 of 4.)

```
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
```

Fig. 7.13 | Passing arrays and individual array elements to methods. (Part 4 of 4.)

Questions?