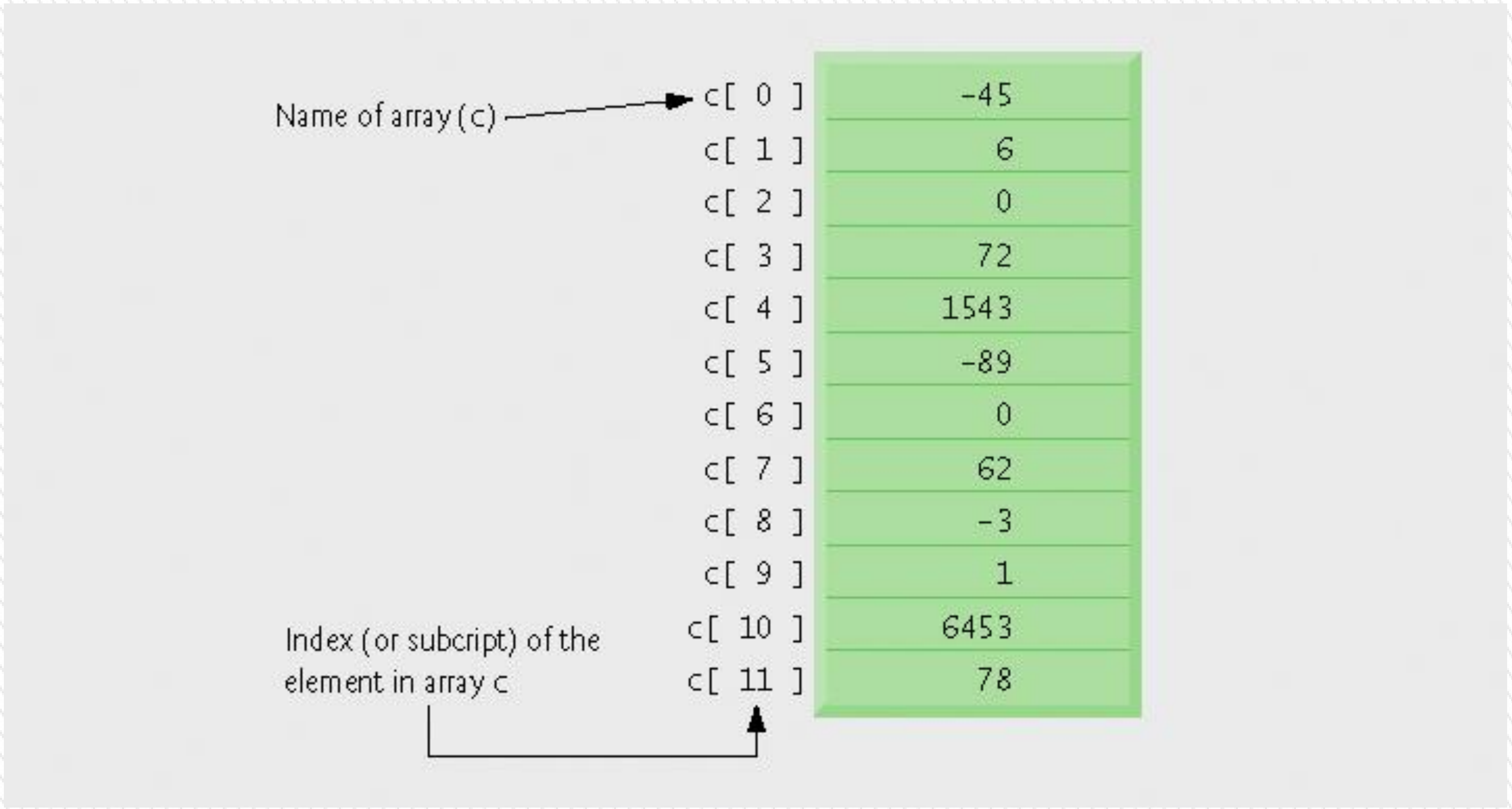
CS212: Object-Oriented Programming

Arrays in Java

Hirra Anwar

INTRODUCTION - ARRAYS

- It’s a Data structure
- A Group of variables having the same data type
- Remain same size once created.
  - Fixed-length entries



Arrays Index

- 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 ]
```

Declaring and Creating Arrays (in Java)

- 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
```

  - We can create arrays of objects too

```
String b[] = new String[ 100 ];
String[] b = new String[ 100 ], x = new
  String[ 27 ];
```

Important Notes

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

- 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.

```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 a                         lizer.

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
```

Important Notes

- Constant variables also are called named constants or read-only variables. Such variables often make programs more readable than programs that use literal values (e.g., 10)—a named constant such as ARRAY_LENGTH clearly indicates its purpose, whereas a literal value could have different meanings based on the context in which it is used.
- 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.

Enhanced For Loop Statement

» Enhanced for statement

- New feature of J2SE 5.0
- Allows iterates through elements of an array or a collection without using a counter
  - Avoiding the possibility of “stepping outside” the array
- Syntax

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

- where parameter has a type and an identifier.
- type of the parameter must be consistent with the type of the elements in the array

Enhanced For Loop 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
```

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

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

Enhanced For Loop Statement

- Lines 12-13 are equivalent to

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

- Usage Limitations
  - Can access array elements
  - Cannot modify array elements
  - Cannot access the counter indicating the index

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 modifyElement
   public static void main( String args[] )
   {
      int array[] = { 1, 2, 3, 4, 5 };

      System.out.println(
         "Effects of passing reference to entire
         "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
```

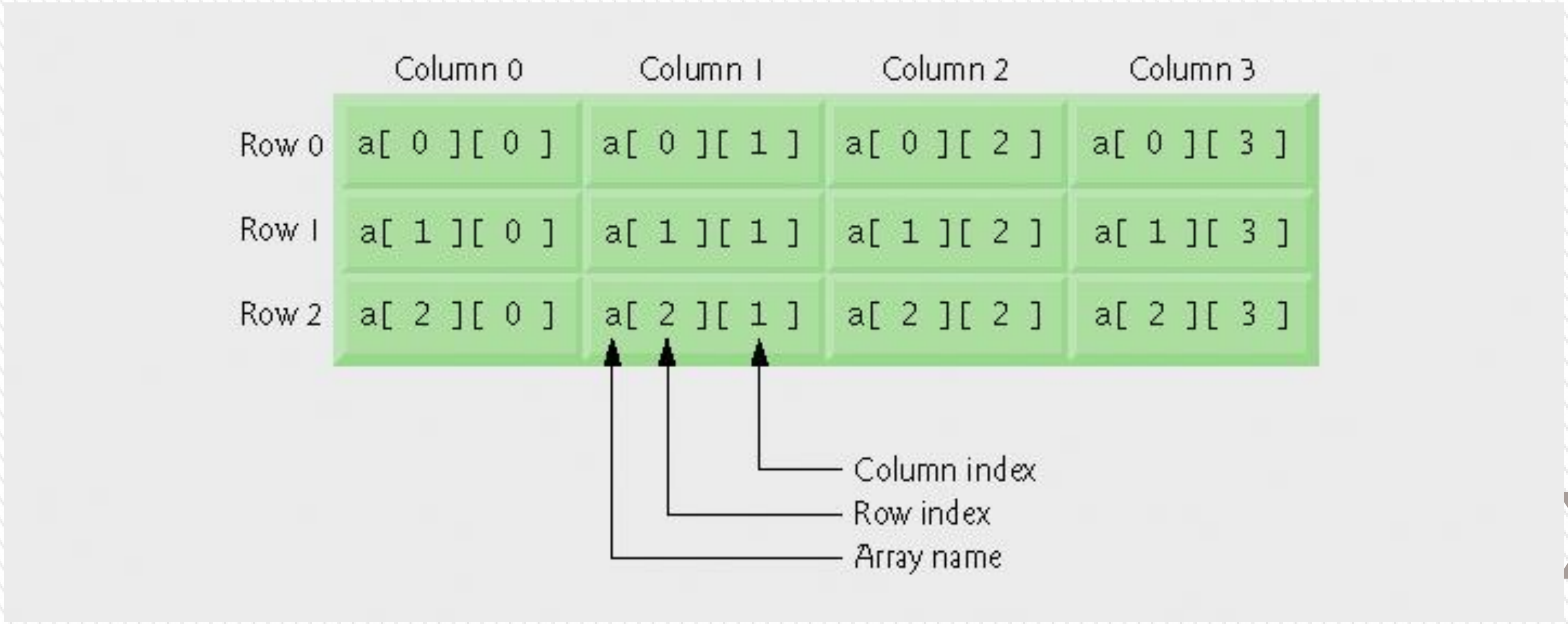
Passing Arrays to Methods (Cont.)

» Two ways to pass arguments to methods

- **Pass-by-value**
  - Copy of argument's value is passed to called method
    - In Java, every primitive is pass-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
  - In Java, every object is pass-by-reference
    - In Java, arrays are objects
    - Therefore, arrays are passed to methods by reference

Multidimensional Arrays

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



MULTIDIMENSIONAL ARRAYS

» 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

- Lengths of rows in array are not required to be the same!

Multidimensional Arrays

» Creating two-dimensional arrays with array-creation expressions

- ❖ Can be created dynamically
  - 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 cols for
 row 0
 b[ 1 ] = new int[ 3 ]; // create 3 cols for
 row 1
```

Variable Length Arguments

- Variable-length argument lists
  - New feature in J2SE 5.0
  - Unspecified number of arguments
  - Use ellipsis (…) in method's parameter list
    - Can occur only once in parameter list
    - Must be placed at the end of parameter list
  - Array whose elements are all of the same type

Variable Length Arguments

```java
// Fig. 7.20: VarargsTest.java
// Using variable-length argument lists.

public class VarargsTest
{
   // calculate average
   public static double average( double... numbers )
   {
      double total = 0.0; // initialize total

      // calculate total using the enhanced for s
      for ( double d : numbers )
         total += d;

      return total / numbers.length;
   } // end method average

   public static void main( String args[] )
   {
      double d1 = 10.0;
      double d2 = 20.0;
      double d3 = 30.0;
      double d4 = 40.0;

```

Method `average` receives a variable length sequence of `double`s

Calculate the total of the `double`s in the array

Access `numbers.length` to obtain the size of the `numbers` array

Variable Length Arguments

```java
      System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n",
         d1, d2, d3, d4 );

      System.out.printf( "Average of d1 and d2 is %.1f\n",
         average( d1, d2 ) );
      System.out.printf( "Average of d1, d2 and d3 is %.1f\n",
         average( d1, d2, d3 ) );
      System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
         average( d1, d2, d3, d4 ) );
   } // end main
} // end class VarargsTest
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

Invoke method average with two arguments

Invoke method average with three arguments

Invoke method average with four arguments

Common Programming Error

» Placing an ellipsis in the middle of a method parameter list is a syntax error. An ellipsis may be placed only at the end of the parameter list.

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 ]`

```java
// Fig. 7.21: InitArray.java
// Using command-line arguments to initialize an array.

public class InitArray
{
   public static void main( String args[] )
   {
      // check number of command-line arguments
      if ( args.length != 3 )
         System.out.println(
            "Error: Please re-enter the entire command, including\n" +
            "an array size, initial value and increment." );
      else
      {
         // get array size from first command-line argument
         int arrayLength = Integer.parseInt( args[ 0 ] );
         int array[] = new int[ arrayLength ]; // create array

         // get initial value and increment from command-line argument
         int initialValue = Integer.parseInt( args[ 1 ] );
         int increment = Integer.parseInt( args[ 2 ] );

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

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

```

Array `args` stores command-line arguments

Check number of arguments passed in from the command line

Obtain first command-line argument

Obtain second and third command-line arguments

Calculate the value for each array element based on command-line arguments

```
            // display array index and value
            for ( int counter = 0; counter < array.length; counter++ )
               System.out.printf( "%5d%8d\n", counter, array[ counter ] );
         } // end else
      } // end main
   } // end class InitArray
```

```
java InitArray
Error: Please re-enter the entire command, including
an array size, initial value and increment.
```

Missing command-line arguments

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

Three command-line arguments are 5, 0 and 4

```
java InitArray 10 1 2
Index   Value
    0       1
    1       3
    2       5
    3       7
    4       9
    5      11
    6      13
    7      15
    8      17
    9      19
```

Three command-line arguments are 10, 1 and 2

Questions/Answers & Discussion