Arrays
7.1 Introduction

• Arrays
  – Data structures
  – Related data items of same type
  – Remain same size once created
    • Fixed-length entries
7.2 Arrays

• Array
  – Group of variables
    • Have same type
  – Reference type
**Fig. 7.1** | A 12-element array.

<table>
<thead>
<tr>
<th>Index (or subscript) of the element in array c</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c[0]$</td>
<td>-45</td>
</tr>
<tr>
<td>$c[1]$</td>
<td>6</td>
</tr>
<tr>
<td>$c[2]$</td>
<td>0</td>
</tr>
<tr>
<td>$c[3]$</td>
<td>72</td>
</tr>
<tr>
<td>$c[4]$</td>
<td>1543</td>
</tr>
<tr>
<td>$c[5]$</td>
<td>-89</td>
</tr>
<tr>
<td>$c[6]$</td>
<td>0</td>
</tr>
<tr>
<td>$c[7]$</td>
<td>62</td>
</tr>
<tr>
<td>$c[8]$</td>
<td>-3</td>
</tr>
<tr>
<td>$c[9]$</td>
<td>1</td>
</tr>
<tr>
<td>$c[10]$</td>
<td>6453</td>
</tr>
</tbody>
</table>
7.2 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 \( c[11] \)
Common Programming Error 7.1

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.
• Examine array $c$
  – $c$ is the array name
  – $c$.length accesses array $c$’s length
  – $c$ has 12 elements ($c[0], c[1], \ldots, c[11]$)
    • The value of $c[0]$ is $-45$
7.3 Declaring and Creating Arrays

• Declaring and Creating arrays
  – Arrays are objects that occupy memory
  – Created dynamically with keyword `new`
    
    ```java
    int c[] = new int[12];
    ```
    – Equivalent to
    
    ```java
    int c[]; // declare array variable
    c = new int[12]; // create array
    ```

• We can create arrays of objects too
  
  ```java
  String b[] = new String[100];
  ```
Common Programming Error 7.2

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.
7.4 Examples Using Arrays

- Declaring arrays
- Creating arrays
- Initializing arrays
- Manipulating array elements
7.4 Examples Using Arrays

• Creating and initializing an array
  – Declare array
  – Create array
  – Initialize array elements
// 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
", "Index", "Value" ); // column headings

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

Index   Value
0       0
1       0
2       0
3       0
4       0
5       0
6       0
7       0
8       0
9       0
7.4 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`
// 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
        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

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>32</td>
</tr>
<tr>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>95</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
</tr>
<tr>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td>9</td>
<td>37</td>
</tr>
</tbody>
</table>
7.4 Examples Using Arrays (Cont.)

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

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>16</td>
</tr>
<tr>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
</tr>
</tbody>
</table>
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.
7.4 Examples Using Arrays (Cont.)

• **Summing the elements of an array**
  – Array elements can represent a series of values
    • We can sum these values
// 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
7.4 Examples Using Arrays (Cont.)

• Using the elements of an array as counters
  – Use a series of counter variables to summarize data
// Fig. 7.7: RollDie.java
// Roll a six-sided die 6000 times.
import java.util.Random;

public class RollDie
{
    public static void main( String args[] )
    {
        Random randomNumbers = new Random(); // random number generator
        int frequency[] = new int[7]; // array of frequency counters

        // roll die 6000 times; use die value as frequency index
        for ( int roll = 1; roll <= 6000; roll++ )
        {
            ++frequency[1 + randomNumbers.nextInt(6)];
        }

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

        // output each array element's value
        for ( int face = 1; face < frequency.length; face++ )
        {
            System.out.printf( "%4d%10d
", face, frequency[face] );
        }
    } // end main
} // end class RollDie

<table>
<thead>
<tr>
<th>Face</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>988</td>
</tr>
<tr>
<td>2</td>
<td>963</td>
</tr>
<tr>
<td>3</td>
<td>1018</td>
</tr>
<tr>
<td>4</td>
<td>1041</td>
</tr>
<tr>
<td>5</td>
<td>978</td>
</tr>
<tr>
<td>6</td>
<td>1012</td>
</tr>
</tbody>
</table>
Error-Prevention Tip 7.1

An exception indicates that an error has occurred in a program. A programmer often can write code to recover from an exception and continue program execution, rather than abnormally terminating the program. When a program attempts to access an element outside the array bounds, an \texttt{ArrayIndexOutOfBoundsException} occurs. Exception handling is discussed in Chapter 13.
7.6 Enhanced for Statement

• Enhanced for statement
  – New feature of J2SE 5.0
  – Allows iterates through elements of an array or a collection without using a counter
  – Syntax
    ```
    for ( parameter : arrayName )
    statement
    ```
// 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.
7.6 Enhanced for Statement (Cont.)

- **Lines 12-13 are equivalent to**
  ```java
  for ( int counter = 0; counter < array.length; counter++ )
  total += array[ counter ];
  ```

- **Usage**
  - Can access array elements
  - Cannot modify array elements
  - Cannot access the counter indicating the index
7.7 Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - Array `hourlyTemperatures` is declared as
      ```java
      int hourlyTemperatures = new int[24];
      ```
    - The method call
      ```java
      modifyArray(hourlyTemperatures);
      ```
    - Passes array `hourlyTemperatures` to method `modifyArray`
public class PassArray {
    // main creates array and calls modifyArray and modifyElement
    public static void main( String args[] )
    {
        int array[] = { 1, 2, 3, 4, 5 };  // Declare 5-int array with initializer list
        System.out.println( "Effects of passing reference to entire array:
                             " + "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:
   array[3] before modifyElement: %d\n", array[3] );
    }
}
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
• 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
      – 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
7.9 Multidimensional Arrays

- **Multidimensional arrays**
  - Tables with rows and columns
    - Two-dimensional array
    - m-by-n array
Fig. 7.16 | Two-dimensional array with three rows and four columns.
7.9 Multidimensional Arrays (Cont.)

• Arrays of one-dimensional array
  – Declaring two-dimensional array \( b[2][2] \)
    \[
    \text{int } b[][] = \{ \{ 1, 2 \}, \{ 3, 4 \} \};
    \]
  – 1 and 2 initialize \( b[0][0] \) and \( b[0][1] \)
  – 3 and 4 initialize \( b[1][0] \) and \( b[1][1] \)
    \[
    \text{int } b[][] = \{ \{ 1, 2 \}, \{ 3, 4, 5 \} \};
    \]
  – row 0 contains elements 1 and 2
  – row 1 contains elements 3, 4 and 5
7.9 Multidimensional Arrays (Cont.)

• Creating two-dimensional arrays with array-creation expressions
  – Can be created dynamically
    • 3-by-4 array
      ```java
      int b[][];
      b = new int[3][4];
      ```
    • Rows can have different number of columns
      ```java
      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
      ```
7.9 Multidimensional Arrays (Cont.)

- **Common multidimensional-array manipulations performed with `for` statements**
  - Many common array manipulations use `for` statements
  
  **E.g.,**

  ```java
  for ( int column = 0; column < a[2].length; column++ )
    a[2][column] = 0;
  ```
7.10 Case Study: Class GradeBook Using a Two-Dimensional Array

• Class GradeBook
  – One-dimensional array
    • Store student grades on a single exam
  – Two-dimensional array
    • Store grades for a single student and for the class as a whole
Loop through rows of grades to find the lowest grade of any student.
7.11 Variable-Length Argument Lists

- 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
// 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 statement
        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;
    }
} // end class VarargsTest

Method average receives a variable length sequence of doubles
Calculate the total of the doubles in the array
Access numbers.length to obtain the size of the numbers array
25    System.out.printf("d1 = %1f\nd2 = %1f\nd3 = %1f\nd4 = %1f\n", d1, d2, d3, d4);
26
27    System.out.printf("Average of d1 and d2 is %1f\n", average( d1, d2 ) );
28    System.out.printf("Average of d1, d2 and d3 is %1f\n", average( d1, d2, d3 ) );
29    System.out.printf("Average of d1, d2, d3 and d4 is %1f\n", average( d1, d2, d3, d4 ) );
30    } // end main
31 } // end class VarargsTest

// end class VarargsTest

---

Program output:

```
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
```
Common Programming Error 7.6

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.
7.12 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 " + "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 arguments
            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( "%8s
", "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
```java
// 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
```

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

```
Three command-line arguments are 5, 0 and 4
```

```
Three command-line arguments are 10, 1 and 2
```
7.13 (Optional) GUI and Graphics Case Study: Drawing Arcs

• Draw rainbow
  – Use arrays
  – Use repetition statement
  – Use Graphics method fill Arc
// Fig. 7.22: DrawRainbow.java
// Demonstrates using colors in an array.
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawRainbow extends JPanel {
    final Color VIOLET = new Color( 128, 0, 128 );
    final Color INDIGO = new Color( 75, 0, 130 );

    private Color colors[] = {
        Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
        Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED
    };

    public DrawRainbow() {
        setBackground( Color.WHITE ); // set the background to white
    } // end DrawRainbow constructor

    public void paintComponent( Graphics g ) {
        super.paintComponent( g );
        int radius = 20; // radius of an arch
// draw the rainbow near the bottom center
int centerX = getWidth() / 2;
int centerY = getHeight() - 10;

// draws filled arcs starting with the outermost
for (int counter = colors.length; counter > 0; counter-- )
{
   // set the color for the current arc
g.setColor( colors[ counter - 1 ] );

   // fill the arc from 0 to 180 degrees
g.fillArc( centerX - counter * radius,
            centerY - counter * radius,
            counter * radius * 2, counter * radius * 2, 0, 180 );
}

} // end method paintComponent

} // end class DrawRainbow

// draw a filled semicircle
package DrawRainbowTest;

class DrawRainbowTest {
    public static void main(String[] args) {
        DrawRainbow panel = new DrawRainbow();
        JFrame application = new JFrame();
        application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        application.add(panel);
        application.setSize(400, 250);
        application.setVisible(true);
    }
}
Fig. 7.24 | Drawing a spiral using `drawLine` (left) and `drawArc` (right).
7.14 (Optional) Software Engineering Case Study: Collaboration Among Objects

• Collaborations
  – When objects communicate to accomplish task
    • Accomplished by invoking operations (methods)
  – One object sends a message to another object
• Identifying the collaborations in a system
  – Read requirements document to find
    • What ATM should do to authenticate a user
    • What ATM should do to perform transactions
  – For each action, decide
    • Which objects must interact
      – Sending object
      – Receiving object
<table>
<thead>
<tr>
<th>An object of class...</th>
<th>sends the message...</th>
<th>to an object of class...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>authenticateUser</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>Execute</td>
<td>Screen</td>
</tr>
<tr>
<td>BankDatabase</td>
<td>getAvailableBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>getTotalBalance</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>displayMessage</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>CashDispenser</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>CashDispenser</td>
<td>CashDispenser</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>getAvailableBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>isSufficientCashAvail</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>debi t</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>dispenseCash</td>
<td>Screen</td>
</tr>
<tr>
<td>Deposit</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>isEnvelopeReceived</td>
<td>DepositSlot</td>
</tr>
<tr>
<td></td>
<td>Credit</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>Account</td>
<td>BankDatabase</td>
</tr>
</tbody>
</table>

**Fig. 7.25 | Collaborations in the ATM system.**
7.14 (Optional) Software Engineering Case Study (Cont.)

• Interaction Diagrams
  – Model interactions use UML
  – Communication diagrams
    • Also called collaboration diagrams
    • Emphasize which objects participate in collaborations
  – Sequence diagrams
    • Emphasize when messages are sent between objects
7.14 (Optional) Software Engineering Case Study (Cont.)

• Communication diagrams
  – Objects
    • Modeled as rectangles
    • Contain names in the form `objectName:className`
  – Objects are connected with solid lines
  – Messages are passed along these lines in the direction shown by arrows
  – Name of message appears next to the arrow
Fig. 7.26 | Communication diagram of the ATM executing a balance inquiry.
7.14 (Optional) Software Engineering Case Study (Cont.)

• **Sequence of messages in a communication diagram**
  – Appear to the left of a message name
  – Indicate the order in which the message is passed
  – Process in numerical order from least to greatest
Fig. 7.27 | Communication diagram for executing a balance inquiry.
7.14 (Optional) Software Engineering Case Study (Cont.)

• Sequence diagrams
  – Help model the timing of collaborations
  – Lifeline
    • Dotted line extending down from an object’s rectangle
      – Represents the progression of time
  – Activation
    • Thin vertical rectangle
      – Indicates that an object is executing
Fig. 7.28 | Sequence diagram that models a withdrawal executing.
Fig. 7.29 | Sequence diagram that models a Deposit executing.