Arrays
Now go, write it before them in a table, and note it in a book.
— Isaiah 30:8

To go beyond is as wrong as to fall short.
— Confucius

Begin at the beginning,… and go on till you come to the end: then stop.
— Lewis Carroll
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.
- To write methods that use variable-length argument lists.
- To read command-line arguments into a program.
<table>
<thead>
<tr>
<th>Section</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>7.2</td>
<td>Arrays</td>
</tr>
<tr>
<td>7.3</td>
<td>Declaring and Creating Arrays</td>
</tr>
<tr>
<td>7.4</td>
<td>Examples Using Arrays</td>
</tr>
<tr>
<td>7.5</td>
<td>Case Study: Card Shuffling and Dealing Simulation</td>
</tr>
<tr>
<td>7.6</td>
<td>Enhanced for Statement</td>
</tr>
<tr>
<td>7.7</td>
<td>Passing Arrays to Methods</td>
</tr>
<tr>
<td>7.8</td>
<td>Case Study: Class GradeBook Using an Array to Store Grades</td>
</tr>
<tr>
<td>7.9</td>
<td>Multidimensional Arrays</td>
</tr>
<tr>
<td>7.10</td>
<td>Case Study: Class GradeBook Using a Two-Dimensional Array</td>
</tr>
<tr>
<td>7.11</td>
<td>Variable-Length Argument Lists</td>
</tr>
<tr>
<td>7.12</td>
<td>Using Command-Line Arguments</td>
</tr>
<tr>
<td>7.13</td>
<td>(Optional) GUI and Graphics Case Study: Drawing Arcs</td>
</tr>
<tr>
<td>7.14</td>
<td>(Optional) Software Engineering Case Study: Collaboration Among Objects</td>
</tr>
<tr>
<td>7.15</td>
<td>Wrap-Up</td>
</tr>
</tbody>
</table>
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.
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

\[
\begin{align*}
  a &= 5; \\
  b &= 6; \\
\end{align*}
\]

• 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], ... 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.
Good Programming Practice 7.1

For readability, declare only one variable per declaration. Keep each declaration on a separate line, and include a comment describing the variable being declared.
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.
7.4 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

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
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 element
        int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

        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
7.4 Examples Using Arrays (Cont.)

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

// 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
", "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       2
1       4
2       6
3       8
4       10
5       12
6       14
7       16
8       18
9       20
Good Programming Practice 7.2

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.
Common Programming Error 7.4

Assigning a value to a constant after the variable has been initialized is a compilation error.
Common Programming Error 7.5

Attempting to use a constant before it is initialized is a compilation error.
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 bar charts to display array data graphically
  – Present data in graphical manner
    • E.g., bar chart
  – Examine the distribution of grades
public class BarChart {
    public static void main( String args[] )
    {
        int array[] = { 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
        System.out.println( "Grade distribution:" );
        // for each array element, output a bar of the chart
        for ( int counter = 0; counter < array.length; counter++ )
        {
            // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
            if ( counter == 10 )
                System.out.printf( "%5d: ", 100 );
            else
                System.out.printf( "%02d-%02d: ",
                                    counter * 10, counter * 10 + 9 );
            // print bar of asterisks
            for ( int stars = 0; stars < array[ counter ]; stars++ )
                System.out.print( "*" );
            System.out.println(); // start a new line of output
        } // end outer for
    } // end main
} // end class BarChart

// Fig. 7.6: BarChart.java
// Bar chart printing program.
public class BarChart
{
    public static void main( String args[] )
    {
        int array[] = { 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
        System.out.println( "Grade distribution:" );
        // for each array element, output a bar of the chart
        for ( int counter = 0; counter < array.length; counter++ )
        {
            // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
            if ( counter == 10 )
                System.out.printf( "%5d: ", 100 );
            else
                System.out.printf( "%02d-%02d: ",
                                    counter * 10, counter * 10 + 9 );
            // print bar of asterisks
            for ( int stars = 0; stars < array[ counter ]; stars++ )
                System.out.print( "*" );
            System.out.println(); // start a new line of output
        } // end outer for
    } // end main
} // end class BarChart

Outline

BarChart.java

(1 of 2)

Line 8
Declare array
with initializer
list

Line 19
Use the 0 flag
to display one-
digit grade with
a leading 0

Use the 0 flag to display one-
digit grade with a leading 0

For each array element, print
associated number of asterisks

© 1992-2007 Pearson Education, Inc. All rights reserved.
Grade distribution:
00–09: 
10–19: 
20–29: 
30–39: 
40–49: 
50–59: 
60–69: * 
70–79: ** 
80–89: **** 
90–99: ** 
100: *
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\n", "Face", "Frequency" );

        // output each array element's value
        for ( int face = 1; face < frequency.length; face++ )
        {
            System.out.printf( "%4d%10d\n", 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>
7.4 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
// 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, 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 ] );
    }
}

© 1992-2007 Pearson Education, Inc. All rights reserved.
<table>
<thead>
<tr>
<th>Rating</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

Outline

StudentPoll.java

(2 of 2)

Program output
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 `ArrayIndexOutOfBoundsException` occurs. Exception handling is discussed in Chapter 13.
Error-Prevention Tip 7.2

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.
7.5 Case Study: Card Shuffling and Dealing Simulation

• Program simulates card shuffling and dealing
  – Use random number generation
  – Use an array of reference type elements to represent cards
  – Three classes
    • Card
      – Represents a playing card
    • DeckOfCards
      – Represents a deck of 52 playing cards
    • DeckOfCardsTest
      – Demonstrates card shuffling and dealing
// Fig. 7.9: Card.java
// Card class represents a playing card.

public class Card {
    private String face; // face of card ("Ace", "Deuce", ...)
    private String suit; // suit of card ("Hearts", "Diamonds", ...)

    // two-argument constructor initializes card's face and suit
    public Card(String cardFace, String cardSuit) {
        face = cardFace; // initialize face of card
        suit = cardSuit; // initialize suit of card
    } // end two-argument Card constructor

    // return String representation of Card
    public String toString() {
        return face + " of " + suit;
    } // end method toString

} // end class Card

Return the string representation of a card
// Fig. 7.10: DeckOfCards.java
// DeckOfCards class represents a deck of playing cards.
import java.util.Random;

public class DeckOfCards {
    private Card deck[]; // array of Card objects
    private int currentCard; // index of next Card to be dealt
    private final int NUMBER_OF_CARDS = 52; // constant number of Cards
    private Random randomNumbers; // random number generator

    // constructor fills deck of Cards
    public DeckOfCards() {
        String suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };

        deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
        currentCard = 0; // set currentCard so first Card dealt is deck[ 0 ]
        randomNumbers = new Random(); // create random number generator

        // populate deck with Card objects
        for ( int count = 0; count < deck.length; count++ )
            deck[ count ] = new Card( faces[ count % 13 ], suits[ count / 13 ] );
    } // end DeckOfCards constructor
// shuffle deck of Cards with one-pass algorithm
public void shuffle()
{
    // after shuffling, dealing should start at deck[ 0 ] again
    currentCard = 0; // reinitialize currentCard

    // for each Card, pick another random Card and swap them
    for ( int first = 0; first < deck.length; first++ )
    {
        // select a random number between 0 and 51
        int second = randomNumbers.nextInt( NUMBER_OF_CARDS );

        // swap current Card with randomly selected Card
        Card temp = deck[ first ];
        deck[ first ] = deck[ second ];
        deck[ second ] = temp;
    } // end for
} // end method shuffle

// deal one Card
public Card dealCard()
{
    // determine whether Cards remain to be dealt
    if ( currentCard < deck.length )
        return deck[ currentCard++ ]; // return current Card in array
    else
        return null; // return null to indicate that all Cards were dealt
} // end method dealCard
// end class DeckOfCards
public class DeckOfCardsTest {
    // execute application
    public static void main( String args[] ) {
        DeckOfCards myDeckOfCards = new DeckOfCards();
        myDeckOfCards.shuffle(); // place Cards in random order
        // print all 52 Cards in the order in which they are dealt
        for ( int i = 0; i < 13; i++ ) {
            // deal and print 4 Cards
            System.out.printf( "%-20s%-20s%-20s%-20s
", myDeckOfCards.dealCard(), myDeckOfCards.dealCard(), myDeckOfCards.dealCard(), myDeckOfCards.dealCard() );
        } // end for
    } // end main
} // end class DeckOfCardsTest
<table>
<thead>
<tr>
<th>Six of Spades</th>
<th>Eight of Spades</th>
<th>Six of Clubs</th>
<th>Nine of Hearts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queen of Hearts</td>
<td>Seven of Clubs</td>
<td>Nine of Spades</td>
<td>King of Hearts</td>
</tr>
<tr>
<td>Three of Diamonds</td>
<td>Deuce of Clubs</td>
<td>Ace of Hearts</td>
<td>Ten of Spades</td>
</tr>
<tr>
<td>Four of Spades</td>
<td>Ace of Clubs</td>
<td>Seven of Diamonds</td>
<td>Four of Hearts</td>
</tr>
<tr>
<td>Three of Clubs</td>
<td>Deuce of Hearts</td>
<td>Five of Spades</td>
<td>Jack of Diamonds</td>
</tr>
<tr>
<td>King of Clubs</td>
<td>Ten of Hearts</td>
<td>Three of Hearts</td>
<td>Six of Diamonds</td>
</tr>
<tr>
<td>Queen of Clubs</td>
<td>Eight of Diamonds</td>
<td>Ten of Diamonds</td>
<td>Six of Hearts</td>
</tr>
<tr>
<td>Three of Spades</td>
<td>King of Diamonds</td>
<td>Seven of Hearts</td>
<td>Eight of Clubs</td>
</tr>
<tr>
<td>Ace of Spades</td>
<td>Four of Diamonds</td>
<td>Five of Hearts</td>
<td>Queen of Spades</td>
</tr>
<tr>
<td>Deuce of Spades</td>
<td>Eight of Hearts</td>
<td>Four of Clubs</td>
<td>Nine of Diamonds</td>
</tr>
<tr>
<td>Jack of Hearts</td>
<td>Seven of Spades</td>
<td>Five of Clubs</td>
<td>King of Spades</td>
</tr>
<tr>
<td>Ace of Diamonds</td>
<td>Queen of Diamonds</td>
<td>Jack of Spades</td>
<td>Jack of Clubs</td>
</tr>
</tbody>
</table>
7.6 Enhanced for Statement

• Enhanced for statement
  – Iterates through elements of an array or a collection without using a counter
  – Syntax
    \[
    \text{for ( parameter : arrayName )}
    \]
    \[
    \text{statement}
    \]
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
7.6 Enhanced for Statement (Cont.)

• Lines 12-13 are equivalent to
  ```
  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
{
    public static void main( String args[] )
    {
        int array[] = { 1, 2, 3, 4, 5 };

        System.out.println("Effects of passing reference to entire array:");
        System.out.println("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 ] );
    }
}

// Fig. 7.13: PassArray.java
// Passing arrays and individual array elements to methods.
public class PassArray
{
    public static void main( String args[] )
    {
        int array[] = { 1, 2, 3, 4, 5 };

        System.out.println("Effects of passing reference to entire array:");
        System.out.println("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 ] );
    }
}
public static void modifyArray(int array2[])
{
    for (int counter = 0; counter < array2.length; counter++)
        array2[counter] *= 2;
} // end method modifyArray

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

Program output

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
      – 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
        • Arrays are objects
        • Therefore, arrays are passed by reference
Performance Tip 7.1

Passing arrays by reference makes sense for performance reasons. If arrays were passed by value, a copy of each element would be passed. For large, frequently passed arrays, this would waste time and consume considerable storage for the copies of the arrays.
7.8 Case Study: Class GradeBook Using an Array to Store Grades

- Further evolve class GradeBook
- Class GradeBook
  - Represents a grade book that stores and analyzes grades
  - Does not maintain individual grade values
  - Repeat calculations require reentering the same grades
    - Can be solved by storing grades in an array
public class GradeBook
{
    private String courseName; // name of course this GradeBook represents
    private int grades[]; // array of student grades

    public GradeBook( String name, int gradesArray[] )
    {
        courseName = name; // initialize courseName
        grades = gradesArray; // store grades
    } // end two-argument GradeBook constructor

    // method to set the course name
    public void setCourseName( String name )
    {
        courseName = name; // store the course name
    } // end method setCourseName

    // method to retrieve the course name
    public String getCourseName()
    {
        return courseName;
    } // end method getCourseName
public void displayMessage()
{
    // getCourseName gets the name of the course
    System.out.printf( "Welcome to the grade book for\n%s!\n\n", getCourseName() );
} // end method displayMessage

public void processGrades()
{
    // output grades array
    outputGrades();
    System.out.printf( "\nClass average is %.2f\n", getAverage() );
    System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n", getMinimum(), getMaximum() );
    outputBarChart();
} // end method processGrades

public int getMinimum()
{
    int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest
// loop through grades array
for (int grade : grades) {
    // if grade lower than lowGrade, assign it to lowGrade
    if (grade < lowGrade)
        lowGrade = grade; // new lowest grade
}

return lowGrade; // return lowest grade

public int getMaximum() {
    int highGrade = grades[0]; // assume grades[0] is largest

    // loop through grades array
    for (int grade : grades) {
        // if grade greater than highGrade, assign it to highGrade
        if (grade > highGrade)
            highGrade = grade; // new highest grade
    }

    return highGrade; // return highest grade
} // end method getMaximum
```java
// determine average grade for test
public double getAverage()
{
    int total = 0; // initialize total
    // sum grades for one student
    for (int grade : grades)
        total += grade;
    // return average of grades
    return (double) total / grades.length;
} // end method getAverage

// output bar chart displaying grade distribution
public void outputBarChart()
{
    System.out.println( "Grade distribution:" );
    // stores frequency of grades in each range of 10 grades
    int frequency[] = new int[ 11 ];
    // for each grade, increment the appropriate frequency
    for (int grade : grades)
        ++frequency[ grade / 10 ];
}
```

Loop through `grades` to sum grades for one student

Loop through `grades` to calculate frequency
// for each grade frequency, print bar in chart
for ( int count = 0; count < frequency.length; count++ )
{
    // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
    if ( count == 10 )
        System.out.printf( "%5d: ", 100 );
    else
        System.out.printf( "%02d-%02d: ",
            count * 10, count * 10 + 9 );

    // print bar of asterisks
    for ( int stars = 0; stars < frequency[ count ]; stars++ )
        System.out.print( "*" );

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

// output the contents of the grades array
public void outputGrades()
{
    System.out.println( "The grades are:\n" );

    // output each student's grade
    for ( int student = 0; student < grades.length; student++ )
        System.out.printf( "Student %2d: %3d\n", student + 1, grades[ student ] );
} // end method outputGrades

Outline

GradeBook.java
(5 of 5)

Lines 134-136

Loop through grades to display each grade
A test harness (or test application) is responsible for creating an object of the class being tested and providing it with data. This data could come from any of several sources. Test data can be placed directly into an array with an array initializer, it can come from the user at the keyboard, it can come from a file (as you will see in Chapter 14), or it can come from a network (as you will see in Chapter 24). After passing this data to the class's constructor to instantiate the object, the test harness should call upon the object to test its methods and manipulate its data. Gathering data in the test harness like this allows the class to manipulate data from several sources.
public class GradeBookTest
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // array of student grades
        int gradesArray[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };;

        GradeBook myGradeBook = new GradeBook(
            "CS101 Introduction to Java Programming", gradesArray );
        myGradeBook.displayMessage();
        myGradeBook.processGrades();
    } // end main
} // end class GradeBookTest
Welcome to the grade book for CS101 Introduction to Java Programming!

The grades are:

Student  1:  87  
Student  2:  68  
Student  3:  94  
Student  4:  100  
Student  5:  83  
Student  6:  78  
Student  7:  85  
Student  8:  91  
Student  9:  76  
Student 10:  87

Class average is 84.90  
Lowest grade is 68  
Highest grade is 100

Grade distribution:
00–09:  
10–19:  
20–29:  
30–39:  
40–49:  
50–59:  
60–69:  *  
70–79:  **  
80–89:  ****  
90–99:  **  
100:  *
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.
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 $b[0][1]$
  
  ```
  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.)

• 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 }};`
7.9 Multidimensional Arrays (Cont.)

- Creating two-dimensional arrays with array-creation expressions
  - 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
    ```
// 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
} // end class InitArray

Use nested array initializers to initialize array1
Use nested array initializers of different lengths to initialize array2
// 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

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

Values in array2 by row are
1 2
3
4 5 6
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
// Fig. 7.18: GradeBook.java
// Grade book using a two-dimensional array to store grades.

public class GradeBook {
    private String courseName; // name of course this grade book represents
    private int grades[][]; // two-dimensional array of student grades

    // two-argument constructor initializes courseName and grades array
    public GradeBook( String name, int gradesArray[][] )
    {
        courseName = name; // initialize courseName
        grades = gradesArray; // store grades
    } // end two-argument GradeBook constructor

    // method to set the course name
    public void setCourseName( String name )
    {
        courseName = name; // store the course name
    } // end method setCourseName

    // method to retrieve the course name
    public String getCourseName()
    {
        return courseName;
    } // end method getCourseName
28    // display a welcome message to the GradeBook user
29    public void displayMessage()
30    {
31       // getCourseName gets the name of the course
32       System.out.printf( "Welcome to the grade book for\n%s!\n\n", 
33          getCourseName() );
34    } // end method displayMessage
35
36    // perform various operations on the data
37    public void processGrades()
38    {
39       // output grades array
40       outputGrades();
41
42       // call methods getMinimum and getMaximum
43       System.out.printf( "\n%s %d
%s %d
\n", 
44          "Lowest grade in the grade book is", getMinimum(), 
45          "Highest grade in the grade book is", getMaximum() );
46
47       // output grade distribution chart of all grades on all tests
48       outputBarChart();
49    } // end method processGrades
50
51    // find minimum grade
52    public int getMinimum()
53    {
54       // assume first element of grades array is smallest
55       int lowGrade = grades[ 0 ][ 0 ];
// loop through rows of grades array
for (int studentGrades[] : grades) {
    // loop through columns of current row
    for (int grade : studentGrades) {
        // if grade less than lowGrade
        if (grade < lowGrade) {
            lowGrade = grade;
        }
    }
}

return lowGrade; // return lowest grade

// find maximum grade
public int getMaximum() {
    // assume first element of grades array is largest
    int highGrade = grades[0][0];
// loop through rows of grades array
for (int studentGrades[] : grades) {
   // loop through columns of current row
   for (int grade : studentGrades) {
      // if grade greater than highGrade
      if (grade > highGrade)
         highGrade = grade;
   } // end inner for
} // end outer for

return highGrade; // return highest grade
} // end method getMaximum

// determine average grade for particular set of grades
public double getAverage(int setOfGrades[]) {
   int total = 0; // initialize total
   // sum grades for one student
   for (int grade : setOfGrades)
      total += grade;
   // return average of grades
   return (double) total / setOfGrades.length;
} // end method getAverage

Loop through rows of grades to find the highest grade of any student

Calculate a particular student’s semester average
// output bar chart displaying overall grade distribution
public void outputBarChart()
{
    System.out.println( "Overall grade distribution:" );

    // stores frequency of grades in each range of 10 grades
    int frequency[] = new int[ 11 ];

    // for each grade in GradeBook, increment the appropriate frequency
    for ( int studentGrades[] : grades )
    {
        for ( int grade : studentGrades )
            ++frequency[ grade / 10 ];
    } // end outer for

    // for each grade frequency, print bar in chart
    for ( int count = 0; count < frequency.length; count++ )
    {
        // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
        if ( count == 10 )
            System.out.printf( "%5d: ", 100 );
        else
            System.out.printf( "%02d-%02d: ",
                    count * 10, count * 10 + 9 );

        // print bar of asterisks
        for ( int stars = 0; stars < frequency[ count ]; stars++ )
            System.out.print( "*" );
    }
}

Calculate the distribution of all student grades
    System.out.println(); // start a new line of output
    } // end outer for
} // end method outputBarChart

// output the contents of the grades array
public void outputGrades()
{
    System.out.println( "The grades are:\n" );
    System.out.println( "          "); // align column heads

    // create a column heading for each of the tests
    for ( int test = 0; test < grades[ 0 ].length; test++ )
        System.out.printf( "Test %d  ");

    System.out.println( "Average" ); // student average column heading

    // create rows/columns of text representing array grades
    for ( int student = 0; student < grades.length; student++ )
    {
        System.out.printf( "Student %2d", student + 1 );

        for ( int test : grades[ student ] ) // output student's grades
            System.out.printf( "%8d", test );
159     // call method getAverage to calculate student's average grade;
160     // pass row of grades as the argument to getAverage
161     double average = getAverage( grades[ student ] );
162     System.out.printf( "%.2f\n", average );
163     } // end outer for
164 } // end method outputGrades
165 } // end class GradeBook
// Fig. 7.19: GradeBookTest.java
// Creates GradeBook object using a two-dimensional array of grades.

public class GradeBookTest
{
    // main method begins program execution
    public static void main( String args[] )
    {
        // two-dimensional array of student grades
        int gradesArray[][] = {{ 87, 96, 70 },
                                { 68, 87, 90 },
                                { 94, 100, 90 },
                                { 100, 81, 82 },
                                { 83, 65, 85 },
                                { 78, 87, 65 },
                                { 85, 75, 83 },
                                { 91, 94, 100 },
                                { 76, 72, 84 },
                                { 87, 93, 73 }};

        GradeBook myGradeBook = new GradeBook(
            "CS101 Introduction to Java Programming",
            gradesArray);
        myGradeBook.displayMessage();
        myGradeBook.processGrades();
    } // end main
} // end class GradeBookTest
Welcome to the grade book for CS101 Introduction to Java Programming!

The grades are:

<table>
<thead>
<tr>
<th>Student</th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87</td>
<td>96</td>
<td>70</td>
<td>84.33</td>
</tr>
<tr>
<td>2</td>
<td>68</td>
<td>87</td>
<td>90</td>
<td>81.67</td>
</tr>
<tr>
<td>3</td>
<td>94</td>
<td>100</td>
<td>90</td>
<td>94.67</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>81</td>
<td>82</td>
<td>87.67</td>
</tr>
<tr>
<td>5</td>
<td>83</td>
<td>65</td>
<td>85</td>
<td>77.67</td>
</tr>
<tr>
<td>6</td>
<td>78</td>
<td>87</td>
<td>65</td>
<td>76.67</td>
</tr>
<tr>
<td>7</td>
<td>85</td>
<td>75</td>
<td>83</td>
<td>81.00</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td>94</td>
<td>100</td>
<td>95.00</td>
</tr>
<tr>
<td>9</td>
<td>76</td>
<td>72</td>
<td>84</td>
<td>77.33</td>
</tr>
<tr>
<td>10</td>
<td>87</td>
<td>93</td>
<td>73</td>
<td>84.33</td>
</tr>
</tbody>
</table>

Lowest grade in the grade book is 65
Highest grade in the grade book is 100

Overall grade distribution:
00-09: 
10-19: 
20-29: 
30-39: 
40-49: 
50-59: 
60-69: ***
70-79: *****
80-89: **********
90-99: ********
100: ***
7.11 Variable-Length Argument Lists

- Variable-length argument lists
  - 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
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;
    }
}
```java
25 System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n", 
   d1, d2, d3, d4 );
27
28 System.out.printf( "Average of d1 and d2 is %.1f\n", 
   average( d1, d2 ) );
29 System.out.printf( "Average of d1, d2 and d3 is %.1f\n", 
   average( d1, d2, d3 ) );
30 System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n", 
   average( d1, d2, d3, d4 ) );
32 } // end main
33 } // 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]
// 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 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
", "Index", "Value" );
        }
    }
}
```java
InitArray.java

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

Missing command-line arguments

Three command-line arguments are 5, 0 and 4

Three command-line arguments are 10, 1 and 2

Program output
```
7.13 (Optional) GUI and Graphics Case Study: Drawing Arcs

• **Draw rainbow**
  - Use arrays
  - Use repetition statement
  - Use `Graphics` method `fillArc`
// 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
{
    // Define indigo and violet
    final Color VIOLET = new Color( 128, 0, 128 );
    final Color INDIGO = new Color( 75, 0, 130 );

    // colors to use in the rainbow, starting from the innermost
    // The two white entries result in an empty arc in the center
    private Color colors[] =
    {
        Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
        Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED
    };

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

    // draws a rainbow using
    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 for
} // end method paintComponent
} // end class DrawRainbow

Draw a filled semicircle
// Fig. 7.23: DrawRainbowTest.java
// Test application to display a rainbow.
import javax.swing.JFrame;

public 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 );
    } // end main
} // end class DrawRainbowTest
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
7.14 (Optional) Software Engineering Case Study (Cont.)

• 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
### Fig. 7.25 | Collaborations in the ATM system.

<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>BalanceInquiry</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>Withdrawal</td>
</tr>
<tr>
<td></td>
<td>Execute</td>
<td>Deposit</td>
</tr>
<tr>
<td>BalanceInquiry</td>
<td>getAvailableBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>getTotalBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>displayMessage</td>
<td>Screen</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>isSufficientCashAvailable</td>
<td>CashDispenser</td>
</tr>
<tr>
<td></td>
<td>debit</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>dispenseCash</td>
<td>CashDispenser</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>BankDatabase</td>
</tr>
<tr>
<td>BankDatabase</td>
<td>validatePIN</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>getAvailableBalance</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>getTotalBalance</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>debit</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>Credit</td>
<td>Account</td>
</tr>
</tbody>
</table>
7.14 (Optional) Software Engineering Case Study (Cont.)

• Interaction Diagrams
  – Used to model interactions in the 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 alone 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.