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.
Introduction

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

12-element array.
Arrays (Cont.)

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

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

• Adds 2 to c[ 11 ]

Common Programming Error

• Using a value of type long
• Must be int (or)
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

- `int c[12];` → 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 /Tricky Interview Question

• `int[] a, b, c;`
• `int a[], b, c;`
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
Examples Using Arrays

- Using an array initializer
  - Use *initializer list*
    - Items enclosed in braces ({ })
    - Items in list separated by commas
      ```java
      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, }; // corrected
        System.out.printf( "Index%8s%s
", "Value", "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 ] );
    }
}

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.
Error-Prevention Tip

• When a program attempts to access an element outside the array bounds, an
  `ArrayIndexOutOfBoundsException` occurs. Exception handling is discussed later.

Error-Prevention Tip

• When writing code to loop through an array, ensure that the array index is always:
  - >= 0 and < array.length
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

Outline

- Card.java
- Lines 17-20

```java
// 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
    }

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

Return the string representation of a card
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]);

        // shuffle deck of Cards with one-pass algorithm
        for (int first = 0; first < deck.length; first++)
            Card temp = deck[first];
        deck[first] = deck[second];
        deck[second] = temp;
    }

    // deal one Card
    public Card dealCard() {
        if (currentCard < deck.length)
            return deck[currentCard++]; // return current Card in array
        else
            return null; // return null to indicate that all Cards were dealt
    }
}

// Fig. 7.10: DeckOfCards.java
// DeckOfCards class represents a deck of playing cards.
import java.util.Random;
// Fig. 7.11: DeckOfCardsTest.java
// Card shuffling and dealing application.

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

Six of Spades       Eight of Spades     Six of Clubs        Nine of Hearts
Queen of Hearts     Seven of Clubs      Nine of Spades      King of Hearts
Three of Diamonds   Deuce of Clubs      Ace of Hearts       Ten of Spades
Four of Spades      Ace of Clubs        Seven of Diamonds   Four of Hearts
Three of Clubs      Deuce of Hearts     Five of Spades      Jack of Diamonds
King of Clubs       Ten of Hearts       Three of Hearts     Six of Diamonds
Queen of Clubs      Eight of Diamonds   King of Diamonds    Ten of Diamonds
Three of Spades     Four of Diamonds    Nine of Clubs       Six of Hearts
Ace of Spades       Seven of Spades     Queen of Diamonds   Five of Clubs
Ace of Diamonds     Ten of Clubs       Jack of Spades      King of Spades
Five of Diamonds    Ten of Clubs

Outline

- DeckOfCardsTest
- DeckOfCardsTest.java

Outline

- DeckOfCardsTest
- DeckOfCardsTest.java

Outline

- DeckOfCardsTest
- DeckOfCardsTest.java
Enhanced for Statement

- Enhanced for statement
  - Iterates through elements of an array or a collection without using a counter
  - Syntax
    ```java
    for (parameter : arrayName) 
    statement
    ```

Outline

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

public class EnhancedForTest
{
    public static void main(String args[]) 
    {
        int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
        int total = 0;
        for (int number : array) 
        {
            total += number;
        }
        System.out.printf( "Total of array elements: %d\n", total );
    }
}
```

Total of array elements: 849

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

Passing Arrays to Methods

• To pass array argument to a method
  — Specify array name without brackets

  ```java
  Array hour ly Temper at ures is declared as
  int hourlyTemperatures = new int[ 24 ];
  ```

  • The method call
    ```java
    modifyArray( int array[] );
    ```

  • Passes array `hourlyTemperatures` to method `modifyArray`
Passing Arrays to Methods

• 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

• Passing arrays by reference
  – No copy, fast
Case Study: Class GradeBook Using an Array to Store Grades

• Check the example in your textbook

Multidimensional Arrays

• Multidimensional arrays
  – Tables with rows and columns
    • Two-dimensional array
    • m-by-n array
Two-dimensional array with three rows and four columns.

<table>
<thead>
<tr>
<th></th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Row 0</td>
<td>a[0][0]</td>
<td>a[0][1]</td>
<td>a[0][2]</td>
<td>a[0][3]</td>
</tr>
<tr>
<td>Row 1</td>
<td>a[1][0]</td>
<td>a[1][1]</td>
<td>a[1][2]</td>
<td>a[1][3]</td>
</tr>
</tbody>
</table>

Multidimensional Arrays (Cont.)

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

• Two-dimensional arrays with rows of different lengths
  – Lengths of rows in array are not required to be the same
    • E.g., \texttt{int b[][] = \{ \{ 1, 2 \}, \{ 1, 1, 1 \} \};}

Multidimensional Arrays (Cont.)

• Creating two-dimensional arrays with array-creation expressions
  – 3-by-4 array
    \begin{verbatim}
    int b[][];
    b = new int[ 3 ][ 4 ];
    \end{verbatim}
  – Rows can have different number of columns
    \begin{verbatim}
    int b[][];
    b = new int[ 2 ][ ];
    b[ 0 ] = new int[ 5 ];
    b[ 1 ] = new int[ 3 ];
    \end{verbatim}
public class InitArray {
    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( "Values in array2 by row are" );
        outputArray( array2 ); // displays array2 by row
    } // end main
}

public static void outputArray( int array[][] ) {
    for ( int row = 0; row < array.length; 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
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;
  ```

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
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

    // display a welcome message to the GradeBook user
    public void displayMessage()
    {
        System.out.printf( "Welcome to the grade book for
%s!

", getCourseName() );
    } // end method displayMessage

    // perform various operations on the data
    public void processGrades()
    {
        outputGrades();
        System.out.printf( "Lowest grade in the grade book is", getMinimum() );
        System.out.printf( "Highest grade in the grade book is", getMaximum() );
        outputBarChart();
    } // end method processGrades

    // find minimum grade
    public int getMinimum()
    {
        // assume first element of grades array is smallest
        int lowGrade = grades[ 0 ][ 0 ];
        System.out.println( "Lowest grade in the grade book is", getMinimum() );
        System.out.println( "Highest grade in the grade book is", getMaximum() );
        outputBarChart();
    } // end method getMinimum
Outline

• GradeBook.java
  (3 of 7)
  Lines 53-67

57 // loop through rows of grades array
58 for (int studentGrades[] : grades )
59 {
60 // loop through columns of current row
61 for (int grade : studentGrades )
62 {
63 // add grade less than lowGrade
64 if ( grade < lowGrade )
65 lowGrade = grade;
66 // end inner for
67 } // end outer for
68 return lowGrade; // return lowest grade
} // end method getMinimum
69
70 // find maximum grade
71 public int getMaximum()
72 {
73 // assume first element of grades array is largest
74 int highGrade = grades[ 0 ][ 0 ];
75 // loop through rows of grades array
76 for ( int studentGrades[] : grades )
77 {
78 // loop through columns of current row
79 for ( int grade : studentGrades )
80 {
81 // if grade greater than highGrade
82 if ( grade > highGrade )
83 highGrade = grade;
84 } // end inner for
85 } // end outer for
86 return highGrade; // return highest grade
87 } // end method getMaximum
88
89 // determine average grade for particular set of grades
90 public double getAverage( int setOfGrades[] )
91 {
92 int total = 0; // initialize total
93 // sum grades for one student
94 for ( int grade : setOfGrades )
95 total += grade;
96 return (double) total / setOfGrades.length;
97 } // end method getAverage
```java
// 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 )
        ++frequency[ grade / 10 ];
    // for each grade frequency, print bar in chart
    for ( int count = 0; count < frequency.length; count++ )
    {
        // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
        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
    }
    // output the contents of the grades array
    public void outputGrades()
    {
        System.out.println(); // start a new line of output
        System.out.printf( "Test %d  ", test + 1 );
        System.out.println( "Average" ); // student average column heading
        for ( int student = 0; student < grades.length; student++ )
            System.out.printf( "Student %2d", student + 1 );
        // create a column heading for each of the tests
        for ( int test = 0; test < grades[ 0 ].length; test++ )
            System.out.printf( "Test %d ", test + 1 );
        // 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( "%d", test );
        }
    }
    // output the bar chart
    outputBarChart();
    // output the contents of the grades array
    outputGrades();
}
```
```java
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( "%9.2f\n", average );
163  } // end outer for
164 }  // end method outputGrades
165 }  // end class GradeBook
```

### Quiz

```java
public class GradeBookTest {
    public static void main( String args[] ) {
        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 } };
        // two-dimensional array of student grades
    }
}
```

Each row represents a student; each column represents an exam grade.
Implement

- float average(int whichExam)
- Void printout()

<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>Student 1</td>
<td>87</td>
<td>96</td>
<td>76</td>
<td>86.33</td>
</tr>
<tr>
<td>Student 2</td>
<td>68</td>
<td>87</td>
<td>90</td>
<td>85.67</td>
</tr>
<tr>
<td>Student 3</td>
<td>94</td>
<td>38</td>
<td>90</td>
<td>94.67</td>
</tr>
<tr>
<td>Student 4</td>
<td>90</td>
<td>93</td>
<td>83</td>
<td>87.67</td>
</tr>
<tr>
<td>Student 5</td>
<td>83</td>
<td>65</td>
<td>85</td>
<td>77.67</td>
</tr>
<tr>
<td>Student 6</td>
<td>78</td>
<td>87</td>
<td>85</td>
<td>80.67</td>
</tr>
<tr>
<td>Student 7</td>
<td>95</td>
<td>75</td>
<td>83</td>
<td>85.33</td>
</tr>
<tr>
<td>Student 8</td>
<td>91</td>
<td>94</td>
<td>100</td>
<td>95.33</td>
</tr>
<tr>
<td>Student 9</td>
<td>76</td>
<td>72</td>
<td>84</td>
<td>77.33</td>
</tr>
<tr>
<td>Student 10</td>
<td>87</td>
<td>93</td>
<td>73</td>
<td>84.33</td>
</tr>
</tbody>
</table>

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

System.out.printf("d1 = %.1f
d2 = %.1f
d3 = %.1f
d4 = %.1f
", d1, d2, d3, d4);

System.out.printf("Average of d1 and d2 is %.1f
", average(d1, d2));
System.out.printf("Average of d1, d2 and d3 is %.1f
", average(d1, d2, d3));
System.out.printf("Average of d1, d2, d3 and d4 is %.1f
", average(d1, d2, d3, d4));

} // end main
} // end class VarargsTest

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

• An ellipsis may be placed only at the end of the parameter list.
  – in the middle: a syntax error.

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]
Outline

// 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 the 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" );

            // display array index and value
            for ( int counter = 0; counter < array.length; counter++ )
                System.out.printf( "%5d%8d
", counter, array[counter] );
        }
    }
}

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

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

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
GUI and Graphics Case Study: Drawing Arcs

• Draw rainbow
  – Use arrays
  – Use repetition statement
  – Use Graphics method fillArc

```java
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
    }

    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        int radius = 35; // radius of an arch
        fillArc(g, 10, 10, radius, radius, 180, 180); // draw an arc
    }
}
```

DrawRainbow
// 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
Drawing a spiral using `drawLine` (left) and `drawArc` (right).