7

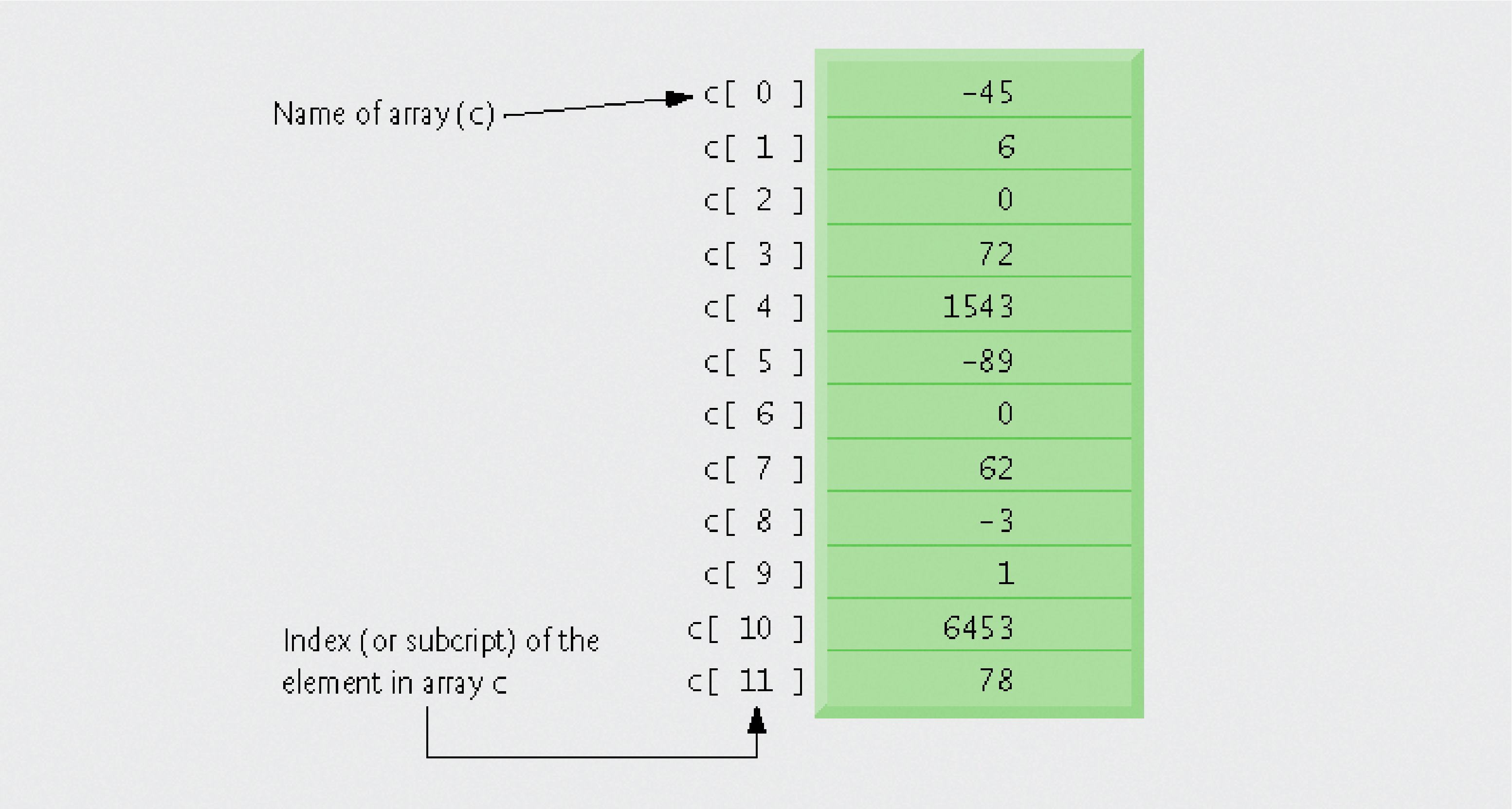
Arrays

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



Name of array (c)

c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

Index (or subcript) of the element in array c

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**

```
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`.

7.2 Arrays (Cont.)

- **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`**

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

  - **Equivalent to**

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

- **We can create arrays of objects too**

```
String b[] = new String[ 100 ];
```

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

Outline

```
1  // Fig. 7.2: InitArray.java
2  // Creating an array.
3
4  public class InitArray
5  {
6     public static void main( Stri
7     {
8        int array[]; // declare array named array
9
10       array = new int[ 10 ]; // create the sp
11
12       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
13
14       // output each array element's value
15       for ( int counter = 0; counter < array.length; counter++ )
16          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
17    } // end main
18 } // 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
```

Declare `array` as an array of `ints`

Create `10` `ints` for `array`; each `int` is initialized to `0` by default

`array.length` returns length of `array`

Each `int` is initialized to `0` by default

`array[counter]` returns `int` associated with index in `array`

InitArray.java

Line 8
Declare array as an array of **ints**

Line 10
Create 10 **ints** for array; each **int** is initialized to **0** by default

Line 15
array.length returns length of array

Line 16
array[counter] returns **int** associated with index in array

Program output

◀ ▶

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`**

```
1  // Fig. 7.3: InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main( String args[] )
7     {
8        // initializer list specifies the value for each element
9        int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
12
13       // output each array element's value
14       for ( int counter = 0; counter < array.length; counter++ )
15          System.out.printf( "%5d%8d\n", counter, array[ counter ] );
16    } // end main
17 } // end class InitArray
```

Declare `array` as an array of `int`s

Compiler uses initializer list to allocate array

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

Outline

itArray.java

Line 9
Declare array as an array of ints

Line 9
Compiler uses initializer list to allocate array

Program output

7.4 Examples Using Arrays (Cont.)

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

Outline

```
1  // Fig. 7.4: InitArray.java
2  // Calculating values to be placed into elements of an array.
3
4  public class InitArray
5  {
6     public static void main( String args[] )
7     {
8        final int ARRAY_LENGTH = 10; // declare constant
9        int array[] = new int[ ARRAY_LENGTH ]; // create ar
10
11       // calculate value for each array element
12       for ( int counter = 0; counter < array.length; counter++ )
13          array[ counter ] = 2 + 2 * counter;
14
15       System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings
16
17       // output each array element's value
18       for ( int counter = 0; counter < array.length; counter++ )
19          System.out.printf( "%5d%8d\n", counter
20    } // end main
21 } // end class InitArray
```

Declare constant variable ARRAY_LENGTH using the final modifier

Declare and create array that contains 10 ints

Use array index to assign array value

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

InitArray.java

8
are constant
variable

Line 9
Declare and create array that contains 10 ints

Line 13
Use array index to assign array

Program output

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.

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**

Outline

```
1  // Fig. 7.5: SumArray.java
2  // Computing the sum of the elements of
3
4  public class SumArray
5  {
6     public static void main( String args[] )
7     {
8        int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int counter = 0; counter < array.length; counter++ )
13          total += array[ counter ];
14
15       System.out.printf( "Total of array elements: %d\n",   total );
16    } // end main
17 } // end class SumArray
```

Declare array with initializer list

Sum all array values

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

SumArray.java

Line 8
Declare array with initializer list

Lines 12-13
Sum all array values

Program output

7.4 Examples Using Arrays (Cont.)

- **Using the elements of an array as counters**
  - **Use a series of counter variables to summarize data**

Outline

```
1  // Fig. 7.7: RollDie.java
2  // Roll a six-sided die 6000 times.
3  import java.util.Random;
4
5  public class RollDie
6  {
7     public static void main( String args[] )
8     {
9        Random randomNumbers = new Random(); // random number generator
10       int frequency[] = new int[ 7 ]; // array of frequency counters
11
12       // roll die 6000 times; use die value as frequency ind
13       for ( int roll = 1; roll <= 6000; roll++ )
14          ++frequency[ 1 + randomNumbers.nextInt( 6 ) ];
15
16       System.out.printf( "%s%10s\n"
17
18       // output each array element's value
19       for ( int face = 1; face < frequency.length; face++ )
20          System.out.printf( "%4d%10d\n", face, frequency[ face ] );
21    } // end main
22 } // end class RollDie
```

Declare frequency as array of 7 ints

Generate 6000 random integers in range 1-6

Increment frequency values at index associated with random number

```
Face Frequency
   1       988
   2       963
   3      1018
   4      1041
   5       978
   6      1012
```

Die.java

Line 10
Declare frequency as
of 7 ints

3-14
Generate 6000 random integers in range 1-6

Line 14
Increment frequency values at index associated with random number

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.

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
```

Outline

EnhancedForTest.java

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

public class EnhancedForTest
{
   public static void main( String args[] )
   {
      int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
      int total = 0;

      // add each element's value to total
      for ( int number : array )
         total += number;

      System.out.printf( "Total of array elements: %d\n", total );
   } // end main
} // end class EnhancedForTest
```

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

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

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

      ```
      int hourlyTemperatures = new int[ 24 ];
      ```

    - The method call

      ```
      modifyArray( hourlyTemperatures );
      ```

    - Passes array `hourlyTemperatures` to method `modifyArray`

Outline

```
1  // Fig. 7.13: PassArray.java
2  // Passing arrays and individual array elements to methods.
3
4  public class PassArray
5  {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main( String args[] )
8     {
9        int array[] = { 1, 2, 3, 4, 5 };
10
11       System.out.println(
12          "Effects of passing reference to entire array:\n" +
13          "The values of the original array are:" );
14
15       // output original array elements
16       for ( int value : array )
17          System.out.printf( "   %d", value );
18
19       modifyArray( array ); // pass array reference
20       System.out.println( "\n\nThe values of the modified array are:" );
21
22       // output modified array elements
23       for ( int value : array )
24          System.out.printf( "   %d", value );
25
26       System.out.printf(
27          "\n\nEffects of passing array element value:\n" +
28          "array[3] before modifyElement: %d\n", array[ 3 ] );
```

Declare 5-`int array` with initializer list

Pass entire array to method `modifyArray`

`PassArray.java`

(1 of 2)

Line 9

Line 19

```
29
30       modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
31       System.out.printf(
32          "array[3] after modifyElement
33    } // end main
34
35    // multiply each element of an array by 2
36    public static void modifyArray( int array2[] )
37    {
38       for ( int counter = 0; counter < array2.length; counter++ )
39          array2[ counter ] *= 2;
40    } // end method modifyArray
41
42    // multiply argument by 2
43    public static void modifyElement( int element )
44    {
45       element *= 2;
46       System.out.printf(
47          "Value of element in modifyElement: %d\n", element );
48    } // end method modifyElement
49 } // end class PassArray
```

Pass array element `array[3]` to method `modifyElement`

Method `modifyArray` manipulates the array directly

Method `modifyElement` manipulates a primitive’s copy

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Outline

ava

(2 of 2)

Line 30

Lines 36-40

Lines 43-48

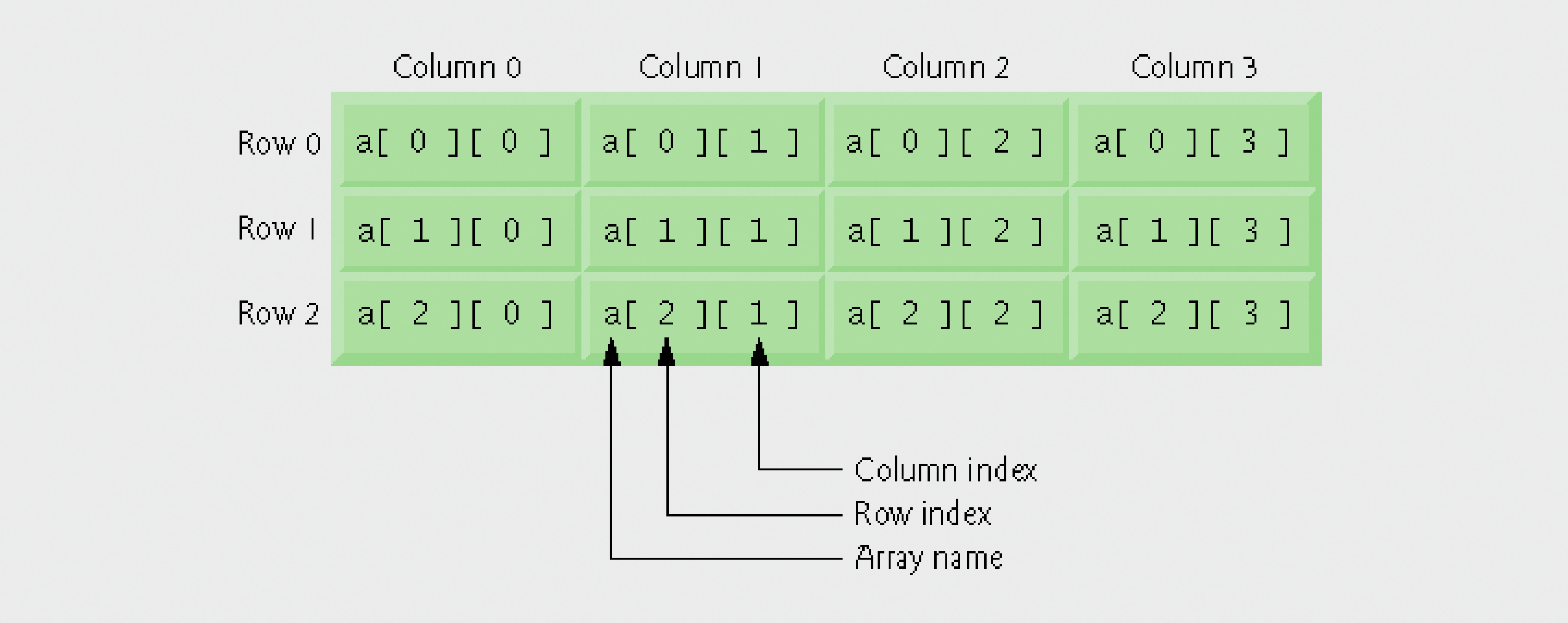
Program output

7.7 Passing Arrays to Methods (Cont.)

- 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**



	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

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]`

    ```
    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`

7.9 Multidimensional Arrays (Cont.)

- Creating two-dimensional arrays with array-creation expressions
  - Can be created dynamically
    - 3-by-4 array

```
int b[][];
b = new int[ 3 ][ 4 ];
```

    - Rows can have different number of columns

```
int b[][];
b = new int[ 2 ][ ];   // create 2 rows
b[ 0 ] = new int[ 5 ]; // create 5 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.,

```
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

Outline

```
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             // if grade less than lowGra
64             if ( grade < lowGrade )
65                lowGrade = grade;
66          } // end inner for
67       } // end outer for
68
69       return lowGrade; // return lowest grade
70    } // end method getMinimum
71
72    // find maximum grade
73    public int getMaximum()
74    {
75       // assume first element of grades array is largest
76       int highGrade = grades[ 0 ][ 0 ];
77
```

Loop through rows of `grades` to find the lowest grade of any student

`GradeBook.java`

(3 of 7)

Lines 58-67

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**

```
1  // Fig. 7.20: VarargsTest.java
2  // Using variable-length argument lists.
3
4  public class VarargsTest
5  {
6     // calculate average
7     public static double average( double... numbers )
8     {
9        double total = 0.0; // initialize total
10
11       // calculate total using the enha
12       for ( double d : numbers )
13          total += d;
14
15       return total / numbe
16    } // end method average
17
18    public static void m
19    {
20       double d1 = 10.0;
21       double d2 = 20.0;
22       double d3 = 30.0;
23       double d4 = 40.0;
24
```

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

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

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

Outline

`VarargsTest`

`.java`

(1 of 2)

Line 7

Lines 12-13

Line 15

Outline

VarargsTest

.java

2)

Line 29

Line 31

Line 33

Program output

```
25       System.out.printf( "d1 = %.1f\nd2 = %.1f\nd3 = %.1f\nd4 = %.1f\n\n",
26          d1, d2, d3, d4 );
27
28       System.out.printf( "Average of d1 and d2 is %.1f\n",
29          average( d1, d2 ) );
30       System.out.printf( "Average of d1, d2 and d
31          average( d1, d2, d3 ) );
32       System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
33          average( d1, d2, d3, d4 ) );
34    } // end main
35 } // end class VarargsTest
```

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

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

Invoke method average with two arguments

Invoke method average with three arguments

Invoke method average with four arguments

Common Programming Error 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 ]`

```
1  // Fig. 7.21: InitArray.java
2  // Using command-line arguments to initialize an array.
3
4  public class InitArray
5  {
6     public static void main( String args[] )
7     {
8        // check number of command-line arguments
9        if ( args.length != 3 )
10          System.out.println(
11             "Error: Please re-enter                                n" +
12             "an array                                              " );
13       else
14       {
15          // get array size from first command-line argument
16          int arrayLength = Integer.parseInt( args[ 0 ] );
17          int array[] = new int[ arrayLength ]; // create array
18
19          // get initial value and increment from co
20          int initialValue = Integer.parseInt( args[ 1 ] );
21          int increment = Integer.parseInt( args[ 2 ] );
22
23          // calculate value for each array element
24          for ( int counter = 0; counter < array.le
25             array[ counter ] = initialValue + incr
26
27          System.out.printf( "%s%8s\n", "Index", "Value" );
28
```

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

Outline

`InitArray.java`

(1 of 2)

Line 6

Line 9

Line 16

Lines 20-21

Lines 24-25

Outline

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

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

Missing command-line arguments

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

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

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

Three command-line arguments are `10`, `1` and `2`

`InitArray.java`

(2 of 2)

Program output

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

- Draw rainbow
  - Use arrays
  - Use repetition statement
  - Use `Graphics` method `fillArc`

```java
1  // Fig. 7.22: DrawRainbow.java
2  // Demonstrates using colors in an array.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawRainbow extends JPanel
8  {
9     // Define indigo and violet
10    final Color VIOLET = new Color( 128, 0, 128 );
11    final Color INDIGO = new Color( 75, 0, 130 );
12
13    // colors to use in the rainbow, starting from the innermost
14    // The two white entries result in an empty arc in the center
15    private Color colors[] =
16       { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
17         Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };
18
19    // constructor
20    public DrawRainbow()
21    {
22       setBackground( Color.WHITE ); // set the background to white
23    } // end DrawRainbow constructor
24
25    // draws a rainbow using
26    public void paintComponent( Graphics g )
27    {
28       super.paintComponent( g );
29
30       int radius = 20; // radius of an arch
```

Set component background to white

Outline

DrawRainbow

.java

(1 of 2)

Line 22

```java
31
32       // draw the rainbow near the bottom-center
33       int centerX = getWidth() / 2;
34       int centerY = getHeight() - 10;
35
36       // draws filled arcs starting with the outermost
37       for ( int counter = colors.length; counter > 0; counter-- )
38       {
39          // set the color for the current arc
40          g.setColor( colors[ counter - 1 ] );
41
42          // fill the arc from 0 to 180 degrees
43          g.fillArc( centerX - counter * radius,
44             centerY - counter * radius,
45             counter * radius * 2, counter * radius * 2, 0, 180 );
46       } // end for
47    } // end method paintComponent
48 } // end class DrawRainbow
```

Draw a filled semicircle

Outline

`DrawRainbow`

`.java`

(2 of 2)

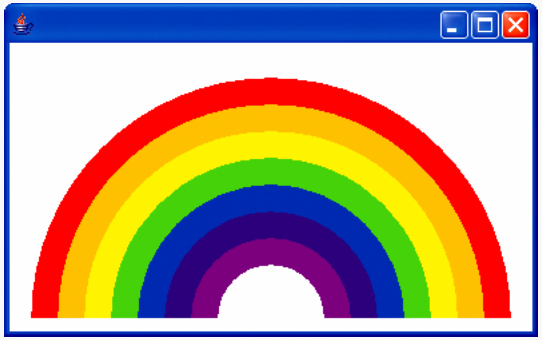
Lines 43-45

Outline

```
// 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
```



DrawRainbowTest
.java

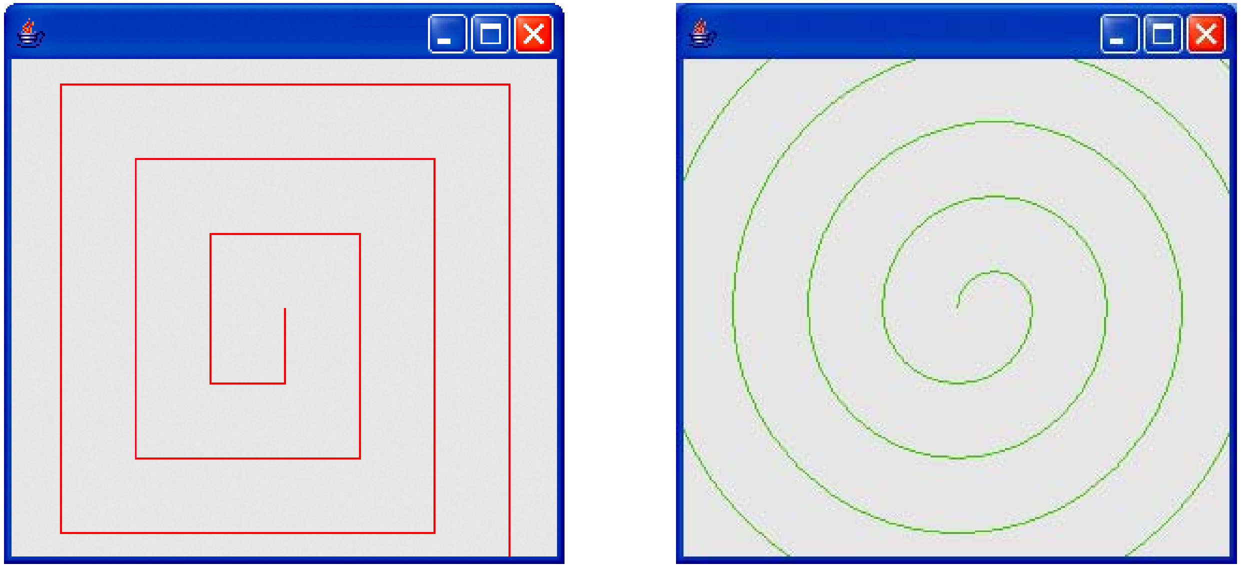


Fig. 7.24 | **Drawing a spiral using `drawLine` (left) and `drawArc` (right).**

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 use
    - What ATM should do to perform transactions
  - For each action, decide
    - Which objects must interact
      - Sending object
      - Receiving object

An object of class…	sends the message…	to an object of class…
ATM	displayMessage	Screen
	getInput	Keypad
	authenticateUser	BankDatabase
	execute	BalanceInquiry
	execute	Withdrawal
	Execute	Deposit
BalanceInquiry	getAvailableBalance	BankDatabase
	getTotalBalance	BankDatabase
	displayMessage	Screen
Withdrawal	displayMessage	Screen
	getInput	Keypad
	getAvailableBalance	BankDatabase
	isSufficientCashAvailable	CashDispenser
	debit	BankDatabase
	dispenseCash	CashDispenser
Deposit	displayMessage	Screen
	getInput	Keypad
	isEnvelopeReceived	DepositSlot
	Credit	BankDatabase
BankDatabase	validatePIN	Account
	getAvailableBalance	Account
	getTotalBalance	Account
	debit	Account
	Credit	Account

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 alone these lines in the direction shown by arrows**
  - **Name of message appears next to the arrow**

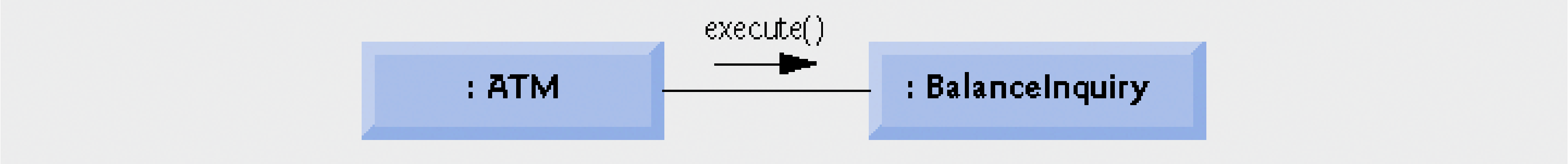


Fig. 7.26 | Communication diagram of the ATM executing a balance inquiry.

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

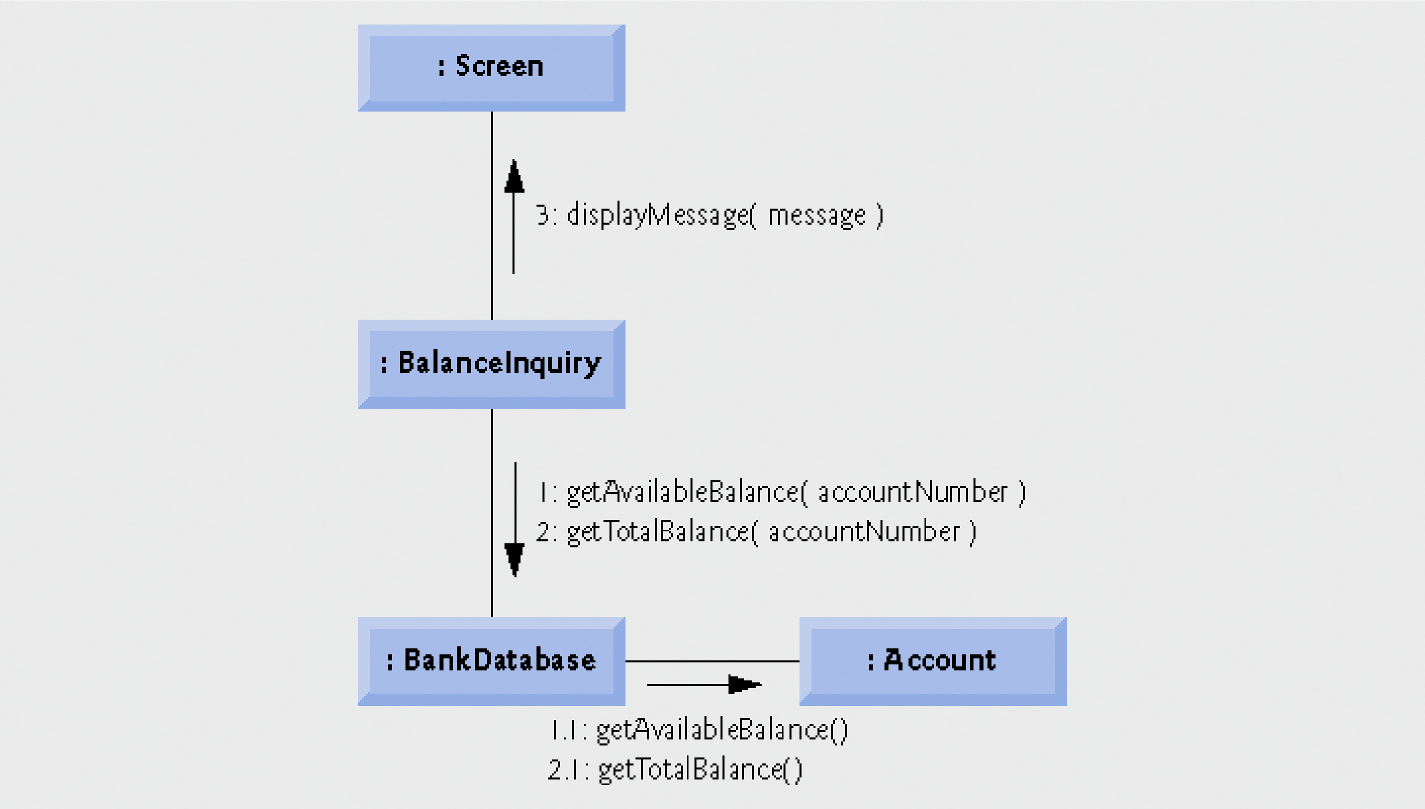

: Screen
3: displayMessage(message)
: BalanceInquiry
1: getAvailableBalance(accountNumber)
2: getTotalBalance(accountNumber)
: BankDatabase
: Account
1.1: getAvailableBalance()
2.1: getTotalBalance()

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

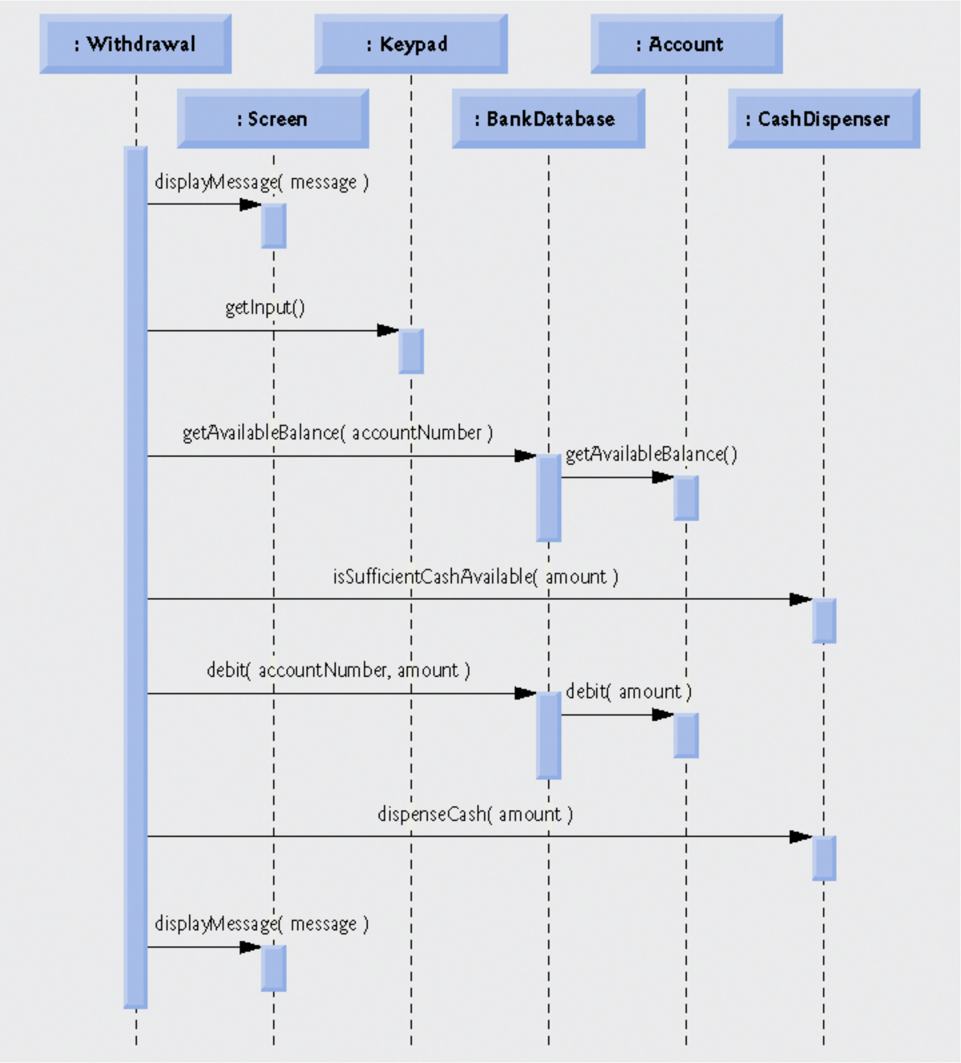

: Withdrawal
: Keypad
: Account
: Screen
: BankDatabase
: CashDispenser
displayMessage(message)
getInput()
getAvailableBalance(accountNumber)
getAvailableBalance()
isSufficientCashAvailable(amount)
debit(accountNumber, amount)
debit(amount)
dispenseCash(amount)
displayMessage(message)

Fig. 7.28 | **Sequence diagram that models a `Withdrawal` executing.**

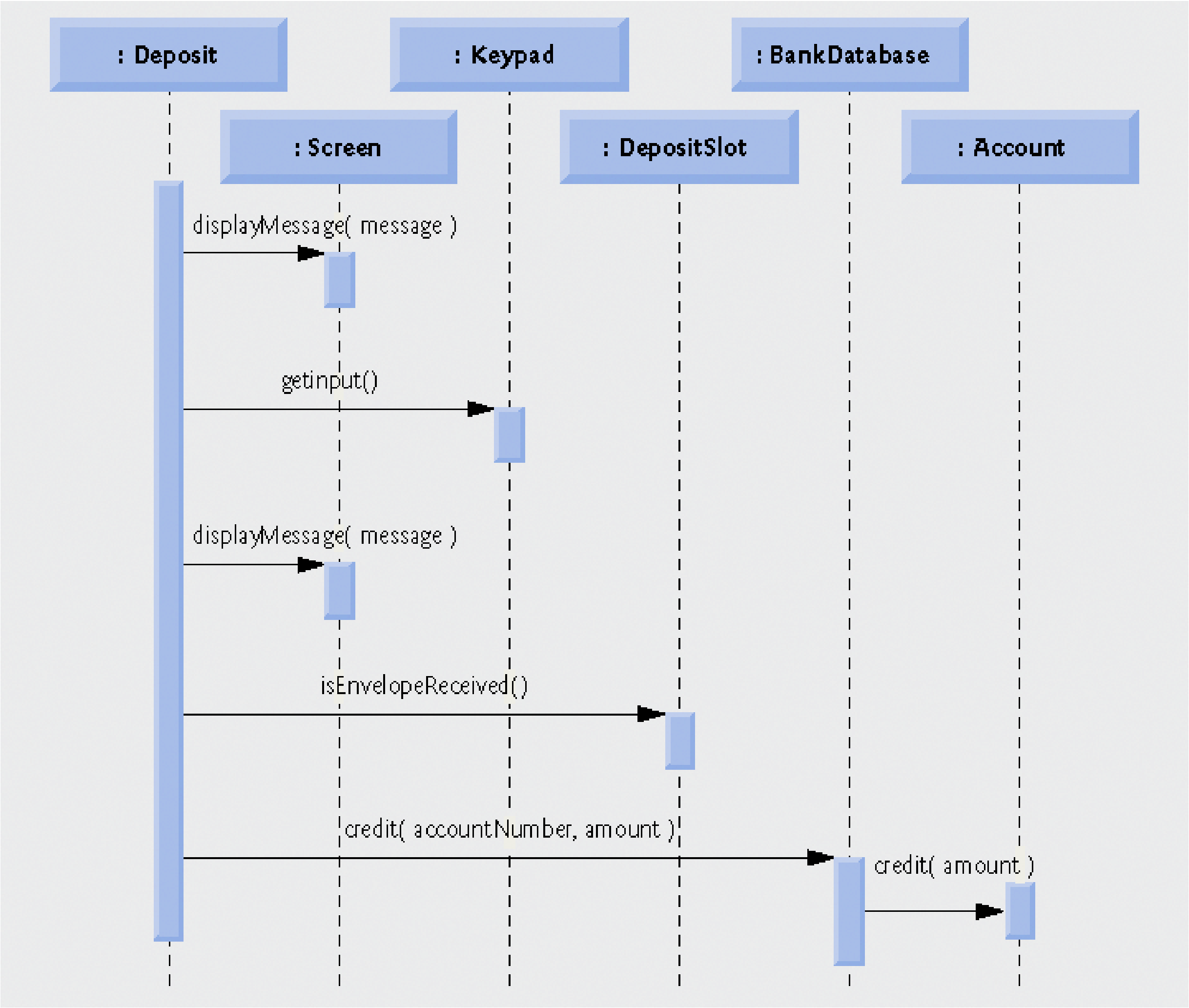


Fig. 7.29 | Sequence diagram that models a `Deposit` executing.