

4a

Arrays

OBJECTIVES

In this lecture 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.

- 4a.1 Introduction**
- 4a.2 Arrays**
- 4a.3 Declaring and Creating Arrays**
- 4a.4 Examples Using Arrays**
- 4a.5 Case Study: Card Shuffling and Dealing Simulation**
- 4a.6 Enhanced for Statement**
- 4a.7 Passing Arrays to Methods**
- 4a.8 Case Study: Class GradeBook Using an Array to Store Grades**
- 4a.9 Multidimensional Arrays**
- 4a.10 Case Study: Class GradeBook Using a Two-Dimensional Array**
- 4a.11 Variable-Length Argument Lists**
- 4a.12 Using Command-Line Arguments**
- 4a.13 Introduction to class Arrays and ArrayList**
- 4a.14 (Optional) GUI and Graphics Case Study: Drawing Arcs**
- 4a.15 Wrap-Up**

4a.1 Introduction

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

4a.2 Arrays

- **Array**
 - **Group of variables**
 - **Have same type**
 - **Reference type**

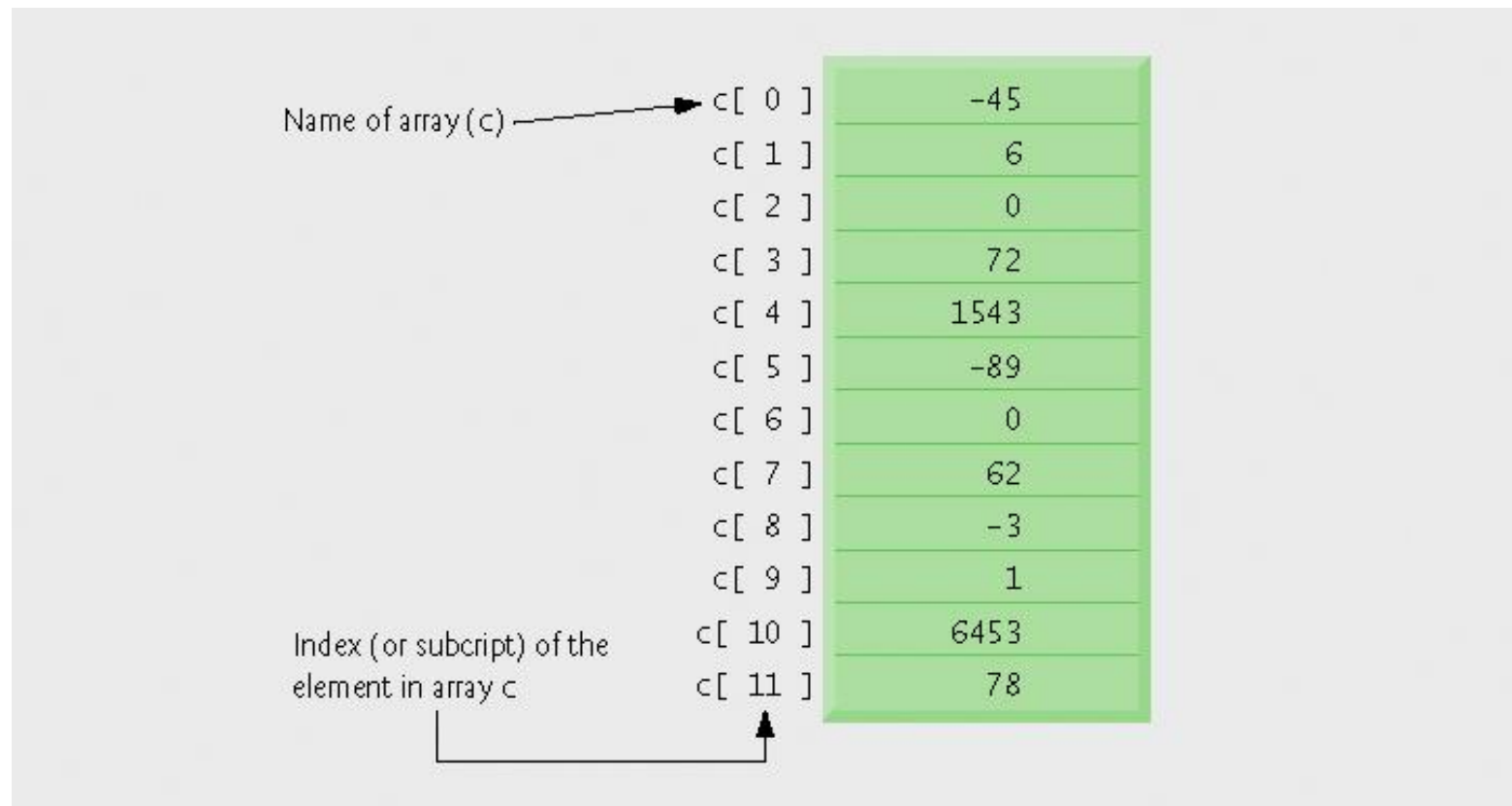


Fig. 4a.1 | A 12-element array.

4a.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 4a.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`.

4a.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`

4a.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 4a.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 4a.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 4a.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.

4a.4 Examples Using Arrays

- **Declaring arrays**
- **Creating arrays**
- **Initializing arrays**
- **Manipulating array elements**

4a.4 Examples Using Arrays

- **Creating and initializing an array**
 - **Declare array**
 - **Create array**
 - **Initialize array elements**

Outline

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

```

1 // Fig. 7.2: InitArray.java
2 // Creating an array.
3
4 public class InitArray
5 {
6     public static void main( String[] args )
7     {
8         int array[]; // declare array named array
9
10        array = new int[ 10 ]; // create the space for 10 ints
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

```

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

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



4a.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**

Outline

```

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 ints

Compiler uses initializer list to allocate array

InitArray.java

Line 9
Declare array as an array of ints

Line 9
Compiler uses initializer list to allocate array

Program output

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



4a.4 Examples Using Arrays (Cont.)

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

Outline

InitArray.java

```

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

are constant variable

Line 9
Declare and create array that contains 10 ints

Line 13
Use array index to assign array

Use array index to assign array value

Program output

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 4a.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 4a.4

Assigning a value to a constant after the variable has been initialized is a compilation error.

Common Programming Error 4a.5

Attempting to use a constant before it is initialized is a compilation error.

4a.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

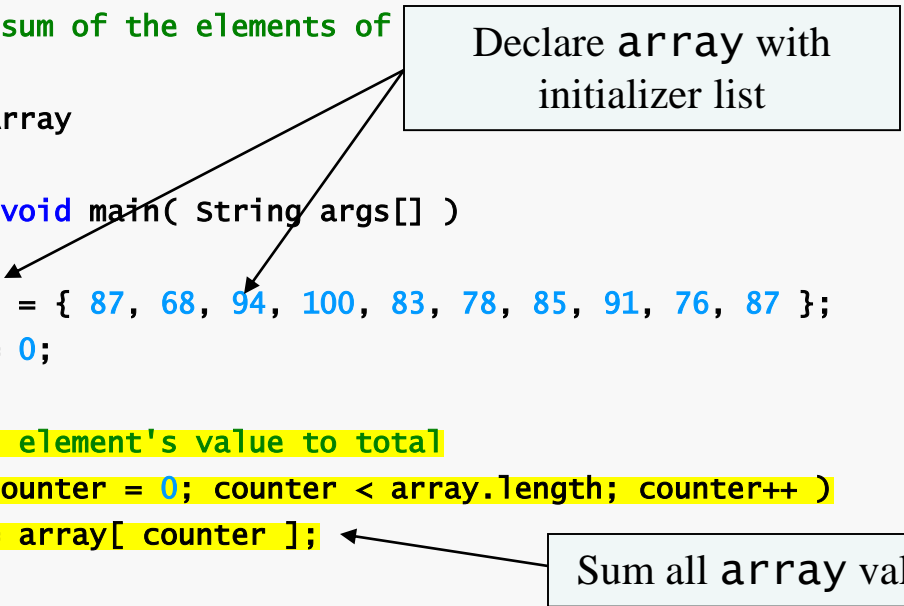
SumArray.java

Line 8
Declare array with
initializer list

Lines 12-13
Sum all array
values

Program output

```
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



4a.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**

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

associated
number of
asterisks

For each array element, print
associated number of asterisks

```

1 // Fig. 7.6: BarChart.java
2 // Bar chart printing program.
3
4 public class BarChart
5 {
6     public static void main( String args[] )
7     {
8         int array[] = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10        System.out.println( "Grade distribution:" );
11
12        // for each array element, output a bar of the chart
13        for ( int counter = 0; counter < array.length; counter++ )
14        {
15            // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
16            if ( counter == 10 )
17                System.out.printf( "%5d: ", 100 );
18            else
19                System.out.printf( "%02d-%02d: ",
20                                counter * 10, counter * 10 + 9 );
21
22            // print bar of asterisks
23            for ( int stars = 0; stars < array[ counter ]; stars++ )
24                System.out.print( "*" );
25
26            System.out.println(); // start a new line of output
27        } // end outer for
28    } // end main
29 } // end class BarChart
  
```

Declare array with
initializer list

Outline

BarChart.java

(2 of 2)

Program output

Grade distribution:

```
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
100: *
```



4a.4 Examples Using Arrays (Cont.)

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

Outline

RollDie.java

```

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 index
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

Line 10
Declare
frequency as
array of 7 ints
Lines 13-14
Generate 6000
random integers
in range 1-6
Line 14
Increment
frequency values
at index
associated with
random number

Program output

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



4a.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**

Outline

```

1 // Fig. 7.8: StudentPoll.java
2 // Poll analysis program.
3
4 public class StudentPoll
5 {
6     public static void main( String args[] )
7     {
8         // array of survey responses
9         int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
10             10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5,
11             4, 8, 6, 8, 10 };
12         int frequency[] = new int[ 11 ]; // array of frequency counters
13
14         // for each answer, select responses element and use that value
15         // as frequency index to determine element to increment
16         for ( int answer = 0; answer < responses.length; answer++ )
17             ++frequency[ responses[ answer ] ];
18
19         system.out.printf( "%s%10s", "Rating", "Frequency" );
20
21         // output each array element's value
22         for ( int rating = 1; rating < frequency.length; rating++ )
23             system.out.printf( "%d%10d", rating, frequency[ rating ] );
24     } // end main
25 } // end class StudentPoll

```

StudentPoll.java (1 of 2)

Declare responses as array to store 40 responses

Declare frequency as array of 11 int and ignore the first element

as array to store 40 responses

Line 12
Declare frequency as array of 11 int

For each response, increment frequency values at index associated with that response

response, increment frequency values at index associated with that response

Outline

StudentPoll.java

(2 of 2)

Program output

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3



Error-Prevention Tip 4a.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 4a.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.

4a.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

Outline

Card.java

Lines 17-20

```
1 // Fig. 7.9: Card.java
2 // Card class represents a playing card.
3
4 public class Card
5 {
6     private String face; // face of card ("Ace", "Deuce", ...)
7     private String suit; // suit of card ("Hearts", "Diamonds", ...)
8
9     // two-argument constructor initializes card's face and suit
10    public Card( String cardFace, String cardSuit )
11    {
12        face = cardFace; // initialize face of card
13        suit = cardSuit; // initialize suit of card
14    } // end two-argument Card constructor
15
16    // return String representation of Card
17    public String toString()
18    {
19        return face + " of " + suit;
20    } // end method toString
21 } // end class Card
```

Return the string
representation of a card



Outline

1	// Fig. 7.10: DeckOfCards.java		
2	// DeckOfCards class represents a deck of playing cards.		
3	import java.util.Random;		
4		Declare deck as array to store Card objects	
5	public class DeckOfCards		
6	{		
7	private Card deck[]; // array of Card objects	Constant NUMBER_OF_CARDS indicates the number of Cards in the deck	(1 of 2)
8	private int currentCard; // index of next Card to be dealt		
9	private final int NUMBER_OF_CARDS = 52; // constant number of Cards		Line 7
10	private Random randomNumbers; // random number generator		
11			
12	// constructor fills deck of Cards	Declare and initialize faces with Strings that represent the face of card	Line 9
13	public DeckOfCards()		
14	{		
15	String faces[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" }; // declare and initialize faces	Declare and initialize suits with Strings that represent the suit of card	Lines 15-16
16	String suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" }; // declare and initialize suits		Line 17
17			
18			
19	deck = new Card[NUMBER_OF_CARDS]; // create array of Card objects		Lines 24-26
20	currentCard = 0; // set currentCard so first Card dealt is deck[0]		
21	randomNumbers = new Random(); // create random number generator	Fill the deck array with Cards	
22			
23	// populate deck with Card objects		
24	for (int count = 0; count < deck.length; count++)		
25	deck[count] =		
26	new Card(faces[count % 13], suits[count / 13]);		
27	} // end DeckOfCards constructor		

Outline

DeckOfCards.java

(2 of 2)

```

28 // shuffle deck of cards with one-pass algorithm
29 public void shuffle()
30 {
31     // after shuffling, dealing should start at deck[ 0 ] again
32     currentCard = 0; // reinitialize currentCard
33
34     // for each Card, pick another random Card and swap them
35     for ( int first = 0; first < deck.length; first++ )
36     {
37         // select a random number between 0 and 51
38         int second = randomNumbers.nextInt( NUMBER_OF_CARDS );
39
40         // swap current Card with randomly selected Card
41         Card temp = deck[ first ];
42         deck[ first ] = deck[ second ];
43         deck[ second ] = temp;
44     } // end for
45 } // end method shuffle
46
47 // deal one Card
48 public Card dealCard()
49 {
50     // determine whether Cards remain to be dealt
51     if ( currentCard < deck.length )
52         return deck[ currentCard++ ]; // return current Card in array
53     else
54         return null; // return null to indicate that all Cards were dealt
55 } // end method dealCard
56 } // end class DeckOfCards

```

Swap current Card with
randomly selected Card

-44

Line 52

Determine whether
deck is empty



Outline

DeckOfCardsTest .java

(1 of 2)

```
1 // Fig. 7.11: DeckOfCardsTest.java
2 // Card shuffling and dealing application.
3
4 public class DeckOfCardsTest
5 {
6     // execute application
7     public static void main( String args[] )
8     {
9         DeckOfCards myDeckOfCards = new DeckOfCards();
10        myDeckOfCards.shuffle(); // place cards in random order
11
12        // print all 52 cards in the order in which they are dealt
13        for ( int i = 0; i < 52; i++ )
14        {
15            // deal and print 4 cards
16            System.out.printf( "%-20s%-20s%-20s%-20s\n",
17                myDeckOfCards.dealCard(), myDeckOfCards.dealCard(),
18                myDeckOfCards.dealCard(), myDeckOfCards.dealCard() );
19        } // end for
20    } // end main
21 } // end class DeckOfCardsTest
```



Outline

DeckOfCardsTest

.java

(2 of 2)

Six of Spades
Queen of Hearts
Three of Diamonds
Four of Spades
Three of Clubs
King of Clubs
Queen of Clubs
Three of Spades
Ace of Spades
Deuce of Spades
Jack of Hearts
Ace of Diamonds
Five of Diamonds

Eight of Spades
Seven of Clubs
Deuce of Clubs
Ace of Clubs
Deuce of Hearts
Ten of Hearts
Eight of Diamonds
King of Diamonds
Four of Diamonds
Eight of Hearts
Seven of Spades
Queen of Diamonds
Ten of Clubs

Six of Clubs
Nine of Spades
Ace of Hearts
Seven of Diamonds
Five of Spades
Three of Hearts
Deuce of Diamonds
Nine of Clubs
Seven of Hearts
Five of Hearts
Four of Clubs
Five of Clubs
Jack of Spades

Nine of Hearts
King of Hearts
Ten of Spades
Four of Hearts
Jack of Diamonds
Six of Diamonds
Ten of Diamonds
Six of Hearts
Eight of Clubs
Queen of Spades
Nine of Diamonds
King of Spades
Jack of Clubs



4a.6 Enhanced for Statement

- **Enhanced for statement**
 - Iterates through elements of an array or a collection without using a counter
 - Syntax

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

Outline

EnhancedForTest
.java

```
1 // Fig. 7.12: EnhancedForTest.java
2 // Using enhanced for statement to total integers in an array.
3
4 public class EnhancedForTest
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 number : array )
13             total += number;
14
15         System.out.printf( "Total of array elements: %d\n", total );
16     } // end main
17 } // 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



4a.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**

4a.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

PassArray.java

(1 of 2)

Line 9

Line 19

Declare 5-int array
with initializer list

Pass entire array to method
modifyArray

```

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 ] );

```



Outline

```

29     modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
30     System.out.printf(
31         "array[3] after modifyElement
32     } // end main
33
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

Line 30

Lines 36-40

Lines 43-48

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



4a.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**
 - **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 4a.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.

4a.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

Outline

GradeBook.java

(1 of 5)

Line 7

Line 13

Declare array **grades** to
store individual grades

Assign the array's reference
to instance variable **grades**

```
1 // Fig. 7.14: GradeBook.java
2 // Grade book using an array to store test grades.
3
4 public class GradeBook
5 {
6     private String courseName; // name of course this GradeBook represents
7     private int grades[]; // array of student grades
8
9     // two-argument constructor initializes courseName
10    public GradeBook( String name, int gradesArray[] )
11    {
12        courseName = name; // initialize courseName
13        grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19        courseName = name; // store the course name
20    } // end method setCourseName
21
22    // method to retrieve the course name
23    public String getCourseName()
24    {
25        return courseName;
26    } // end method getCourseName
27
```



Outline

GradeBook.java

(2 of 5)

```
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 method getAverage to calculate the average grade
43     System.out.printf( "\nClass average is %.2f\n", getAverage() );
44
45     // call methods getMinimum and getMaximum
46     System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n",
47         getMinimum(), getMaximum() );
48
49     // call outputBarChart to print grade distribution chart
50     outputBarChart();
51 } // end method processGrades
52
53 // find minimum grade
54 public int getMinimum()
55 {
56     int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest
57
```



Outline

gradeBook.java

(3 of 5)

Lines 59-64

Lines 75-80

Loop through grades to
find the lowest grade

Loop through grades to
find the highest grade

```
58 // loop through grades array
59 for ( int grade : grades )
60 {
61     // if grade lower than lowGrade, assign it to lowGrade
62     if ( grade < lowGrade )
63         lowGrade = grade; // new lowest grade
64 } // end for

65
66 return lowGrade; // return lowest grade
67 } // end method getMinimum
68
69 // find maximum grade
70 public int getMaximum()
71 {
72     int highGrade = grades[ 0 ]; // assume grades[ 0 ] is largest
73
74     // loop through grades array
75     for ( int grade : grades )
76     {
77         // if grade greater than highGrade, assign it to highGrade
78         if ( grade > highGrade )
79             highGrade = grade; // new highest grade
80     } // end for

81
82     return highGrade; // return highest grade
83 } // end method getMaximum
84
```

Outline

GradeBook.java

(4 of 5)

Lines 91-92

Lines 107-108

```
85 // determine average grade for test
86 public double getAverage()
87 {
88     int total = 0; // initialize total
89
90     // sum grades for one student
91     for ( int grade : grades )
92         total += grade;
93
94     // return average of grades
95     return (double) total / grades.length;
96 } // end method getAverage
97
98 // output bar chart displaying grade distribution
99 public void outputBarChart()
100 {
101     System.out.println( "Grade distribution:" );
102
103     // stores frequency of grades in each range of 10 grades
104     int frequency[] = new int[ 11 ];
105
106     // for each grade, increment the appropriate frequency
107     for ( int grade : grades )
108         ++frequency[ grade / 10 ];
109 }
```

Loop through **grades** to
sum grades for one student

Loop through **grades** to
calculate frequency



Outline

GradeBook.java

(5 of 5)

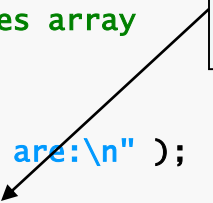
Lines 134-136

```

110 // for each grade frequency, print bar in chart
111 for ( int count = 0; count < frequency.length; count++ )
112 {
113     // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
114     if ( count == 10 )
115         System.out.printf( "%5d: ", 100 );
116     else
117         System.out.printf( "%02d-%02d: ",
118             count * 10, count * 10 + 9 );
119
120     // print bar of asterisks
121     for ( int stars = 0; stars < frequency[ count ]; stars++ )
122         System.out.print( "*" );
123
124     System.out.println(); // start a new line of output
125 } // end outer for
126 } // end method outputBarChart
127
128 // output the contents of the grades array
129 public void outputGrades()
130 {
131     System.out.println( "The grades are:\n" );
132
133     // output each student's grade
134     for ( int student = 0; student < grades.length; student++ )
135         System.out.printf( "Student %2d: %3d\n",
136             student + 1, grades[ student ] );
137 } // end method outputGrades
138 } // end class GradeBook

```

Loop through grades to
display each grade




Software Engineering Observation 4a.1

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.

Outline

```
1 // Fig. 7.15: GradeBookTest.java
2 // Creates GradeBook object using an array of grades.
3
4 public class GradeBookTest
5 {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9         // array of student grades
10        int gradesArray[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
11
12        GradeBook myGradeBook = new GradeBook(
13            "CS101 Introduction to Java Programming", gradesArray );
14        myGradeBook.displayMessage();
15        myGradeBook.processGrades();
16    } // end main
17 } // end class GradeBookTest
```

Declare and initialize
gradesArray with 10 elements

.java

(1 of 2)

Line 10

Line 13

Pass gradesArray to
GradeBook constructor



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: *
```

Outline

GradeBookTest

.java

(2 of 2)

Program output



4a.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	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
Row 1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
Row 2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>

Diagram illustrating a 2D array structure with three rows and four columns. The array is represented as a grid of elements, each labeled with its row and column indices (e.g., `a[0][0]` for the first element). The columns are labeled Column 0, Column 1, Column 2, and Column 3. The rows are labeled Row 0, Row 1, and Row 2. Arrows point from the labels 'Column index', 'Row index', and 'Array name' to the corresponding parts of the element `a[2][1]` in the grid.

Fig. 4a.16 | Two-dimensional array with three rows and four columns.

4a.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

4a.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 } };`

4a.9 Multidimensional Arrays (Cont.)

- **Creating two-dimensional arrays with array-creation expressions**

- **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
```

Outline

InitArray.java

```
1 // Fig. 7.17: InitArray.java
2 // Initializing two-dimensional arrays.
3
4 public class InitArray
5 {
6     // create and output two-dimensional arrays
7     public static void main( String args[] )
8     {
9         int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
10        int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
11
12        System.out.println( "Values in array1 by row are" );
13        outputArray( array1 ); // displays array1 by row
14
15        System.out.println( "\nValues in array2 by row are" );
16        outputArray( array2 ); // displays array2 by row
17    } // end main
18
```

Use nested array initializers
to initialize array1

Use nested array initializers
of different lengths to
initialize array2

1 of 2)

Line 9

Line 10




```

19 // output rows and columns of a two-dimensional array
20 public static void outputArray( int array[][] )
21 {
22     // loop through array's rows
23     for ( int row = 0; row < array.length; row++ )
24     {
25         // loop through columns of current row
26         for ( int column = 0; column < array[ row ].length; column++ )
27             system.out.printf( "%d ", array[ row ][ column ] );
28
29         system.out.println(); // start new line of output
30     } // end outer for
31 } // end method outputArray
32 } // end class InitArray

```

array[row].length returns number of columns associated with row subscript

InitArray.java

(2 of 2)

Line 26

Use double-bracket notation to access two-dimensional array values

Values in array1 by row are

```

1 2 3
4 5 6

```

Values in array2 by row are

```

1 2
3
4 5 6

```

Program output



4a.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;
```

4a.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

GradeBook.java

(1 of 7)

Line 7

Line 10

```
1 // Fig. 7.18: GradeBook.java
2 // Grade book using a two-dimensional array to store grades.
3
4 public class GradeBook
5 {
6     private String courseName; // name of course this grade book represents
7     private int grades[][]; // two-dimensional array of student grades
8
9     // two-argument constructor initializes courseName and grades array
10    public GradeBook( String name, int gradesArray[][])
11    {
12        courseName = name; // initialize courseName
13        grades = gradesArray; // store grades
14    } // end two-argument GradeBook constructor
15
16    // method to set the course name
17    public void setCourseName( String name )
18    {
19        courseName = name; // store the course name
20    } // end method setCourseName
21
22    // method to retrieve the course name
23    public String getCourseName()
24    {
25        return courseName;
26    } // end method getCourseName
27
```

Declare two-dimensional array grades

GradeBook constructor
accepts a **String** and a
two-dimensional array



Outline

GradeBook.java

(2 of 7)

```
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\n%s %d\n\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 ];
56
```



Outline

GradeBook.java

(3 of 7)

Lines 58-67

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

```
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 lowGrade
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 }
```

Outline

GradeBook.java

(4 of 7)

Lines 79-88

Lines 94-104

```

78 // loop through rows of grades array
79 for ( int studentGrades[] : grades )
80 {
81     // loop through columns of current row
82     for ( int grade : studentGrades )
83     {
84         // if grade greater than highGrade
85         if ( grade > highGrade )
86             highGrade = grade;
87     } // end inner for
88 } // end outer for
89
90 return highGrade; // return highest grade
91 } // end method getMaximum
92
93 // determine average grade for particular set of grades
94 public double getAverage( int setOfGrades[] )
95 {
96     int total = 0; // initialize total
97
98     // sum grades for one student
99     for ( int grade : setOfGrades )
100         total += grade;
101
102     // return average of grades
103     return (double) total / setOfGrades.length;
104 } // end method getAverage
105

```

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

Calculate a particular student's semester average



Outline

GradeBook.java

(5 of 7)

Lines 115-119

```

106 // output bar chart displaying overall grade distribution
107 public void outputBarChart()
108 {
109     System.out.println( "Overall grade distribution:" );
110
111     // stores frequency of grades in each range of 10 grades
112     int frequency[] = new int[ 11 ];
113
114     // for each grade in GradeBook, increment the appropriate frequency
115     for ( int studentGrades[] : grades )
116     {
117         for ( int grade : studentGrades )
118             ++frequency[ grade / 10 ];
119     } // end outer for
120
121     // for each grade frequency, print bar in chart
122     for ( int count = 0; count < frequency.length; count++ )
123     {
124         // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
125         if ( count == 10 )
126             System.out.printf( "%5d: ", 100 );
127         else
128             System.out.printf( "%02d-%02d: ",
129                                 count * 10, count * 10 + 9 );
130
131         // print bar of asterisks
132         for ( int stars = 0; stars < frequency[ count ]; stars++ )
133             System.out.print( "*" );

```

Calculate the distribution of
all student grades

Outline

GradeBook.java

(6 of 7)

```
134         System.out.println(); // start a new line of output
135     } // end outer for
136 } // end method outputBarChart
137
138
139 // output the contents of the grades array
140 public void outputGrades()
141 {
142     System.out.println( "The grades are:\n" );
143     System.out.print( "          " ); // align column heads
144
145     // create a column heading for each of the tests
146     for ( int test = 0; test < grades[ 0 ].length; test++ )
147         System.out.printf( "Test %d ", test + 1 );
148
149     System.out.println( "Average" ); // student average column heading
150
151     // create rows/columns of text representing array grades
152     for ( int student = 0; student < grades.length; student++ )
153     {
154         System.out.printf( "Student %2d", student + 1 );
155
156         for ( int test : grades[ student ] ) // output student's grades
157             System.out.printf( "%8d", test );
158     }
```



```
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
```

Outline

GradeBook.java

(7 of 7)



Outline

```
1 // Fig. 7.19: GradeBookTest.java
2 // Creates GradeBook object using a two-dimensional array of grades.
3
4 public class GradeBookTest
5 {
6     // main method begins program execution
7     public static void main( String args[] )
8     {
9         // two-dimensional array of student grades
10        int gradesArray[][] = { { 87, 96, 70 },
11                                { 68, 87, 90 },
12                                { 94, 100, 90 },
13                                { 100, 81, 82 },
14                                { 83, 65, 85 },
15                                { 78, 87, 65 },
16                                { 85, 75, 83 },
17                                { 91, 94, 100 },
18                                { 76, 72, 84 },
19                                { 87, 93, 73 } };
20
21        GradeBook myGradeBook = new GradeBook(
22            "CS101 Introduction to Java Programming" );
23        myGradeBook.displayMessage();
24        myGradeBook.processGrades();
25    } // end main
26 } // end class GradeBookTest
```

Declare `gradesArray` as 10-by-3 array

.java

(1 of 2)

Lines 10-19

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



Outline

GradeBookTest

.java

(2 of 2)

Program output

Welcome to the grade book for
CS101 Introduction to Java Programming!

The grades are:

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

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: ***



4a.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

Outline

VarargsTest .java

(1 of 2)

Line 7

Lines 12-13

Line 15

```
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 enhanced for loop
12         for ( double d : numbers )
13             total += d;
14
15         return total / numbers.length;
16     } // end method average
17
18     public static void main( String args[] )
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 **doubles**

Calculate the total of the **doubles** in the array

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

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 d3 is %.1f\n",
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

```

Invoke method average with
two arguments

Invoke method average with
three arguments

Invoke method average with
four arguments

```

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 4a.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.

4a.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]`

Outline

InitArray.java

(1 of 2)

Line 6

Line 9

Line 16

Lines 20-21

Lines 24-25

```

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
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 command-line arguments
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.length; counter++ )
25                     array[ counter ] = initialValue + increment * counter;
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

(2 of 2)

Program output

```

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

```

java InitArray

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

java Init

Missing command-line arguments

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



4a.13 Introduction to class Arrays and ArrayList

Class Arrays

- The **java.util.Arrays** class contains various static methods for sorting and searching arrays, comparing arrays, filling array elements, and returning a string representation of the array. These methods are overloaded for all primitive types
- This class is a member of the **Java Collections Framework**.

4a.13 Introduction to class Arrays and ArrayList

You can use the **sort** or **parallelSort** method to sort a whole array or a partial array. For example, the following code **sorts an array of numbers** and an **array of characters**.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers); // Sort the whole array  
java.util.Arrays.parallelSort(numbers); // Sort the whole array  
  
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};  
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array  
java.util.Arrays.parallelSort(chars, 1, 3); // Sort part of the array
```

4a.13 Introduction to class Arrays and ArrayList

Invoking `sort(numbers)` sorts the whole array `numbers`. Invoking `sort(chars, 1, 3)` sorts a partial array from `chars[1]` to `chars[3-1]`.

```
public static void sort(int[] a,  
                        int fromIndex,  
                        int toIndex) .
```

The range to be sorted extends from the index `fromIndex`, **inclusive**, to the index `toIndex`, **exclusive**. If `fromIndex == toIndex`, the range to be sorted is empty.

4a.13 Introduction to class Arrays and ArrayList

Java SE 8—Class Arrays Method parallelSort

- ▶ The Arrays class now has several new “parallel” methods that take advantage of multi-core hardware.
- ▶ Arrays method parallelSort can sort large arrays more efficiently on multi-core systems.

parallelSort is **more efficient** if your computer has multiple processors

4a.13 Introduction to class Arrays and ArrayList

- You can use the `binarySearch` method to search for a key in an array. **The array must be presorted in increasing order.**
- If the **key is not in the array**, the method returns **– (insertionIndex + 1)** . For example, the following code **searches the keys** in an array of integers and an array of characters

4a.13 Introduction to class Arrays and ArrayList

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};  
System.out.println("1. Index is " +  
    java.util.Arrays.binarySearch(list, 11));  
System.out.println("2. Index is " +  
    java.util.Arrays.binarySearch(list, 12));  
  
char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};  
System.out.println("3. Index is " +  
    java.util.Arrays.binarySearch(chars, 'a'));  
System.out.println("4. Index is " +  
    java.util.Arrays.binarySearch(chars, 't'));
```

The output of this code is

- 1. Index is 4**
- 2. Index is -6**
- 3. Index is 0**
- 4. Index is -4**

4a.13 Introduction to class Arrays and ArrayList

You can use the **equals** method to check whether two arrays are strictly equal. Two arrays are **strictly equal** if their corresponding elements are the same. In the following code, **list1** and **list2** are equal, but **list2** and **list3** are not.

```
int[] list1 = {2, 4, 7, 10};  
int[] list2 = {2, 4, 7, 7, 7, 10};  
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array  
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array
```

4a.13 Introduction to class Arrays and ArrayList

You can use the **fill** method to fill in all or part of the array. For example, the following code fills **list1** with **5** and fills **8** into elements **list2[1]** through **list2[5-1]**.

```
int[] list1 = {2, 4, 7, 10};  
int[] list2 = {2, 4, 7, 7, 7, 10};  
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array  
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array
```

You can easily **output the elements of an array** as

```
String res = Arrays.toString(list1);  
System.out.println(res);
```

Output: [2, 4, 7, 10]

4a.13 Introduction to class Arrays and ArrayList

Class ArrayList

An **ArrayList** object can be used to store a list of objects. You can create an array to store objects. But, once the array is created, its size is fixed. Java provides the **ArrayList**. class, which can be used to store an unlimited number of objects. Some methods in **ArrayList** are displayed on the following slide

This class is a member of the **Java Collections Framework**.

4a.13 Introduction to class Arrays and ArrayList

`java.util.ArrayList<E>`

```
+ArrayList()  
+add(o: E): void  
+add(index: int, o: E): void  
+clear(): void  
+contains(o: Object): boolean  
+get(index: int): E  
+indexOf(o: Object): int  
+isEmpty(): boolean  
+lastIndexOf(o: Object): int  
+remove(o: Object): boolean  
  
+size(): int  
+remove(index: int): boolean  
  
+set(index: int, o: E): E
```

Creates an empty list.

Appends a new element `o` at the end of this list.

Adds a new element `o` at the specified index in this list.

Removes all the elements from this list.

Returns true if this list contains the element `o`.

Returns the element from this list at the specified index.

Returns the index of the first matching element in this list.

Returns true if this list contains no elements.

Returns the index of the last matching element in this list.

Removes the first element `o` from this list. Returns true if an element is removed.

Returns the number of elements in this list.

Removes the element at the specified index. Returns true if an element is removed.

Sets the element at the specified index.

4a.13 Introduction to class Arrays and ArrayList

ArrayList is known as a generic class with a generic type **E**. You can specify a concrete type to replace **E** when creating an **ArrayList**.

For example, the following statement creates an **ArrayList** and assigns its reference to variable **cities**. This **ArrayList** object can be used to store strings.

```
ArrayList<String> cities =  
    new ArrayList<String>();
```



4a.13 Introduction to class Arrays and ArrayList

The following statement creates an **ArrayList** and assigns its reference to variable **dates**. This **ArrayList** object can be used to store dates.

```
ArrayList<java.util.Date> dates =  
    new ArrayList<java.util.Date> ();  
ArrayList<Number> nums =  
    new ArrayList<>(Arrays.asList(2,3,1,5));  
Collections.sort(nums);  
int p = scan.nextInt();  
int index = Collections.binarySearch(nums,p);
```

4a.13 Introduction to class Arrays and ArrayList

Note:

Since JDK 7, the statement

```
ArrayList<AConcreteType> list = new  
ArrayList<AConcreteType>();
```

can be simplified by means of the **diamond operator** <>

```
ArrayList<AConcreteType> list =  
    new ArrayList<>();
```

The concrete type is no longer required in the constructor thanks to a feature called ***type inference***. The compiler is able to infer the type from the variable declaration

4a.13 Introduction to class Arrays and ArrayList

Note:

```
ArrayList<Integer> i = new ArrayList<Integer>();  
i.add(0);  
i.add(2);  
i.add(1);  
i.add(3);  
System.out.println(i.toString());  
i.remove(new Integer(1));  
System.out.println(i.toString());  
i.remove(1);  
System.out.println(i.toString());
```

[0, 2, 1, 3]

[0, 2, 3]

[0, 3]

4a.13 Introduction to class Arrays and ArrayList

Differences and Similarities between Arrays and ArrayList

Operation	Array	ArrayList
Creating an array/ArrayList	<code>String[] a = new String[10]</code>	<code>ArrayList<String> list = new ArrayList<>();</code>
Accessing an element	<code>a[index]</code>	<code>list.get(index);</code>
Updating an element	<code>a[index] = "London";</code>	<code>list.set(index, "London");</code>
Returning size	<code>a.length</code>	<code>list.size();</code>
Adding a new element		<code>list.add("London");</code>
Inserting a new element		<code>list.add(index, "London");</code>
Removing an element		<code>list.remove(index);</code>
Removing an element		<code>list.remove(Object);</code>
Removing all elements		<code>list.clear();</code>

4a.13 Introduction to class Arrays and ArrayList

```
public class TestArrayList {  
    public static void main(String[] args) {  
        // Create a list to store cities  
        ArrayList<String> cityList = new ArrayList<>();  
  
        // Add some cities in the list  
        cityList.add("London");  
        // cityList now contains [London]  
        cityList.add("Denver");  
        // cityList now contains [London, Denver]  
        cityList.add("Paris");  
        // cityList now contains [London, Denver, Paris]  
        cityList.add("Miami");  
        // cityList now contains [London, Denver, Paris, Miami]  
        cityList.add("Seoul");  
        // Contains [London, Denver, Paris, Miami, Seoul]  
        cityList.add("Tokyo");  
        // Contains [London, Denver, Paris, Miami, Seoul, Tokyo]  
    }  
}
```

4a.13 Introduction to class Arrays and ArrayList

```
public class TestArrayList {  
    public static void main(String[] args) {  
        // Create a list to store cities  
        ArrayList<String> cityList = new ArrayList<>();  
  
        // Add some cities in the list  
        cityList.add("London");  
        // cityList now contains [London]  
        cityList.add("Denver");  
        // cityList now contains [London, Denver]  
        cityList.add("Paris");  
        // cityList now contains [London, Denver, Paris]  
        cityList.add("Miami");  
        // cityList now contains [London, Denver, Paris, Miami]  
        cityList.add("Seoul");  
        // Contains [London, Denver, Paris, Miami, Seoul]  
        cityList.add("Tokyo");  
        // Contains [London, Denver, Paris, Miami, Seoul, Tokyo]
```

4a.13 Introduction to class Arrays and ArrayList

```
// Contains [London, Denver, Paris, Miami, Seoul, Tokyo]

System.out.println("List size? " + cityList.size());
System.out.println("Is Miami in the list? " +
    cityList.contains("Miami"));
System.out.println("The location of Denver in the list? "
    + cityList.indexOf("Denver"));
System.out.println("Is the list empty? " +
    cityList.isEmpty()); // Print false

// Insert a new city at index 2
cityList.add(2, "Xian");
// Contains [London, Denver, Xian, Paris, Miami, Seoul, Tokyo]

// Remove a city from the list
cityList.remove("Miami");
// Contains [London, Denver, Xian, Paris, Seoul, Tokyo]
```

4a.13 Introduction to class Arrays and ArrayList

```
// Remove a city at index 1
cityList.remove(1);
// Contains [London, Xian, Paris, Seoul, Tokyo]

// Display the contents in the list
System.out.println(cityList.toString());

// Display the contents in the list in reverse order
for (int i = cityList.size() - 1; i >= 0; i--)
    System.out.print(cityList.get(i) + " ");
System.out.println();

// Create a list to store two circles
ArrayList<CircleFromSimpleGeometricObject> list
    = new ArrayList<>();

// Add two circles
list.add(new CircleFromSimpleGeometricObject(2));
list.add(new CircleFromSimpleGeometricObject(3));

// Display the area of the first circle in the list
System.out.println("The area of the circle? " +
    list.get(0).getArea());
}
```

4a.14 (Optional) GUI and Graphics Case Study: Drawing Arcs

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

Outline

DrawRainbow

.java

(1 of 2)

Line 22

```
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

(2 of 2)

Lines 43-45

```
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
```

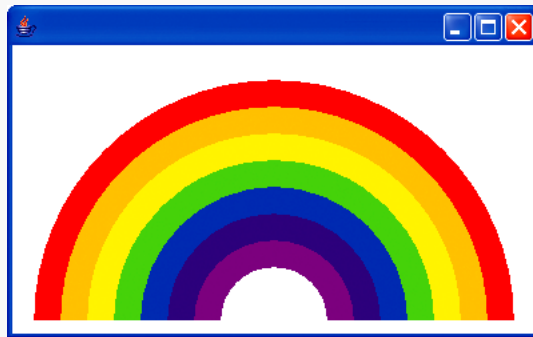


Draw a filled semicircle

Outline

DrawRainbowTest .java

```
1 // Fig. 7.23: DrawRainbowTest.java
2 // Test application to display a rainbow.
3 import javax.swing.JFrame;
4
5 public class DrawRainbowTest
6 {
7     public static void main( String args[] )
8     {
9         DrawRainbow panel = new DrawRainbow();
10        JFrame application = new JFrame();
11
12        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
13        application.add( panel );
14        application.setSize( 400, 250 );
15        application.setVisible( true );
16    } // end main
17 } // end class DrawRainbowTest
```



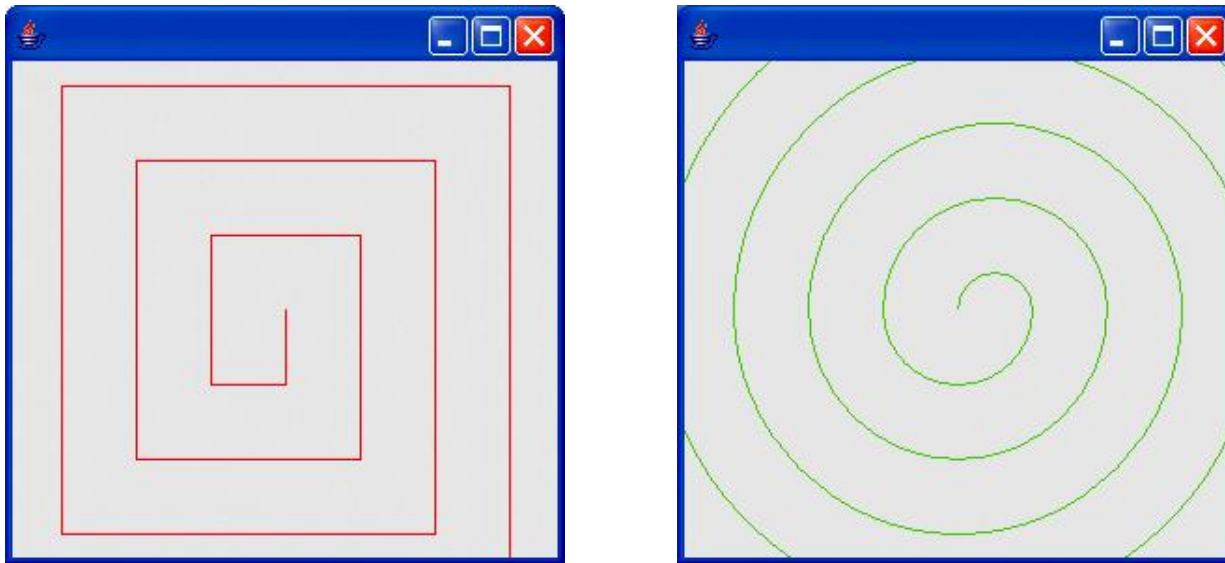


Fig. 4a.24 | Drawing a spiral using `drawLine` (left) and `drawArc` (right).