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.

Outline

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**

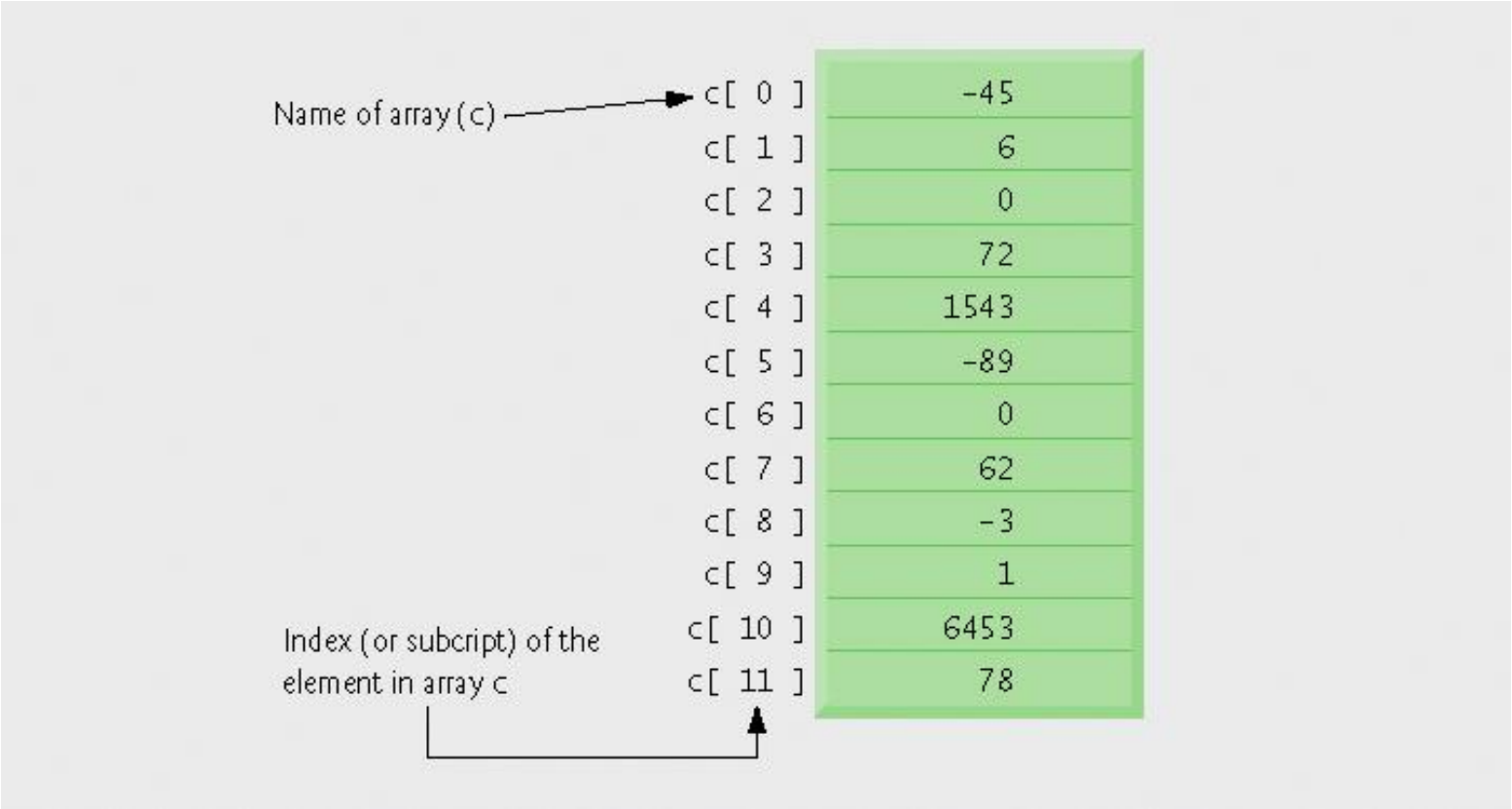


Fig. 4a.1 | A 12-element array.

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

```
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 `int`s

Create 10 `int`s 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 **int**s

Line 10
Create 10 **int**s 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

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

itArray.java

Line 9
Declare array as an array of ints

Line 9
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
```

Program output

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

```
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 `int`s

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 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

```
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

4a.4 Examples Using Arrays (Cont.)

Output arrays

Using Java 8 you can do this in a very clean way:

```
String.join(delimiter, elements);
```

This works in three ways:

1) directly specifying the elements

```
String joined1 = String.join(",", "a", "b", "c");
```

2) using arrays

```
String[] array = new String[] { "a", "b", "c" };
String joined2 = String.join(",", array);
```

3) using iterables

```
List<String> list = Arrays.asList(array);
String joined3 = String.join(",", list);
```

4a.4 Examples Using Arrays (Cont.)

Output arrays

In the general case, use:

```
String joined4 = String.join(delimiter,
                             Arrays.toString(array));
```

For example:

```
int[] array = new int[]{1,2,3};

System.out.println(String.join(“, “,
                             Arrays.toString(array));
```

Output:

```
[1, 2, 3]
```

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**

```
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

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

For each `array` element, print associated number of asterisks

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

associated number of asterisks

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

Outline

BarChart.java

(2 of 2)

Program output

4a.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 `int`s

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

Lines 13-14
Generate 6000 random integers in range 1-6

Line 14
Increment frequency values at index associated with random number

Program output

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**

```
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
```

Declare `responses` as array to store 40 responses

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

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

Outline

StudentPoll.java

(1 of 2)

Line 9
Declare responses as array to store 40 responses

Line 12
Declare frequency as array of 11 int and ignore the element

Line 17
For each response, increment frequency values at index associated with that response

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

Outline

`StudentPoll.java`

(2 of 2)

Program output

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
5  public class DeckOfCards
6  {
7     private Card deck[]; // array of Card objects
8     private int currentCard; // index of next Card to be dea
9     private final int NUMBER_OF_CARDS = 52; // constant number of Cards
10    private Random randomNumbers; // random number generator
11
12    // constructor fills deck of Cards
13    public DeckOfCards()
14    {
15       String faces[] = { "Ace", "Deuce
16          "Seven", "Eight", "Nine", "Te
17       String suits[] = { "Hearts", "Di
18
19       deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
20       currentCard = 0; // set currentCard so first Card dealt is deck[ 0 ]
21       randomNumbers = new Random(); // create random nu
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
```

Declare `deck` as array to store `Card` objects

Constant `NUMBER_OF_CARDS` indicates the number of Cards in the deck

Declare and initialize `faces` with `String`s that represent the face of card

Declare and initialize `suits` with `String`s that represent the suit of card

Fill the `deck` array with `Card`s

(1 of 2)

Line 7

Line 9

Lines 15-16

Line 17

Lines 24-26

Outline

DeckOfCards.java

(2 of 2)

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

Swap current `Card` with randomly selected `Card`

Line 52

Determine whether `deck` is empty

```java
// Fig. 7.11: DeckOfCardsTest.java
// Card shuffling and dealing application.

public class DeckOfCardsTest
{
   // execute application
   public static void main( String args[] )
   {
      DeckOfCards myDeckOfCards = new DeckOfCards();
      myDeckOfCards.shuffle(); // place Cards in random order

      // print all 52 Cards in the order in which they are dealt
      for ( int i = 0; i < 13; i++ )
      {
         // deal and print 4 Cards
         System.out.printf( "%-20s%-20s%-20s%-20s\n",
            myDeckOfCards.dealCard(), myDeckOfCards.dealCard(),
            myDeckOfCards.dealCard(), myDeckOfCards.dealCard() );
      } // end for
   } // end main
} // end class DeckOfCardsTest
```

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


DeckOfCardsTest.java

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

```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
```

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

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

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`**

```
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`

Outline

`PassArray.java`

(1 of 2)

Line 9

Line 19

Outline

```
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

ava

(2 of 2)

Line 30

Lines 36-40

Lines 43-48

```
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
```

Program output

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

```
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
```

Declare array `grades` to store individual grades

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

Outline

GradeBook.java

(2 of 5)

```
   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // getCourseName gets the name of the course
      System.out.printf( "Welcome to the grade book for\n%s!\n\n",
         getCourseName() );
   } // end method displayMessage

   // perform various operations on the data
   public void processGrades()
   {
      // output grades array
      outputGrades();

      // call method getAverage to calculate the average grade
      System.out.printf( "\nClass average is %.2f\n", getAverage() );

      // call methods getMinimum and getMaximum
      System.out.printf( "Lowest grade is %d\nHighest grade is %d\n\n",
         getMinimum(), getMaximum() );

      // call outputBarChart to print grade distribution chart
      outputBarChart();
   } // end method processGrades

   // find minimum grade
   public int getMinimum()
   {
      int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest

```

```
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
```

Loop through `grades` to find the lowest grade

Loop through `grades` to find the highest grade

Outline

radeBook.java

(3 of 5)

Lines 59-64

Lines 75-80

```
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`

(4 of 5)

Lines 91-92

Lines 107-108

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.

```
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

Pass `gradesArray` to `GradeBook` constructor

Outline

eBookTest

.java

(1 of 2)

Line 10

Line 13

```
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**

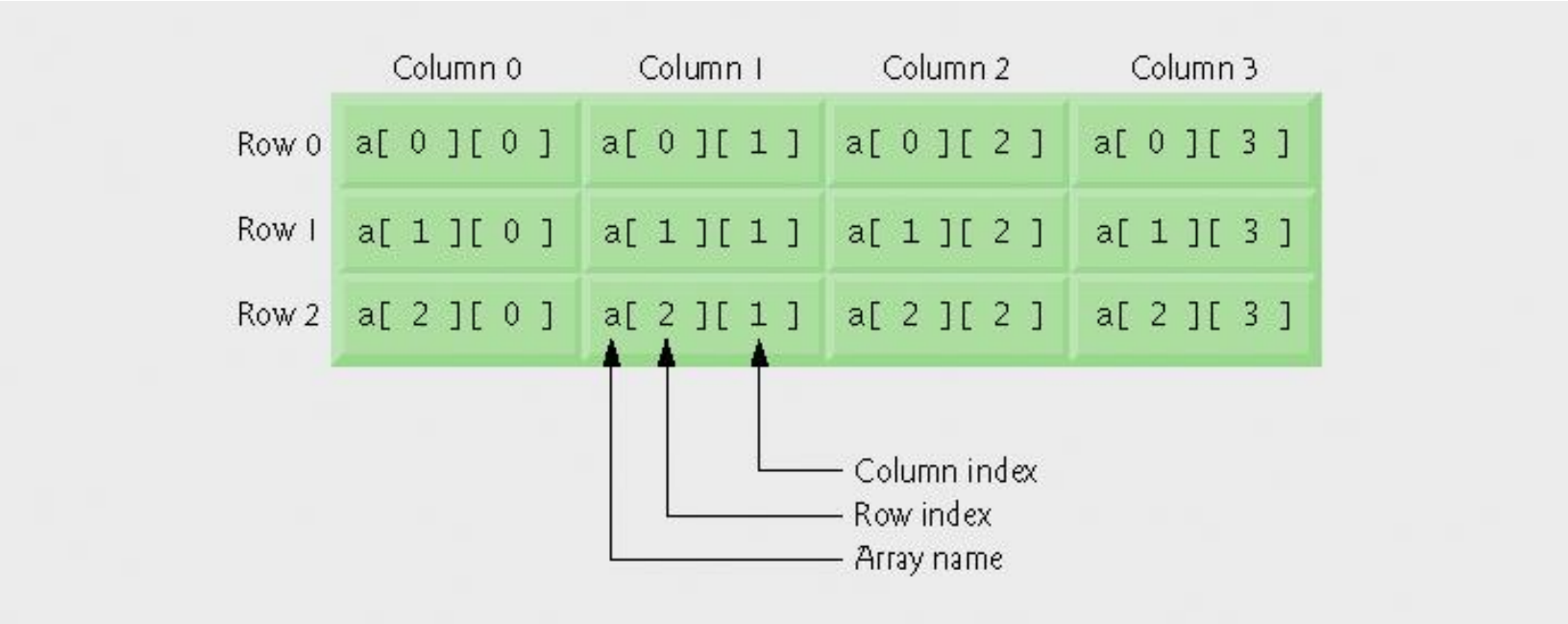


Fig. 4a.16 | Two-dimensional array with three rows and four columns.

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 of 2)

Line 9

Line 10

```
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`

Outline

```
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

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

InitArray.java

(2 of 2)

Line 26

Line 27

```
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

```java
   // display a welcome message to the GradeBook user
   public void displayMessage()
   {
      // getCourseName gets the name of the course
      System.out.printf( "Welcome to the grade book for\n%s!\n\n",
         getCourseName() );
   } // end method displayMessage

   // perform various operations on the data
   public void processGrades()
   {
      // output grades array
      outputGrades();

      // call methods getMinimum and getMaximum
      System.out.printf( "\n%s %d\n%s %d\n\n",
         "Lowest grade in the grade book is", getMinimum(),
         "Highest grade in the grade book is", getMaximum() );

      // output grade distribution chart of all grades on all tests
      outputBarChart();
   } // end method processGrades

   // find minimum grade
   public int getMinimum()
   {
      // assume first element of grades array is smallest
      int lowGrade = grades[ 0 ][ 0 ];

```

```
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

Outline

`GradeBook.java`

(3 of 7)

Lines 58-67

```
78      // loop through rows of grades array
79      for ( int studentGrades[] : grades )
80      {
81         // loop through columns of current row
82         for ( int grade : studentGrad
83         {
84            // if grade greater than h
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`

(4 of 7)

Lines 79-88

Lines 94-104

```
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

(5 of 7)

Lines 115-119

```java

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

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

      // create a column heading for each of the tests
      for ( int test = 0; test < grades[ 0 ].length; test++ )
         System.out.printf( "Test %d  ", test + 1 );

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

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

         for ( int test : grades[ student ] ) // output student's grades
            System.out.printf( "%8d", test );

```

Outline

GradeBook.java

(6 of 7)

```java
         // call method getAverage to calculate student's average grade;
         // pass row of grades as the argument to getAverage
         double average = getAverage( grades[ student ] );
         System.out.printf( "%9.2f\n", average );
      } // end outer for
   } // end method outputGrades
} // end class GradeBook
```

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 GradeB
22          "CS101 Introduction to Java Pro
23       myGradeBook.displayMessage();
24       myGradeBook.processGrades();
25    } // end main
26 } // end class GradeBookTest
```

Declare `gradesArray` as 10-by-3 array

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

.java

(1 of 2)

Lines 10-19

```
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: ***
```

Outline

GradeBookTest

.java

(2 of 2)

Program output

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

```
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 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 `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

```
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
```

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
```

VarargsTest.java

2)

Line 29

Line 31

Line 33

Program output

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 ]`

```
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 co
20          int initialValue = Integer.parseInt( args[
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

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.
```

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

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()	Creates an empty list.
+add(o: E): void	Appends a new element o at the end of this list.
+add(index: int, o: E): void	Adds a new element o at the specified index in this list.
+clear(): void	Removes all the elements from this list.
+contains(o: Object): boolean	Returns true if this list contains the element o.
+get(index: int): E	Returns the element from this list at the specified index.
+indexOf(o: Object): int	Returns the index of the first matching element in this list.
+isEmpty(): boolean	Returns true if this list contains no elements.
+lastIndexOf(o: Object): int	Returns the index of the last matching element in this list.
+remove(o: Object): boolean	Removes the first element o from this list. Returns true if an element is removed.
+size(): int	Returns the number of elements in this list.
+remove(index: int): boolean	Removes the element at the specified index. Returns true if an element is removed.
+set(index: int, o: E): E	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	`String[] a = new String[10]`	`ArrayList<String> list = new ArrayList<>();`
Accessing an element	`a[index]`	`list.get(index);`
Updating an element	`a[index] = "London";`	`list.set(index, "London");`
Returning size	`a.length`	`list.size();`
Adding a new element		`list.add("London");`
Inserting a new element		`list.add(index, "London");`
Removing an element		`list.remove(index);`
Removing an element		`list.remove(Object);`
Removing all elements		`list.clear();`

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 custom `javafx.scene.paint.Color`
- Drawing arcs

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

Colors can be created with the constructor or with one of several utility methods. The following lines of code all create the same blue color:

```
Color c = Color.BLUE;                //use the blue constant
Color c = new Color(0,0,1,1.0); // standard constructor, use
                          0->1.0 values, explicit alpha of 1.0
Color c = Color.color(0,0,1.0); //use 0->1.0 values.
                                   implicit alpha of 1.0
Color c = Color.color(0,0,1.0,1.0); //use 0->1.0 values,
                                     explicit alpha of 1.0
Color c = Color.rgb(0,0,255); //use 0->255 integers,
                                implicit alpha of 1.0
Color c = Color.rgb(0,0,255,1.0); //use 0->255 integers,
                                   explicit alpha of 1.0
```

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

```
public Arc(double centerX, double centerY,
           double radiusX, double radiusY,
           double startAngle, double endAngle)
```

Creates a new instance of Arc.

Parameters:

`centerX` - the X coordinate of the center point of the arc

`centerY` - the Y coordinate of the center point of the arc

`radiusX` - the overall width (horizontal radius) of the full ellipse of which this arc is a partial section

`radiusY` - the overall height (vertical radius) of the full ellipse of which this arc is a partial section

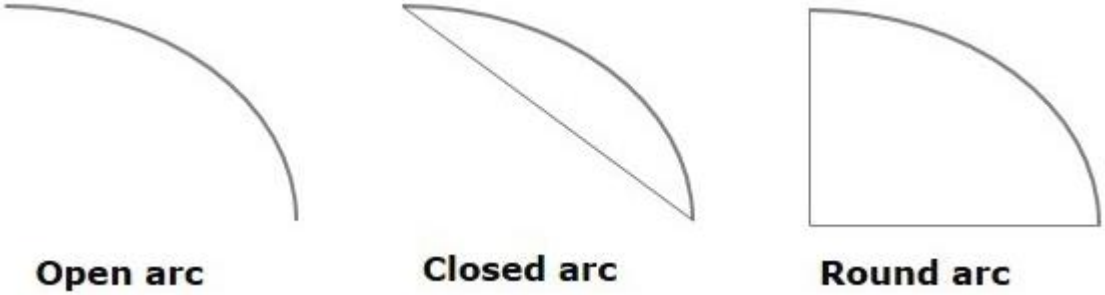
`startAngle` - the starting angle of the arc in degrees

`endAngle` - the angular extent of the arc in degrees

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

```
public Arc(double centerX, double centerY,
           double radiusX, double radiusY,
           double startAngle, double endAngle)
```

You can also set the type of the arc (round, chord or open) by using the `setType()` method.



//Setting the properties of the a

```
arc.setCenterX(300.0);
arc.setCenterY(150.0);
arc.setRadiusX(90.0);
arc.setRadiusY(90.0);
arc.setStartAngle(40.0);
arc.setLength(239.0);
arc.setType(ArcType.ROUND);
```

```
1  import javafx.application.Application;
2  import javafx.scene.Scene;
3  import javafx.scene.layout.Pane;
4  import javafx.scene.paint.Color;
5  import javafx.scene.shape.Arc;
6  import javafx.scene.shape.ArcType;
7  import javafx.stage.Stage;
8
9  public class DrawRainbow extends Application {
10     // colors to use in the rainbow, starting from the innermost
11     // The two white entries result in an empty arc in the center
12     private final Color VIOLET ;
13     private final Color INDIGO ;
14     private Color colors[];
15
16     public DrawRainbow() {
17         VIOLET = Color.rgb(75, 0, 130, 1.0);
18         INDIGO = Color.rgb(128, 0, 128, 1.0);
19         colors = new Color[]{Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
20                              Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED};
21     }
22     public void start(Stage primaryStage) {
23         Pane root = new Pane();
24         Arc arc;
25         Scene scene = new Scene(root, 800, 400);
26         int radius = 40; //  radius of the arc
```

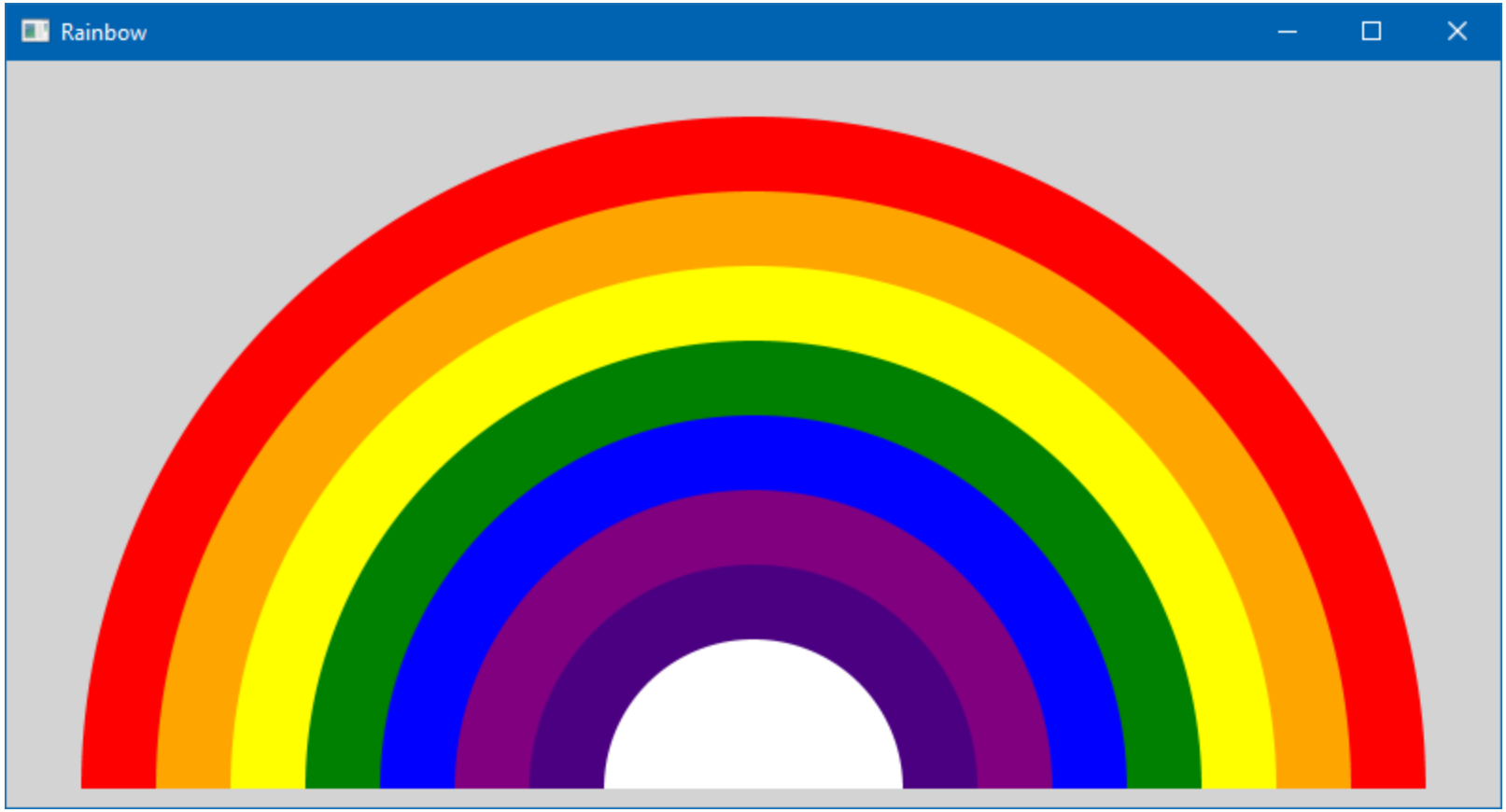
Setup the colors for drawing the rainbow

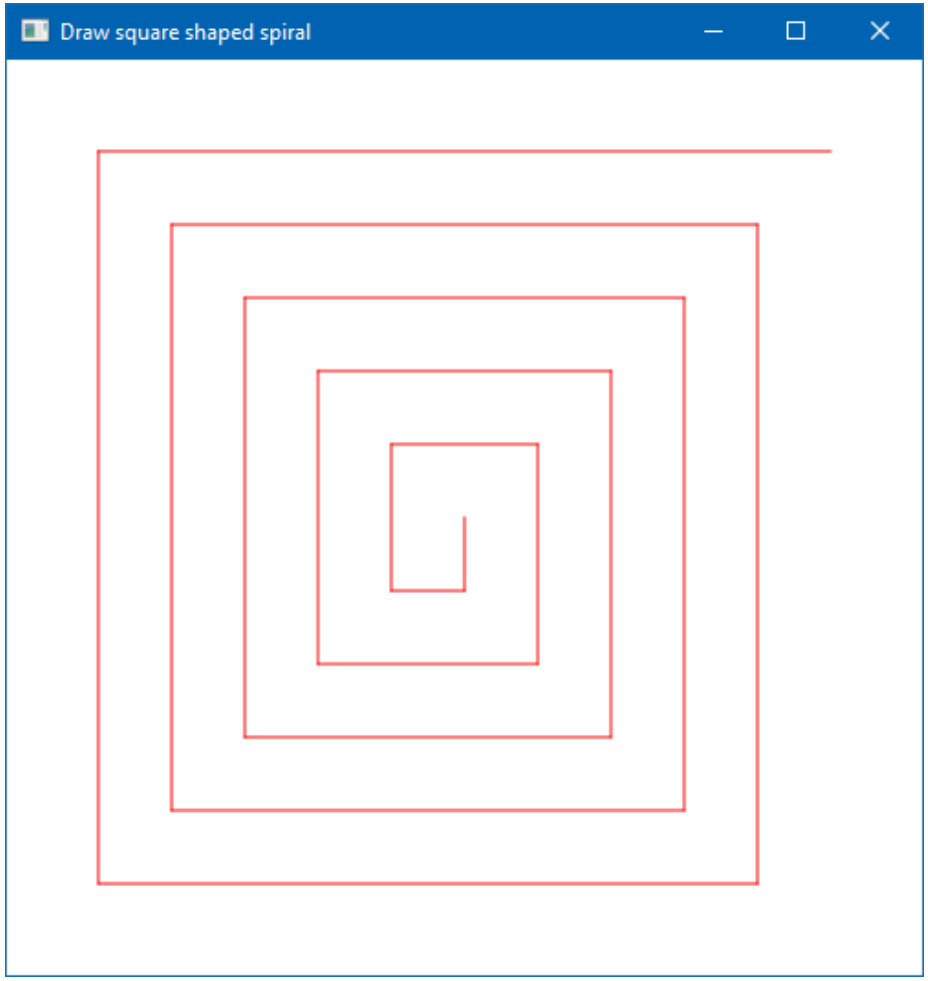
Setup the parent Node, the Shape and the Scene dimensions

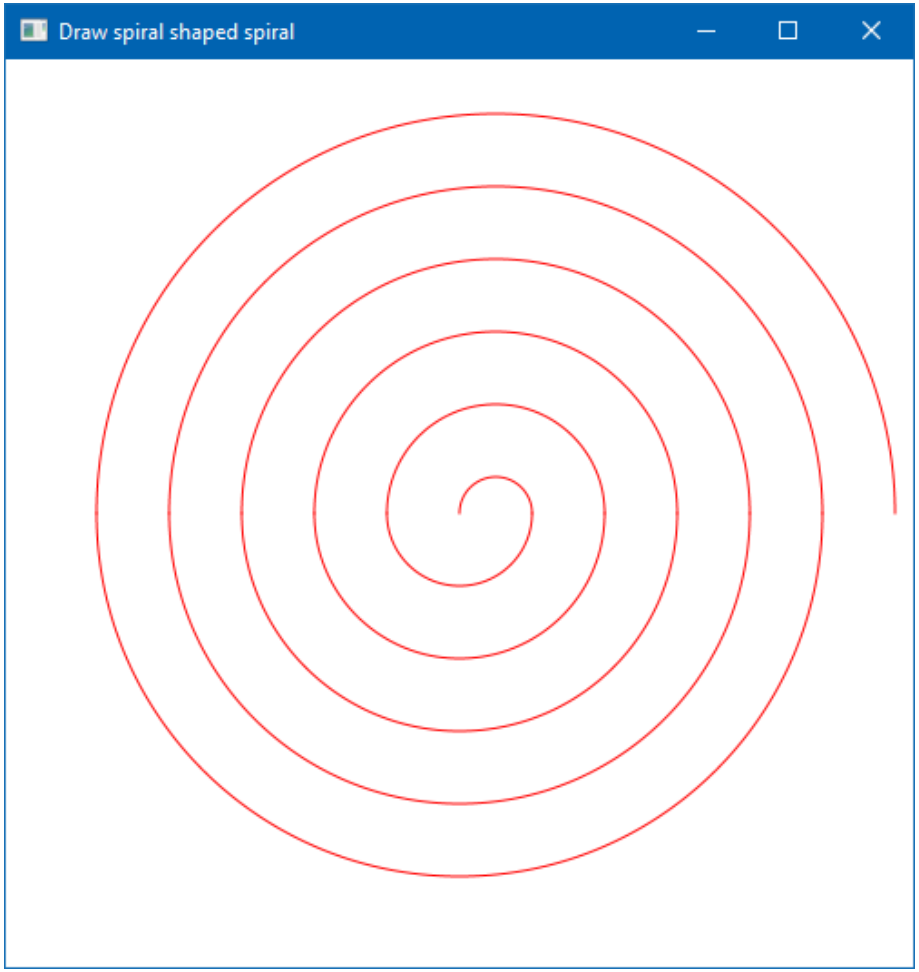
```
27          //using styles with a Pane Node
28          root.setStyle("-fx-background-color: lightgray;");
29          // draw the rainbow near the bottom-center
30          double centerX = scene.getWidth() / 2;
31          double centerY = scene.getHeight() - 10;
32
33          // draws filled arcs starting with the outermost
34          for (int counter = colors.length; counter > 0; counter--) {
35              // fill the arc from 0 to 180 degrees
36              arc = new Arc(centerX ,centerY ,
37                            counter * radius , counter * radius,
38                            0, 180);
39              arc.setFill(colors[counter - 1]);// set the color for the current arc
40              arc.setType(ArcType.OPEN);
41
42              root.getChildren().add(arc);
43          } // end for
44
45          primaryStage.setTitle("Rainbow");
46          primaryStage.setScene(scene);
47          primaryStage.show();
48      }
49
50      public static void main(String[] args) {
51          launch(args);
52      }
53  }
```

Setup the Arc properties

Add the arc objects to the root Node


Rainbow





Drawing a square spiral (left) and circular spiral (right).

4a.15 Основен модел на рекурсия

Основни елементи на рекурсията

- ***Базов(и) (граничен/ни) случай/и***
  - ***Рекурсивен метод, който дава решение само за най- простия случай—the base case***
  - ***Ако рекурсивният метод се извика с базовия случай, методът връща резултат (не се извиква рекурсивно)***
- ***Пример:***

  При обхождане на списък, граничният случай е списък с един елемент

Индиректна рекурсия

- ***Възможно е при изпълнението на рекурсивната стъпка да се извика друг метод, който от своя страна да извика рекурсивния метод и така да се затвори цикъла на рекурсията***

4a.15 Основен модел на рекурсия

Рекурсивна стъпка

- ***При извикване на рекурсивния метод за решаване на случай, различен от базовия, задачата се разделя на две части— част, която методът дефинира как да изпълни и част, която методът не дефинира как да изпълни (наричана рекурсивна стъпка, извикване)***
- ***Рекурсивна стъпка, извикване- изисквания***
  - ***Трябва да решава зададения първоначален проблем, но в по- прост вид или умален размер***
  - ***Методът извиква себе си за решаване на същия проблем, но с по- малка размерност***
  - ***Обикновено, включва return statement***

- ***Пример за рекурсивна стъпка***
  - ***Обхождането на дърво се свежда до обхождане на лявото и дясното под- дървета***

4a.15 Основен модел на рекурсия

N факториел е произведението

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$

като дефинираме

$1! = 1$

$0! = 1.$

Може да се реализира рекурсивно или итеративно

За рекурсивното решение забелязваме, че:

$n! = n \cdot (n-1)!$

Тук

- ***граничният случай е n <=1***
- ***Рекурсивната стъпка е .(n – 1)!***

4a.15 Основен модел на рекурсия

Безкрайна рекурсия– неограничена последователност от рекурсивни извиквания на метода

- ***Програмата прекратява поради изчерпване на отделената памет за изпълнение***
- ***Предизвиква се от пропуснато или неправилно дефинирано гранично условие , при което рекурсивните извиквания не се схождат до базовия (граничен) случай***

4a.15 Основен модел на рекурсия

stack структурирана памет съхранява поредните извиквания на методи и локалните данни, зададени като аргументи на метода

Също както и нерекурсивните методи, адресите на рекурсивните методи се записват отгоре на стек-а с извиквания на методи

Когато рекурсивен метод изпълни return, тяхните записи за действие (activation record) и тези на предхождащите ги рекурсивни извиквания се "изхвърлят" от стек-а

Текущо изпълняваният метод е този метод, чиито запис за действие е на върха на стек-а

```java
// Fig. 15.3: FactorialCalculator.java
// Recursive factorial method.

public class FactorialCalculator
{
   // recursive method factorial
   public long factorial( long number )
   {
      if ( number <= 1 ) // test for base case
         return 1; // base cases: 0! = 1 and 1! = 1
      else // recursion step
         return number * factorial( number - 1 );
   } // end method factorial

   // output factorials for values 0-10
   public void displayFactorials()
   {
      // calculate the factorials of 0 through 10
      for ( int counter = 0; counter <= 10; counter++ )
         System.out.printf( "%d! = %d\n", counter, factorial( counter ) );
   } // end method displayFactorials
} // end class FactorialCalculator
```

Граничният случай връща 1

Рекурсивната стъпка разделя проблема на две части: **едната** методът дефинира как да изпълни, **другата е аналог на зададения проблем, но с по- малка размерност**

Рекурсивно извикване

Първото извикване на рекурсивния метод

Тази част от проблема за пресмятане на **n!** е дефинирана в рекурсивния метод

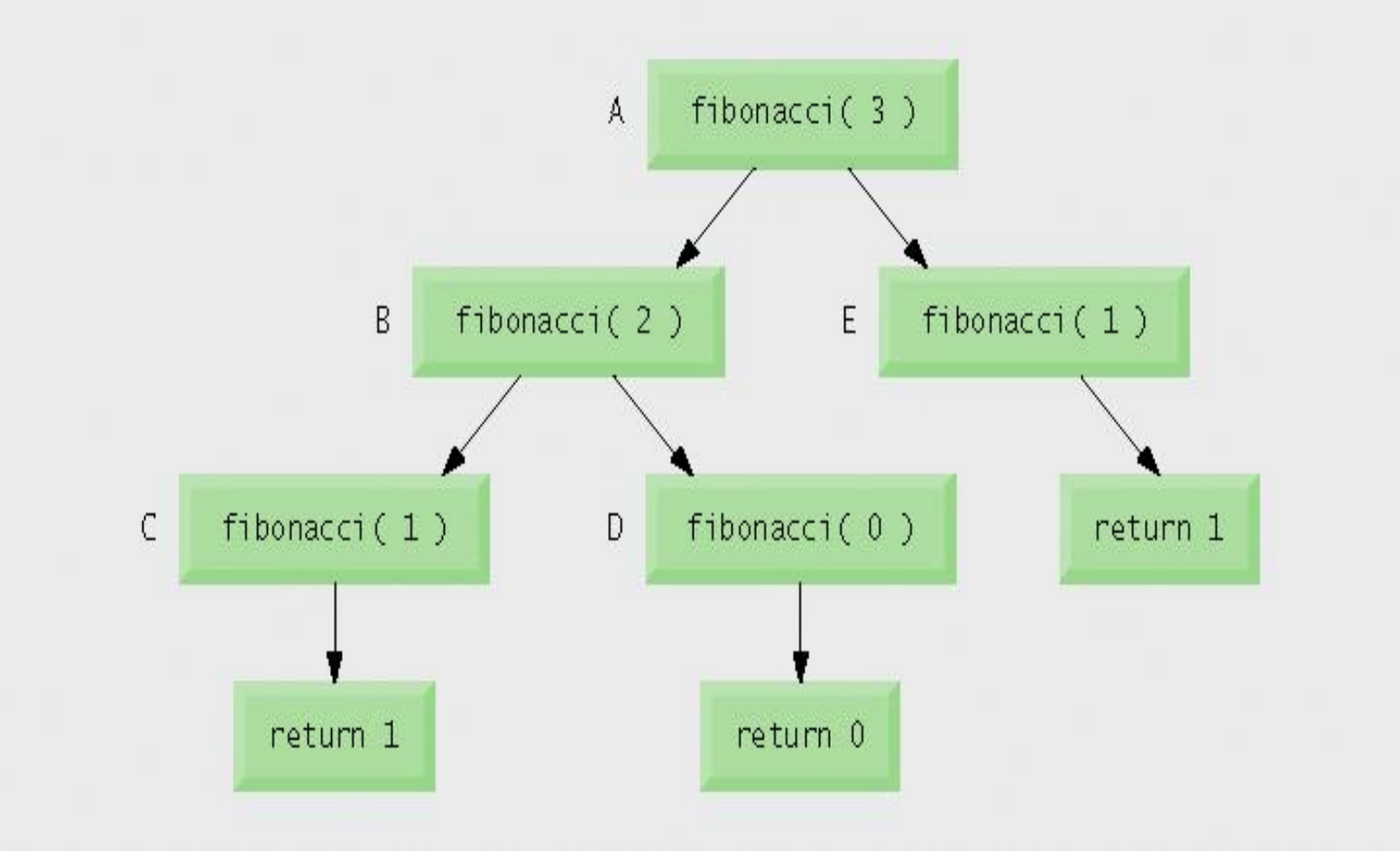
Обичайна грешка при програмиране

Пропускането или неправилното дефиниране на граничния случай води до безкрайна рекурсия докато се изчерпи заделената памет за изпълнение на програмата. Това е аналогична грешка на безкрайния цикъл при итеративните методи.

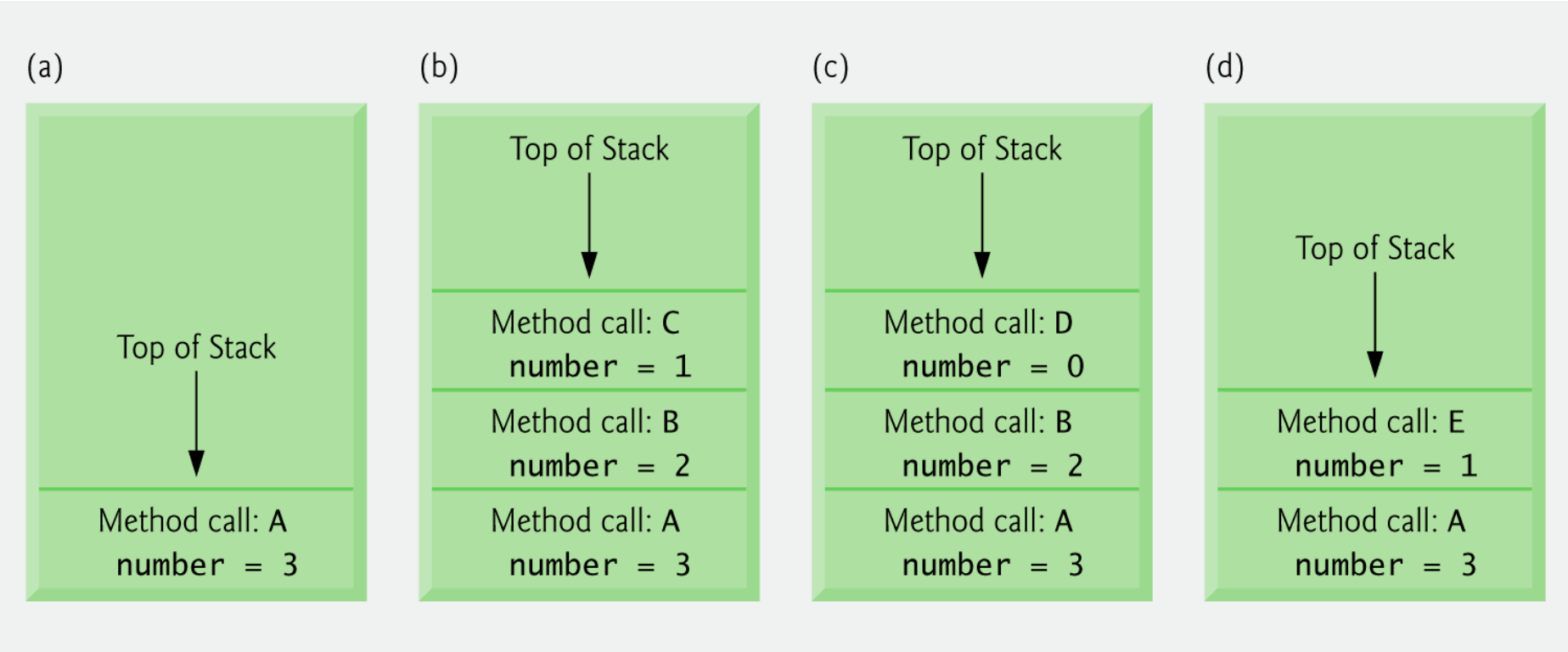
```java
// Fig. 15.4: FactorialTest.java
// Testing the recursive factorial method.

public class FactorialTest
{
   // calculate factorials of 0-10
   public static void main( String args[] )
   {
      FactorialCalculator factorialCalculator = new FactorialCalculator();
      factorialCalculator.displayFactorials();
   } // end main
} // end class FactorialTest
```

Пресмятане и извеждане на факториелите

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```



Извиквания на методи за пресмятане на първите 3 члена от редицата на Фибоначи



Извикванията на методи в изпълнимия стек на програмата.

```
/* Problem Binary Search
   Input: an array arr sorted in increasing order, an integer n
   Output: index of n in arr, -1 if absent
 */
public int binarySearch(int[] arr, int n, int low, int high){
    if (high <= low) {
        if (arr[high] == n)
            return low;
        else
            return -1;
    } else {
        int middle =( high + low + 1)/2;
        if (arr[middle] > n) {
            return binarySearch(arr, n, low, middle -1 );
        } else {
            if (arr[middle] < n)
                return binarySearch(arr, n, middle +1,  high);
            else
                return middle;
        }
    }
}
```

Три гранични условия

Едно рекурсивно извикване, но при различни условия

Намерено е съвпадение в средата на масива

4a.15 Основен модел на рекурсия

Merge sort на масиви

- ***По- ефективен метод за сортиране, но и по сложен***
- ***Разделя дадения масив на два по- малки масива с приблизително равен брой елементи, сортира всеки от тези по- малки масива, накрая смесва двата сортирани подмасива в един масив***
- ***Рекурсивен модел на Merge sort***
  - ***Граничният случай е едно еле ментен масив, който е сортиран***
  - ***Рекурсивната стъпка включва (1)разделяне на масива на две части, (2)сортиране на всяка част и (3) смесването на двете части***

```java
// Figure 16.10: MergeSort.java
// Class that creates an array filled with random integers.
// Provides a method to sort the array with merge sort.
import java.util.Random;

public class MergeSort
{
   private int[] data; // array of values
   private static Random generator = new Random();

   // create array of given size and fill with random integers
   public MergeSort( int size )
   {
      data = new int[ size ]; // create space for array

      // fill array with random ints in range 10-99
      for ( int i = 0; i < size; i++ )
         data[ i ] = 10 + generator.nextInt( 90 );
   } // end MergeSort constructor

   // calls recursive split method to begin merge sort
   public void sort()
   {
      sortArray( 0, data.length - 1 ); // split entire array
   } // end method sort

```

Извиква рекурсивният метод

```
27    // splits array, sorts subarrays and merges subarrays into sorted array
28    private void sortArray( int low, int high )
29    {
30       // test base case; size of array equals 1
31       if ( ( high - low ) >= 1 ) // if not base case
32       {
33          int middle1 = ( low + high ) / 2; // calculate middle of array
34          int middle2 = middle1 + 1; // calculate next element over
35
36          // output split step
37          System.out.println( "split:   " + subarray( low, high ) );
38          System.out.println( "         " + subarray( low, middle1 );
39          System.out.println( "         " + subarray( middle2, high ) );
40          System.out.println();
41
42          // split array in half; sort each half (recursive calls)
43          sortArray( low, middle1 ); // first half of array
44          sortArray( middle2, high ); // second half of array
45
46          // merge two sorted arrays after split calls return
47          merge ( low, middle1, middle2, high );
48       } // end if
49    } // end method split
50
```

Тества за граничния случай

Пресмята средата на масива

Пресмята индекса на елемента отдясно на средния

Рекурсивно сортиране на първата половинка масив

...сортира и втората половинка

Смесва сортирано двете половинки

```
51    // merge two sorted subarrays into one sorted subarray
52    private void merge( int left, int middle1, int middle2, int right )
53    {
54       int leftIndex = left; // index into left subarray
55       int rightIndex = middle2; // index into right subarray
56       int combinedIndex = left; // index into temporary working array
57       int[] combined = new int[ data.length ]; // working array
58
59       // output two subarrays before merging
60       System.out.println( "merge:    " + subarray( left, middle1 ) );
61       System.out.println( "          " + subarray( middle2, right ) );
62
63       // merge arrays until reaching end of either
64       while ( leftIndex <= middle1 && rightIndex <= right )
65       {
66          // place smaller of two current elements into result
67          // and move to next space in arrays
68          if ( data[ leftIndex ] <= data[ rightIndex ] )
69             combined[ combinedIndex++ ] = data[ leftIndex++ ];
70          else
71             combined[ combinedIndex++ ] = data[ rightIndex++ ];
72       } // end while
73
```

Индекс за левия масив

Индекс за дясния масив

Индекс за общия масив

Общият масив

Цикъл докато не се изчерпи един от двата масива- ляв и десен

Определя по- малкия от двата

Записва по- малкия елемент в общия масив

```
74      // if left array is empty
75      if ( leftIndex == middle2 )
76         // copy in rest of right array
77         while ( rightIndex <= right )
78            combined[ combinedIndex++ ] = data[ rightIndex++ ];
79      else // right array is empty
80         // copy in rest of left array
81         while ( leftIndex <= middle1 )
82            combined[ combinedIndex++ ] = data[ leftIndex++ ];
83
84      // copy values back into original array
85      for ( int i = left; i <= right; i++ )
86         data[ i ] = combined[ i ];
87
88      // output merged array
89      System.out.println( "          " + subarray( left, right ) );
90      System.out.println();
91   } // end method merge
92
```

Ако левият масив е празен

Допълваме общия с останалите елементи от дясния

Ако дясният масив е празен

Допълваме общия с останалите елементи от лявия

Копираме елементите в зададения масив

```java
   // method to output certain values in array
   public String subarray( int low, int high )
   {
      StringBuffer temporary = new StringBuffer();

      // output spaces for alignment
      for ( int i = 0; i < low; i++ )
         temporary.append( "   " );

      // output elements left in array
      for ( int i = low; i <= high; i++ )
         temporary.append( " " + data[ i ] );

      return temporary.toString();
   } // end method subarray

   // method to output values in array
   public String toString()
   {
      return subarray( 0, data.length - 1 );
   } // end method toString
} // end class MergeSort
```

```java
// Figure 16.11: MergeSortTest.java
// Test the merge sort class.

public class MergeSortTest
{
   public static void main( String[] args )
   {
      // create object to perform merge sort
      MergeSort sortArray = new MergeSort( 10 );

      // print unsorted array
      System.out.println( "Unsorted:" + sortArray + "\n" );

      sortArray.sort(); // sort array

      // print sorted array
      System.out.println( "Sorted:  " + sortArray );
   } // end main
} // end class MergeSortTest
```

```
Unsorted: 75 56 85 90 49 26 12 48 40 47

split:    75 56 85 90 49 26 12 48 40 47
          75 56 85 90 49
                         26 12 48 40 47

split:    75 56 85 90 49
          75 56 85
                   90 49

split:    75 56 85
          75 56
                85

split:    75 56
          75
             56
```

```
merge:    75
             56
          56 75

merge:    56 75
                85
          56 75 85

split:             90 49
                   90
                      49

merge:             90
                      49
                   49 90

merge:    56 75 85
                   49 90
          49 56 75 85 90

split:                  26 12 48 40 47
                        26 12 48
                                 40 47

split:                  26 12 48
                        26 12
                              48

split:                  26 12
                        26
                           12
```

```
merge:                    26
                             12
                          12 26

merge:                    12 26
                                48
                          12 26 48

split:                            40 47
                                  40
                                     47

merge:                            40
                                     47
                                  40 47

merge:                    12 26 48
                                  40 47
                          12 26 40 47 48

merge:    49 56 75 85 90

          12 26 40 47 48 49 56 75 85 90

Sorted:   12 26 40 47 48 49 56 75 85 90
```

4a.15 Основен модел на рекурсия

Merge sort

- ***Много по- ефективен от сиртиране с избор и вмъкване***
- ***Последният merge изисква n – 1 сравнения за смесване на сортираните части от масива***
- ***На всяко по- долно ниво има два пъти повече извиквания на merge, и всяко извикване работи с половината от елементите на предишното нива- O(n) общо сравнения***
- ***Общо нивата на разделяне на половинки е O(log n)***
- ***Общата ефективност е O(n log n)***