Arrays
Introduction

- Array
  - Group of contiguous memory locations
    - Each memory location has same name
    - Each memory location has same type
  - Remain same size once created
    - *Static* entries

Name of array (Note that all elements of this array have the same name, **c**)

- Examine array **c**
  - **c** is the array *name*
  - **c.length** accesses array **c**'s *length*
  - **c** has 12 *elements* (**c[0]**, … **c[11]**)
    - The *value* of **c[0]** is **-45**
    - The brackets (**[]**) are in highest precedence in Java

Position number (index of subscript) of the element within array **c**

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

Fig. 7.1 A 12-element array.

Arrays

- Subscript
  - Also called an *index*
  - Position number in square brackets
  - Must be integer or integer expression

```
a = 5;
b = 6;
c[ a + b ] += 2;
```

3

Operators	Associativity	Type
`() [] .`	left to right	highest
`++ --`	right to left	unary postfix
`++ -- + - ! (type)`	right to left	unary
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`&`	left to right	boolean logical AND
`^`	left to right	boolean logical exclusive OR
`|`	left to right	boolean logical inclusive OR
`&&`	left to right	logical AND
`||`	left to right	logical OR
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 7.2 Precedence and associativity of the operators discussed so far.

4

Declaring and Allocating Arrays

- Declaring and Allocating arrays
  - Arrays are objects that occupy memory
  - Allocated dynamically with operator **new**

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

    - Equivalent to

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

    - We can allocate arrays of objects too

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

- Initializing arrays

1. int [][] a; 2. int []a[]; 3. int a[][];
   4. int [ ]a[ ] are ok.

5

```
2  // Creating an array.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int array[];            // declare reference to an array
13
14       array = new int[ 10 ];  // dynamica
15
16       String output = "Subscript\tValue\n";
17
18       // append each array element's value to String output
19       for ( int counter = 0; counter < array.length; counter++ )
20          output += counter + "\t" + array[ counter ] + "\n";
21
22       JTextArea outputArea = new JTextArea();
23       outputArea.setText( output );
24
25       JOptionPane.showMessageDialog( null, outputArea,
26          "Initializing an Array of int Values",
27          JOptionPane.INFORMATION_MESSAGE );
28
29       System.exit( 0 );
30    }
31 }
```

array.length returns length of **array**

Outline

InitArray.java

Line 12
Declare **array** as an array of **int**s

Line 14
Allocate **10** **int**s for **array**; each **int** is initialized to **0** by default

Line 19
array.length returns length of **array**

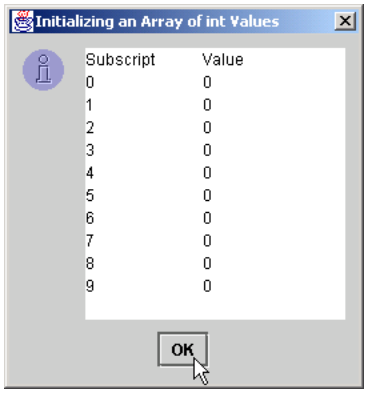
Line 20
array[counter] returns **int** associated with index in **array**

6

Outline

`InitArray.java`

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



7

Using an Initializer List to Initialize Elements of an Array

- Initialize array elements
  - 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
      - Subscripts of **`0`**, **`1`**, **`2`**, **`3`**, **`4`**
  - Do not need operator **`new`**
- int n[]=new int[] {1,2,3,4};
- int n[]=new [5];

8

```
2  // Initializing an array with a declaration.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       // initializer list specifies number of elements and
13       // value for each element
14       int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
15
16       String output = "Subscript\tValue\n";
17
18       // append each array element's value to String output
19       for ( int counter = 0; counter < array.length; counter++ )
20          output += counter + "\t" + array[ counter ] + "\n";
21
22       JTextArea outputArea = new JTextArea();
23       outputArea.setText( output );
24
25       JOptionPane.showMessageDialog( null, outputArea,
26          "Initializing an Array with a Declaration",
27          JOptionPane.INFORMATION_MESSAGE );
28
29       System.exit( 0 );
30    }
31 }
```

Outline

InitArray.java

Line 14
Declare **array** as an array of **int**s

Line 14
Compiler uses initializer list to allocate array

9

```
2  // Initialize array with the even integers from 2 to 20.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class InitArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       final int ARRAY_SIZE = 10;
13       int array[];                    // reference to int array
14
15       array = new int[ ARRAY_SIZE ];  // allocate array
16
17       // calculate value for each array element
18       for ( int counter = 0; counter < array.length; counter++ )
19          array[ counter ] = 2 + 2 * counter;
20
21       String output = "Subscript\tValue\n";
22
23       for ( int counter = 0; counter < array.length; counter++ )
24          output += counter + "\t" + array[ counter ] + "\n";
25
26       JTextArea outputArea = new JTextArea();
27       outputArea.setText( output );
28
29       JOptionPane.showMessageDialog( null, outputArea,
30          "Initializing to Even Numbers from 2 to 20",
31          JOptionPane.INFORMATION_MESSAGE );
32
33       System.exit( 0 );
34    }
35 }
```

Outline

InitArray.java

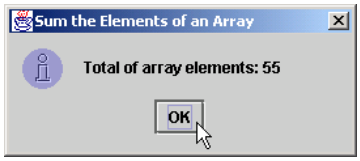
Line 13
Declare **array** as an array of **int**s

Line 15
Allocate **10** **int**s for **array**

Line 19
Use **array** subscript to assign array value

10

```
2  // Total the values of the elements of an array.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class SumArray {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
13       int total = 0;
14
15       // add each element's value to total
16       for ( int counter = 0; counter < array.length; counter++ )
17          total += array[ counter ];
18
19       JOptionPane.showMessageDialog( null,
20          "Total of array elements: " + total,
21          "Sum the Elements of an Array",
22          JOptionPane.INFORMATION_MESSAGE );
23
24       System.exit( 0 );
25    }
26 }
```



Outline

SumArray.java

Line 12
Declare **array** with initializer list

Line 17
Sum all **array** values

11

Using Histograms to Display Array Data Graphically

- Present array values graphically
  - Histogram
    - Plot each numeric value as bar of asterisks (*****)

12

```
2  // Histogram printing program.
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class Histogram {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int array[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
13
14       String output = "Element\tValue\tHistogram";
15
16       // for each array element, output a bar in histogram
17       for ( int counter = 0; counter < array.length; counter++ ) {
18          output +=
19             "\n" + counter + "\t" + array[ counter ] + "\t";
20
21          // print bar of asterisks
22          for ( int stars = 0; stars < array[ counter ]; stars++ )
23             output += "*";
24       }
25
26       JTextArea outputArea = new JTextArea();
27       outputArea.setText( output );
28
29       JOptionPane.showMessageDialog( null, outputArea,
30          "Histogram Printing Program",
31          JOptionPane.INFORMATION_MESSAGE );
32
33       System.exit( 0 );
34    }
35 }
```

Outline

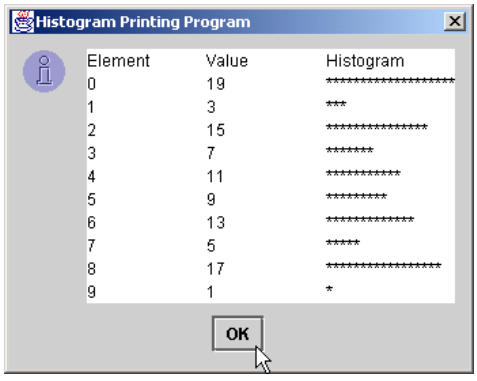
Histogram.java

Line 12
Declare **array** with initializer list

Line 23
For each **array** element, print associated number of asterisks

Outline

Histogram.java



Using the Elements of an Array as Counters

- Use series of counters to summarize data
  - Array can store these counters

15

```
2  // Roll a six-sided die 6000 times
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class RollDie {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int face, frequency[] = new int[ 7 ];
13
14       // roll die 6000 times
15       for ( int roll = 1; roll <= 6000; roll++ ) {
16          face = 1 + ( int ) ( Math.random() * 6 );
17
18          // use face value as subscript for frequency array
19          ++frequency[ face ];
20       }
21
22       String output = "Face\tFrequency";
23
24       // append frequencies to String output
25       for ( face = 1; face < frequency.length; face++ )
26          output += "\n" + face + "\t" + frequency[ face ];
27
28       JTextArea outputArea = new JTextArea();
29       outputArea.setText( output );
30
31       JOptionPane.showMessageDialog( null, outputArea,
32          "Rolling a Die 6000 Times",
33          JOptionPane.INFORMATION_MESSAGE );
34
```

Outline

RollDie.java

Line 12
Declare **frequency** as array of **7 int**s

Lines 15-16
Generate **6000** random integers in range **1-6**

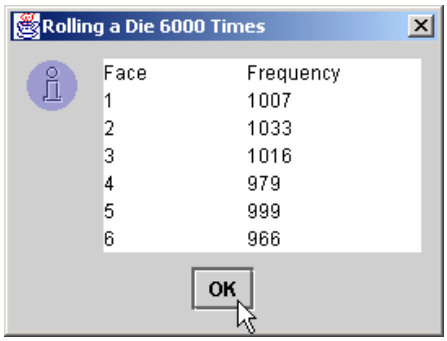
Line 19
Increment **frequency** values at index associated with random number

16

```
35         System.exit( 0 );
36      }
37   }
```

Outline

RollDie.java



17

Using Arrays to Analyze Survey Results

- Problem statement
  - 40 students rate the quality of food
    - **1-10** Rating scale: **1** mean awful, **10** means excellent
  - Place 40 responses in array of integers
  - Summarize results

18

```
2  // Student poll program
3
4  // Java extension packages
5  import javax.swing.*;
6
7  public class StudentPoll {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
13                           1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
14                           6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
15                           5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
16       int frequency[] = new int[ 11 ];
17
18       // for each answer, select value of an element of
19       // responses array and use that value as subscript in
20       // frequency array to determine element to increment
21       for ( int answer = 0; answer < responses.length; answer++ )
22          ++frequency[ responses[ answer ] ];
23
24       String output = "Rating\tFrequency\n";
25
26       // append frequencies to String output
27       for ( int rating = 1; rating < frequency.length; rating++ )
28          output += rating + "\t" + frequency[ rating ] + "\n";
29
30       JTextArea outputArea = new JTextArea();
31       outputArea.setText( output );
32
33       JOptionPane.showMessageDialog( null, outputArea,
34          "Student Poll Program",
35          JOptionPane.INFORMATION_MESSAGE );
```

Outline

StudentPoll.java

Lines 12-15
Declare **responses** as array to store **40** responses

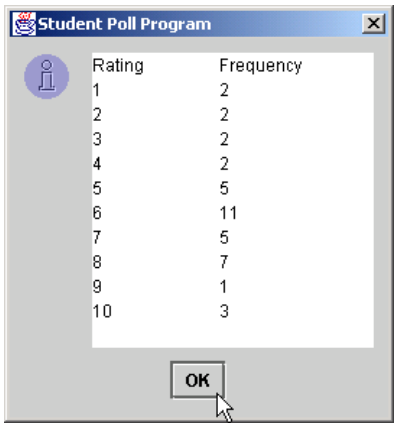
Line 16
Declare **frequency** as array of **11** **int** and ignore the first element

Lines 21-22
For each response, increment **frequency** values at index associated with that response


```
36
37       System.exit( 0 );
38    }
39 }
```

Outline

StudentPoll.java



Using Arrays to Analyze Survey Results (cont.)

- Some additional points
  - When looping through an array
    - Subscript should never go below 0
    - Subscript should be less than total number of array elements
  - When invalid array reference occurs
    - Java generates **`ArrayIndexOutOfBoundsException`**
      - Chapter 14 discusses exception handling

21

References and Reference Parameters

- Two ways to pass arguments to methods
  - **Pass-by-value**
    - Copy of argument's value is passed to called method
    - In Java, **every primitive is pass-by-value**
  - **Pass-by-reference**
    - Caller gives called method direct access to caller's data
    - Called method can manipulate this data
    - Improved performance over pass-by-value
    - In Java, **every object is pass-by-reference**
      - In Java, **arrays are objects**
        - Therefore, arrays are passed to methods by reference

22

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`**


```
2  // Passing arrays and individual array elements to methods
3
4  // Java core packages
5  import java.awt.Container;
6
7  // Java extension packages
8  import javax.swing.*;
9
10 public class PassArray extends JApplet {
11
12    // initialize applet
13    public void init()
14    {
15       JTextArea outputArea = new JTextArea();
16       Container container = getContentPane();
17       container.add( outputArea );
18
19       int array[] = { 1, 2, 3, 4, 5 };
20
21       String output =
22          "Effects of passing entire array by reference:\n" +
23          "The values of the original array are:\n";
24
25       // append original array elements to String output
26       for ( int counter = 0; counter < array.length; counter++ )
27          output += "   " + array[ counter ];
28
29       modifyArray( array );  // array passed by reference
30
31       output += "\n\nThe values of the modified array are:\n";
32
33       // append modified array elements to String output
34       for ( int counter = 0; counter < array.length; counter++ )
35          output += "   " + array[ counter ];
```

Outline

`PassArray.java`

Line 19
Declare **`5`**-**`int array`** with initializer list

Line 29
Pass **`array`** by reference to method **`modifyArray`**

```
36
37       output += "\n\nEffects of passing array " +
38                 "element by value:\n" +
39                 "a[3] before modifyElement: " + array[ 3 ];
40
41       // attempt to modify array[ 3 ]
42       modifyElement( array[ 3 ] );
43
44       output += "\na[3] after modifyElement: " + array[ 3 ];
45       outputArea.setText( output );
46
47    }  // end method init
48
49    // multiply each element of an array by 2
50    public void modifyArray( int array2[] )
51    {
52       for ( int counter = 0; counter < array2.length; counter++ )
53          array2[ counter ] *= 2;
54    }
55
56    // multiply argument by 2
57    public void modifyElement( int element )
58    {
59       element *= 2;
60    }
61
62 }  // end class PassArray
```

Outline

`PassArray.java`

Line 42
Pass **`array[3]`** by value to method **`modifyElement`**

Lines 50-54
Method **`modifyArray`** manipulates the array directly

Lines 57-60
Method **`modifyElement`** manipulates a primitive's copy

Lines 59
The original primitive is left unmodified

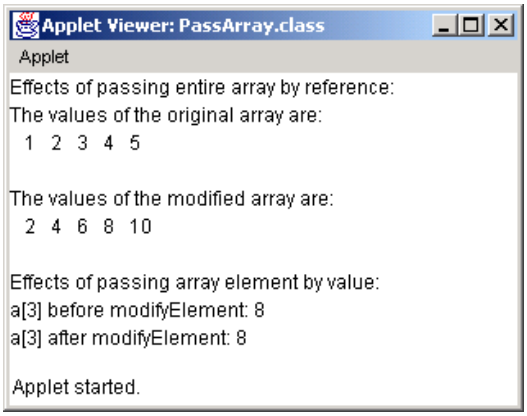
25

Outline

`PassArray.java`

The object passed-by-reference is modified

The primitive passed-by-value is unmodified



26

Sorting Arrays

- Sorting data
  - Attracted intense research in computer-science field
  - Bubble sort
    - Smaller values “bubble” their way to top of array
    - Larger values “sink” to bottom of array
    - Use nested loops to make several passes through array
      - Each pass compares successive pairs of elements
        - Pairs are left along if increasing order (or equal)
        - Pairs are swapped if decreasing order

27

```
2  // Sort an array's values into ascending order.
3
4  // Java core packages
5  import java.awt.*;
6
7  // Java extension packages
8  import javax.swing.*;
9
10 public class BubbleSort extends JApplet {
11
12    // initialize applet
13    public void init()
14    {
15       JTextArea outputArea = new JTextArea();
16       Container container = getContentPane();
17       container.add( outputArea );
18
19       int array[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
20
21       String output = "Data items in original order\n";
22
23       // append original array values to String output
24       for ( int counter = 0; counter < array.length; counter++ )
25          output += "   " + array[ counter ];
26
27       bubbleSort( array );  // sort array
28
29       output += "\n\nData items in ascending order\n";
30
31       // append sorted\ array values to String output
32       for ( int counter = 0; counter < array.length; counter++ )
33          output += "   " + array[ counter ];
34
```

Outline

BubbleSort.java

Line 19
Declare **10**-**int** **array** with initializer list

Line 27
Pass **array** by reference to method **bubbleSort** to sort **array**

28

```
35          outputArea.setText( output );
36       }
37
38       // sort elements of array with bubble sort
39       public void bubbleSort( int array2[] )
40       {
41          // loop to control number of passes
42          for ( int pass = 1; pass < array2.length; pass++ ) {
43
44             // loop to control number of comparisons
45             for ( int element = 0;
46                   element < array2.length - 1;
47                   element++ ) {
48
49                // compare side-by-side elements and swap them if
50                // first element is greater than second element
51                if ( array2[ element ] > array2[ element + 1 ] )
52                   swap( array2, element, element + 1 );
53
54             }  // end loop to control comparisons
55
56          }  // end loop to control passes
57
58       }  // end method bubbleSort
59
60       // swap two elements of an array
61       public void swap( int array3[], int first, int second )
62       {
63          int hold;  // temporary holding area for swap
64
65          hold = array3[ first ];
66          array3[ first ] = array3[ second ];
67          array3[ second ] = hold;
68       }
```

Outline

BubbleSort.java

Line 39
Method **bubbleSort** receives **array** reference as parameter

Lines 42-47
Use loop and nested loop to make passes through **array**

Lines 51-52
If pairs are in decreasing order, invoke method **swap** to swap pairs

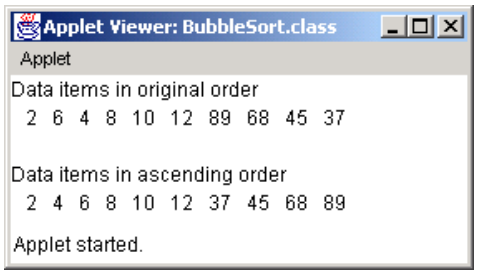
Lines 61-68
Method **swap** swaps two values in **array** reference

29

```
69
70 }  // end class BubbleSort
```

Outline

BubbleSort.java



30

Searching Arrays: Linear Search and Binary Search

- Searching
  - Finding elements in large amounts of data
    - Determine whether array contains value matching *key value*
  - Linear searching
  - Binary searching

31

Searching an Array with Linear Search

- Linear search
  - Compare each array element with *search key*
    - If search key found, return element subscript
    - If search key not found, return **`-1`** (invalid subscript)
  - Works best for small or unsorted arrays
  - Inefficient for larger arrays

32

```
2  // Linear search of an array
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7
8  // Java extension packages
9  import javax.swing.*;
10
11 public class LinearSearch extends JApplet
12    implements ActionListener {
13
14    JLabel enterLabel, resultLabel;
15    JTextField enterField, resultField;
16    int array[];
17
18    // set up applet's GUI
19    public void init()
20    {
21       // get content pane and set its layout to FlowLayout
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24
25       // set up JLabel and JTextField for user input
26       enterLabel = new JLabel( "Enter integer search key" );
27       container.add( enterLabel );
28
29       enterField = new JTextField( 10 );
30       container.add( enterField );
31
32       // register this applet as enterField's action listener
33       enterField.addActionListener( this );
34
```

Outline

LinearSearch.java

Line 16
Declare **array** of **int**s


```
35       // set up JLabel and JTextField for displaying results
36       resultLabel = new JLabel( "Result" );
37       container.add( resultLabel );
38
39       resultField = new JTextField( 20 );
40       resultField.setEditable( false );
41       container.add( resultField );
42
43       // create array and populate with even integers 0 to 198
44       array = new int[ 100 ];
45
46       for ( int counter = 0; counter < array.length; counter++ )
47          array[ counter ] = 2 * counter;
48
49    }  // end method init
50
51    // Search array for specified key value
52    public int linearSearch( int array2[], int key )
53    {
54       // loop through array elements
55       for ( int counter = 0; counter < array2.length; counter++ )
56
57          // if array element equals key value, return location
58          if ( array2[ counter ] == key )
59             return counter;
60
61       return -1;  // key not found
62    }
63
64    // obtain user input and call method linearSearch
65    public void actionPerformed( ActionEvent actionEvent )
66    {
67       // input also can be obtained with enterField.getText()
68       String searchKey = actionEvent.getActionCommand();
69
```

Outline

LinearSearch.java

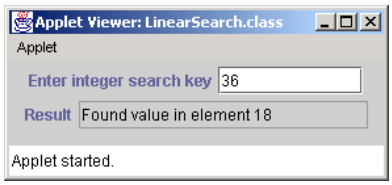
Lines 44-47
Allocate **100** **int**s for **array** and populate **array** with even **int**s

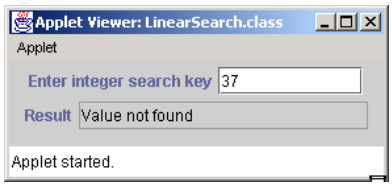
Line 55
Loop through **array**

Lines 58-59
If **array** element at subscript matches *search key*, **return** subscript

Line 65
Invoked when user presses Enter

```
70       // Array a is passed to linearSearch even though it
71       // is an instance variable. Normally an array will
72       // be passed to a method for searching.
73       int element =
74          linearSearch( array, Integer.parseInt( searchKey ) );
75
76       // display search result
77       if ( element != -1 )
78          resultField.setText( "Found value in element " +
79             element );
80       else
81          resultField.setText( "Value not found" );
82    }
83
84 }  // end class LinearSearch
```





Outline

LinearSearch.java

Lines 73-74
Invoke method **linearSearch**, using **array** and search key as arguments

35

Searching a Sorted Array with Binary Search

- Binary search
  - Efficient for large, sorted arrays
  - Eliminates half of the elements in search through each pass
    - Compare middle array element to search key
      - If element equals key
        - Return array subscript
      - If element is less than key
        - Repeat search on first half of array
      - If element is greater then key
        - Repeat search on second half of array
      - Continue search until
        - element equals search key (success)
        - Search contains one element not equal to key (failure)

36

```
2  // Binary search of an array
3
4  // Java core packages
5  import java.awt.*;
6  import java.awt.event.*;
7  import java.text.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class BinarySearch extends JApplet
13    implements ActionListener {
14
15    JLabel enterLabel, resultLabel;
16    JTextField enterField, resultField;
17    JTextArea output;
18
19    int array[];
20    String display = "";
21
22    // set up applet's GUI
23    public void init()
24    {
25       // get content pane and set its layout to FlowLayout
26       Container container = getContentPane();
27       container.setLayout( new FlowLayout() );
28
29       // set up JLabel and JTextField for user input
30       enterLabel = new JLabel( "Enter integer search key" );
31       container.add( enterLabel );
32
33       enterField = new JTextField( 10 );
34       container.add( enterField );
35
```

Outline

BinarySearch.java

Line 19
Declare **array** of **int**s


```
36       // register this applet as enterField's action listener
37       enterField.addActionListener( this );
38
39       // set up JLabel and JTextField for displaying results
40       resultLabel = new JLabel( "Result" );
41       container.add( resultLabel );
42
43       resultField = new JTextField( 20 );
44       resultField.setEditable( false );
45       container.add( resultField );
46
47       // set up JTextArea for displaying comparison data
48       output = new JTextArea( 6, 60 );
49       output.setFont( new Font( "Monospaced", Font.PLAIN, 12 ) );
50       container.add( output );
51
52       // create array and fill with even integers 0 to 28
53       array = new int[ 15 ];
54
55       for ( int counter = 0; counter < array.length; counter++ )
56          array[ counter ] = 2 * counter;
57
58    }  // end method init
59
60    // obtain user input and call method binarySearch
61    public void actionPerformed( ActionEvent actionEvent )
62    {
63       // input also can be obtained with enterField.getText()
64       String searchKey = actionEvent.getActionCommand();
65
66       // initialize display string for new search
67       display = "Portions of array searched\n";
68
```

Outline

BinarySearch.java

Lines 53-56
Allocate **15** **int**s for **array** and populate **array** with even **int**s

Line 61
Invoked when user presses Enter

```
69       // perform binary search
70       int element =
71          binarySearch( array, Integer.parseInt( searchKey ) );
72
73       output.setText( display );
74
75       // display search result
76       if ( element != -1 )
77          resultField.setText(
78             "Found value in element " + element );
79       else
80          resultField.setText( "Value not found" );
81
82    }  // end method actionPerformed
83
84    // method to perform binary search of an array
85    public int binarySearch( int array2[], int key )
86    {
87       int low = 0;                  // low element subscript
88       int high = array.length - 1;  // high element subscript
89       int middle;                   // middle element subscript
90
91       // loop until low subscript is greater than high subscript
92       while ( low <= high ) {
93
94          // determine middle element subscript
95          middle = ( low + high ) / 2;
96
97          // display subset of array elements used in this
98          // iteration of binary search loop
99          buildOutput( array2, low, middle, high );
100
101         // if key matches middle element, return middle location
102         if ( key == array[ middle ] )
103            return middle;
```

Outline

BinarySearch.java

Lines 70-71
Invoke method **binarySearch**, using **array** and search key as arguments

Lines 102-103
If search key matches middle **array** element, **return** element subscript

39

```
104
105         // if key less than middle element, set new high element
106         else if ( key < array[ middle ] )
107            high = middle - 1;
108
109         // key greater than middle element, set new low element
110         else
111            low = middle + 1;
112      }
113
114      return -1;   // key not found
115
116   }  // end method binarySearch
117
118   // build row of output showing subset of array elements
119   // currently being processed
120   void buildOutput( int array3[],
121      int low, int middle, int high )
122   {
123      // create 2-digit integer number format
124      DecimalFormat twoDigits = new DecimalFormat( "00" );
125
126      // loop through array elements
127      for ( int counter = 0; counter < array3.length;
128         counter++ ) {
129
130         // if counter outside current array subset, append
131         // padding spaces to String display
132         if ( counter < low || counter > high )
133            display += "    ";
134
```

Outline

BinarySearch.java

Lines 106-107
If search key is less than middle **array** element, repeat search on first **array** half

Lines 110-111
If search key is greater than middle **array** element, repeat search on second **array** half

Lines 120-121
Method **build-Output** displays **array** contents being searched

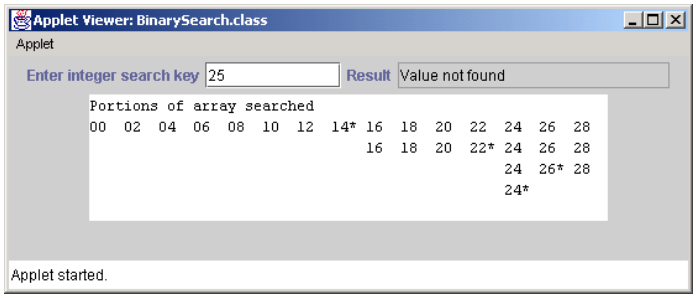
40

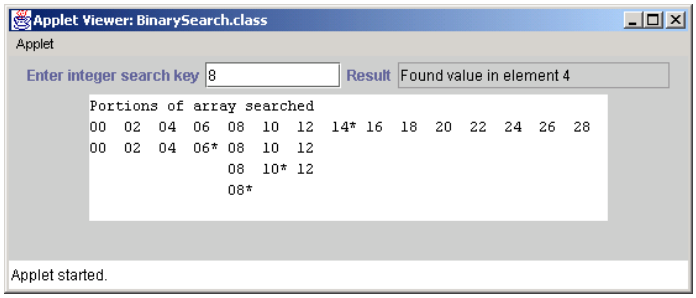
```
135          // if middle element, append element to String display
136          // followed by asterisk (*) to indicate middle element
137          else if ( counter == middle )
138             display +=
139                twoDigits.format( array3[ counter ] ) + "* ";
140
141          // append element to String display
142          else
143             display +=
144                twoDigits.format( array3[ counter ] ) + "  ";
145
146       }  // end for structure
147
148       display += "\n";
149
150    }  // end method buildOutput
151
152 }  // end class BinarySearch
```

Outline

BinarySearch.java

Line 139
Display an asterisk next to middle element





Outline

BinarySearch.java

Multiple-Subscripted Arrays

- Multiple-subscripted arrays
  - Tables with rows and columns
    - Double-subscripted (two-dimensional) array
    - Declaring double-subscripted 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**

43

Multiple-Subscripted Arrays

- Allocating multiple-subscripted arrays
  - Can be allocated dynamically
    - **3**-by-**4** array

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

    - Rows can have different number of columns

```
int b[][];
b = new int[ 2 ][ ];   // allocate rows
b[ 0 ] = new int[ 5 ]; // allocate row 0
b[ 1 ] = new int[ 3 ]; // allocate row 1
```

44

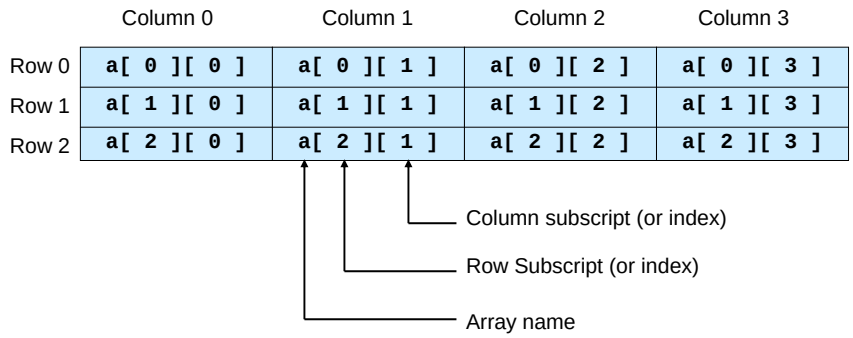


Fig. 7.14 A double-subscripted array with three rows and four columns.

45

```
2  // Initializing multidimensional arrays
3
4  // Java core packages
5  import java.awt.Container;
6
7  // Java extension packages
8  import javax.swing.*;
9
10 public class InitArray extends JApplet {
11    JTextArea outputArea;
12
13    // set up GUI and initialize applet
14    public void init()
15    {
16       outputArea = new JTextArea();
17       Container container = getContentPane();
18       container.add( outputArea );
19
20       int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
21       int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
22
23       outputArea.setText( "Values in array1 by row are\n" );
24       buildOutput( array1 );
25
26       outputArea.append( "\nValues in array2 by row are\n" );
27       buildOutput( array2 );
28    }
29
30    // append rows and columns of an array to outputArea
31    public void buildOutput( int array[][] )
32    {
```

Outline

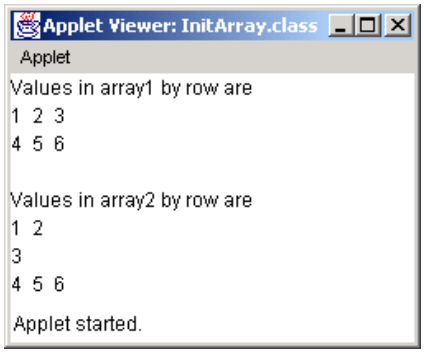
InitArray.java

Line 20
Declare **array1** with six initializers in two sublists

Line 21
Declare **array2** with six initializers in three sublists

46

```
33       // loop through array's rows
34       for ( int row = 0; row < array.length; row++ ) {
35
36          // loop through columns of current row
37          for ( int column = 0;
38                column < array[ row ].length;
39                column++ )
40             outputArea.append( array[ row ][ column ] + "  " );
41
42          outputArea.append( "\n" );
43       }
44    }
45 }
```



Outline

`InitArray.java`

Line 38
`array[row].length` returns number of columns associated with **`row`** subscript

Line 40
Use double-bracket notation to access double-subscripted array values

47

```
2  // Double-subscripted array example
3
4  // Java core packages
5  import java.awt.*;
6
7  // Java extension packages
8  import javax.swing.*;
9
10 public class DoubleArray extends JApplet {
11    int grades[][] = { { 77, 68, 86, 73 },
12                       { 96, 87, 89, 81 },
13                       { 70, 90, 86, 81 } };
14
15    int students, exams;
16    String output;
17    JTextArea outputArea;
18
19    // initialize instance variables
20    public void init()
21    {
22       students = grades.length;     // number of students
23       exams = grades[ 0 ].length;   // number of exams
24
25       // create JTextArea and attach to applet
26       outputArea = new JTextArea();
27       Container container = getContentPane();
28       container.add( outputArea );
29
30       // build output string
31       output = "The array is:\n";
32       buildString();
33
```

Outline

`DoubleArray.java`

Lines 11-13
Declare **`grades`** as 3-by-4 array

Lines 11-13
Each row represents a student; each column represents an exam grade

48

```
34       // call methods minimum and maximum
35       output += "\n\nLowest grade: " + minimum() +
36                 "\nHighest grade: " + maximum() + "\n";
37
38       // call method average to calculate each student's average
39       for ( int counter = 0; counter < students; counter++ )
40          output += "\nAverage for student " + counter + " is " +
41             average( grades[ counter ] );
42
43       // change outputArea's display font
44       outputArea.setFont(
45          new Font( "Courier", Font.PLAIN, 12 ) );
46
47       // place output string in outputArea
48       outputArea.setText( output );
49    }
50
51    // find minimum grade
52    public int minimum()
53    {
54       // assume first element of grades array is smallest
55       int lowGrade = grades[ 0 ][ 0 ];
56
57       // loop through rows of grades array
58       for ( int row = 0; row < students; row++ )
59
60          // loop through columns of current row
61          for ( int column = 0; column < exams; column++ )
62
63             // Test if current grade is less than lowGrade.
64             // If so, assign current grade to lowGrade.
65             if ( grades[ row ][ column ] < lowGrade )
66                lowGrade = grades[ row ][ column ];
67
68       return lowGrade;  // return lowest grade
```

Outline

`DoubleArray.java`

Lines 35-41
Determine minimum, maximum and average for each student

Lines 65-66
Use a nested loop to search for lowest grade in series


```
69    }
70
71    // find maximum grade
72    public int maximum()
73    {
74       // assume first element of grages array is largest
75       int highGrade = grades[ 0 ][ 0 ];
76
77       // loop through rows of grades array
78       for ( int row = 0; row < students; row++ )
79
80          // loop through columns of current row
81          for ( int column = 0; column < exams; column++ )
82
83             // Test if current grade is greater than highGrade.
84             // If so, assign current grade to highGrade.
85             if ( grades[ row ][ column ] > highGrade )
86                highGrade = grades[ row ][ column ];
87
88       return highGrade;  // return highest grade
89    }
90
91    // determine average grade for particular
92    // student (or set of grades)
93    public double average( int setOfGrades[] )
94    {
95       int total = 0;  // initialize total
96
97       // sum grades for one student
98       for ( int count = 0; count < setOfGrades.length; count++ )
99          total += setOfGrades[ count ];
100
101      // return average of grades
102      return ( double ) total / setOfGrades.length;
103   }
```

Outline

`DoubleArray.java`

Lines 85-86
Use a nested loop to search for highest grade in series

Line 93
Method **`average`** takes array of student test results as parameter

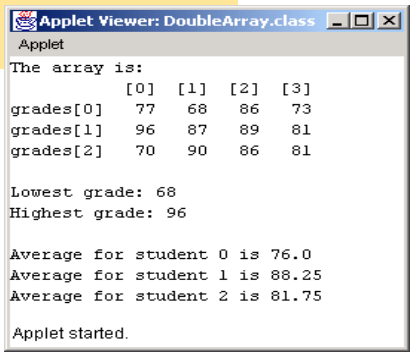
Lines 98-99
Calculate sum of array elements

Line 102
Divide by number of elements to get average

Outline

DoubleArray.java

```
104
105    // build output string
106    public void buildString()
107    {
108       output += "            ";  // used to align column heads
109
110       // create column heads
111       for ( int counter = 0; counter < exams; counter++ )
112          output += "[" + counter + "]  ";
113
114       // create rows/columns of text representing array grades
115       for ( int row = 0; row < students; row++ ) {
116          output += "\ngrades[" + row + "]   ";
117
118          for ( int column = 0; column < exams; column++ )
119             output += grades[ row ][ column ] + "    ";
120       }
121    }
122 }
```



51

Outline

字母	A	B	C	D	E	F	G	H	J	K	L	M	N
代號	10	11	12	13	14	15	16	17	18	19	20	21	22
字母	P	Q	R	S	T	U	V	X	Y	W	Z	I	O
代號	23	24	25	26	27	28	29	30	31	32	33	34	35

身份證公式

52