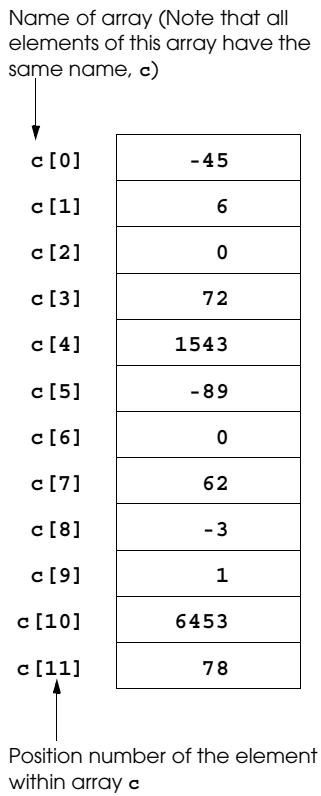


Fig. 7.1 A 12-element array.

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.

```
1  // Fig. 7.3: InitArray.java
2  // initializing an array
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        String output = "";
9        int n[];             // declare reference to an array
10
11       n = new int[ 10 ];  // dynamically allocate array
12
13       output += "Subscript\tValue\n";
14
15       for ( int i = 0; i < n.length; i++ )
16          output += i + "\t" + n[ i ] + "\n";
17
18       JTextArea outputArea = new JTextArea( 11, 10 );
19       outputArea.setText( output );
20
21       JOptionPane.showMessageDialog( null, outputArea,
22          "Initializing an Array of int Values",
23          JOptionPane.INFORMATION_MESSAGE );
24
25       System.exit( 0 );
26    }
27 }
```

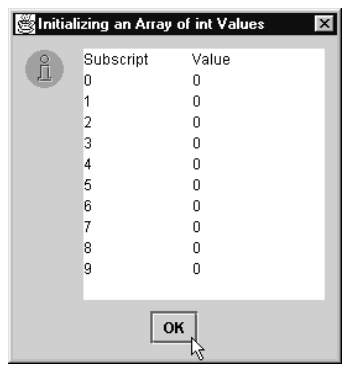


Fig. 7.3 Initializing the elements of an array to zeros.

```
1  // Fig. 7.4: InitArray.java
2  // initializing an array with a declaration
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        String output = "";
9
10       // Initializer list specifies number of elements and
11       // value for each element.
12       int n[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
13
14       output += "Subscript\tValue\n";
15
16       for ( int i = 0; i < n.length; i++ )
17          output += i + "\t" + n[ i ] + "\n";
18
19       JTextArea outputArea = new JTextArea( 11, 10 );
20       outputArea.setText( output );
21
22       JOptionPane.showMessageDialog( null, outputArea,
23          "Initializing an Array with a Declaration",
24          JOptionPane.INFORMATION_MESSAGE );
25
26       System.exit( 0 );
27    }
28 }
```

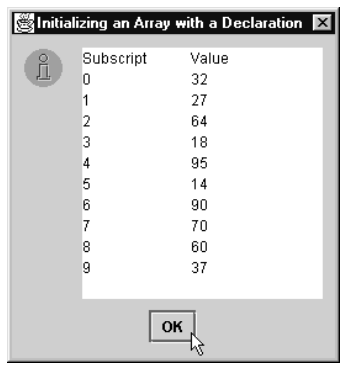


Fig. 7.4 Initializing the elements of an array with a declaration.

```
1  // Fig. 7.5: InitArray.java
2  // initialize array n to the even integers from 2 to 20
3  import javax.swing.*;
4
5  public class InitArray {
6     public static void main( String args[] )
7     {
8        final int ARRAY_SIZE = 10;
9        int n[];                      // reference to int array
10       String output = "";
11
12       n = new int[ ARRAY_SIZE ];  // allocate array
13
14       // Set the values in the array
15       for ( int i = 0; i < n.length; i++ )
16          n[ i ] = 2 + 2 * i;
17
18       output += "Subscript\tValue\n";
19
20       for ( int i = 0; i < n.length; i++ )
21          output += i + "\t" + n[ i ] + "\n";
22
23       JTextArea outputArea = new JTextArea( 11, 10 );
24       outputArea.setText( output );
25
26       JOptionPane.showMessageDialog( null, outputArea,
27          "Initializing to Even Numbers from 2 to 20",
28          JOptionPane.INFORMATION_MESSAGE );
29
30       System.exit( 0 );
31    }
32 }
```

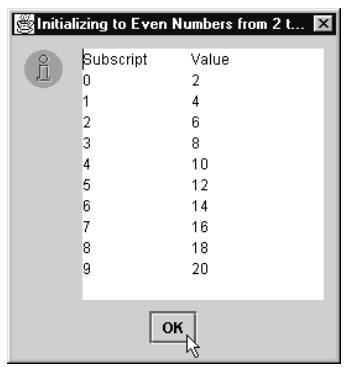


Fig. 7.5 Generating values to be placed into elements of an array.

```
1  // Fig. 7.6: SumArray.java
2  // Compute the sum of the elements of the array
3  import javax.swing.*;
4
5  public class SumArray {
6     public static void main( String args[] )
7     {
8        int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
9        int total = 0;
10
11       for ( int i = 0; i < a.length; i++ )
12          total += a[ i ];
13
```

Fig. 7.6 Computing the sum of the elements of an array (part 1 of 2).

```
14       JOptionPane.showMessageDialog( null,
15          "Total of array elements: " + total,
16          "Sum the Elements of an Array",
17          JOptionPane.INFORMATION_MESSAGE );
18
19       System.exit( 0 );
20    }
21 }
```

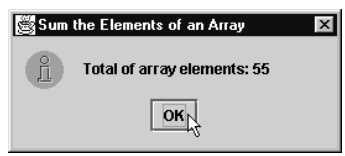


Fig. 7.6 Computing the sum of the elements of an array (part 2 of 2).

```
1  // Fig. 7.7: StudentPoll.java
2  // Student poll program
3  import javax.swing.*;
4
5  public class StudentPoll {
6     public static void main( String args[] )
7     {
8        int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
9                            1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
10                           6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
11                           5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
12       int frequency[] = new int[ 11 ];
13       String output = "";
14
15       for ( int answer = 0;                    // initialize
16             answer < responses.length;         // condition
17             answer++ )                         // increment
18          ++frequency[ responses[ answer ] ];
19
20       output += "Rating\tFrequency\n";
21
22       for ( int rating = 1;
23             rating < frequency.length;
24             rating++ )
25          output += rating + "\t" + frequency[ rating ] + "\n";
26
27       JTextArea outputArea = new JTextArea( 11, 10 );
28       outputArea.setText( output );
29
30       JOptionPane.showMessageDialog( null, outputArea,
31          "Student Poll Program",
32          JOptionPane.INFORMATION_MESSAGE );
33
34       System.exit( 0 );
35    }
36 }
```

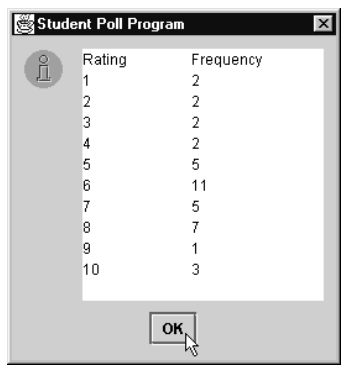


Fig. 7.7 A simple student-poll analysis program.

```
1  // Fig. 7.8: Histogram.java
2  // Histogram printing program
3  import javax.swing.*;
4
5  public class Histogram {
6     public static void main( String args[] )
7     {
8        int n[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
9        String output = "";
10
11       output += "Element\tValue\tHistogram";
12
13       for ( int i = 0; i < n.length; i++ ) {
14          output += "\n" + i + "\t" + n[ i ] + "\t";
15
16          for ( int j = 1; j <= n[ i ]; j++ ) // print a bar
17             output += "*";
18       }
19
20       JTextArea outputArea = new JTextArea( 11, 30 );
21       outputArea.setText( output );
22
23       JOptionPane.showMessageDialog( null, outputArea,
24          "Histogram Printing Program",
25          JOptionPane.INFORMATION_MESSAGE );
26
27       System.exit( 0 );
28    }
29 }
```

Fig. 7.8 A program that prints histograms (part 1 of 2).

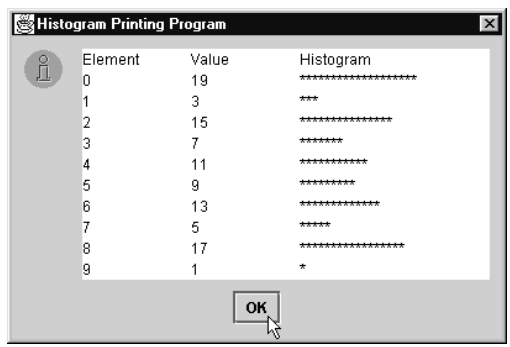


Fig. 7.8 A program that prints histograms (part 2 of 2).

```
1  // Fig. 7.9: RollDie.java
2  // Roll a six-sided die 6000 times
3  import javax.swing.*;
4
5  public class RollDie {
6     public static void main( String args[] )
7     {
8        int face, frequency[] = new int[ 7 ];
9        String output = "";
10
11       for ( int roll = 1; roll <= 6000; roll++ ) {
12          face = 1 + ( int ) ( Math.random() * 6 );
13          ++frequency[ face ];
14       }
15
16       output += "Face\tFrequency";
17
18       for ( face = 1; face < frequency.length; face++ )
19          output += "\n" + face + "\t" + frequency[ face ];
```

Fig. 7.9 Dice-rolling program using arrays instead of **`switch`** (part 1 of 2).

```
20
21       JTextArea outputArea = new JTextArea( 7, 10 );
22       outputArea.setText( output );
23
24       JOptionPane.showMessageDialog( null, outputArea,
25          "Rolling a Die 6000 Times",
26          JOptionPane.INFORMATION_MESSAGE );
27
28       System.exit( 0 );
29    }
30 }
```

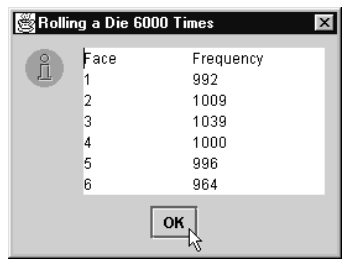


Fig. 7.9 Dice-rolling program using arrays instead of **`switch`** (part 2 of 2).

```
1  // Fig. 7.10: PassArray.java
2  // Passing arrays and individual array elements to methods
3  import java.awt.Container;
4  import javax.swing.*;
5
6  public class PassArray extends JApplet {
7     JTextArea outputArea;
8     String output;
9
10    public void init()
11    {
12       outputArea = new JTextArea();
13       Container c = getContentPane();
14       c.add( outputArea );
15
16       int a[] = { 1, 2, 3, 4, 5 };
17
18       output = "Effects of passing entire " +
19                "array call-by-reference:\n" +
20                "The values of the original array are:\n";
21
22       for ( int i = 0; i < a.length; i++ )
23          output += "   " + a[ i ];
24
25       modifyArray( a );  // array a passed call-by-reference
26
27       output += "\n\nThe values of the modified array are:\n";
28
29       for ( int i = 0; i < a.length; i++ )
30          output += "   " + a[ i ];
31
32       output += "\n\nEffects of passing array " +
33                 "element call-by-value:\n" +
34                 "a[3] before modifyElement: " + a[ 3 ];
35
36       modifyElement( a[ 3 ] );
37
38       output += "\na[3] after modifyElement: " + a[ 3 ];
39       outputArea.setText( output );
40    }
41
42    public void modifyArray( int b[] )
43    {
44       for ( int j = 0; j < b.length; j++ )
45          b[ j ] *= 2;
46    }
47
48    public void modifyElement( int e )
49    {
50       e *= 2;
51    }
52 }
```

Fig. 7.10 Passing arrays and individual array elements to methods (part 1 of 2).

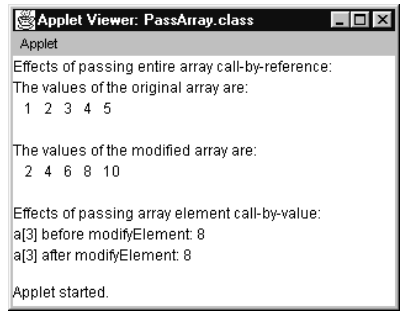


Fig. 7.10 Passing arrays and individual array elements to methods (part 2 of 2).

```
1  // Fig. 7.11: BubbleSort.java
2  // This program sorts an array's values into
3  // ascending order
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class BubbleSort extends JApplet {
8     public void init()
9     {
10       JTextArea outputArea = new JTextArea();
11       Container c = getContentPane();
12       c.add( outputArea );
13
14       int a[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
15
16       String output = "Data items in original order\n";
17
18       for ( int i = 0; i < a.length; i++ )
19          output += "   " + a[ i ];
20
21       bubbleSort( a );
22
23       output += "\n\nData items in ascending order\n";
24
25       for ( int i = 0; i < a.length; i++ )
26          output += "   " + a[ i ];
27
28       outputArea.setText( output );
29    }
30
31    // sort the elements of an array with bubble sort
32    public void bubbleSort( int b[] )
33    {
34       for ( int pass = 1; pass < b.length; pass++ ) // passes
35          for ( int i = 0; i < b.length - 1; i++ ) // one pass
36             if ( b[ i ] > b[ i + 1 ] )        // one comparison
37                swap( b, i, i + 1 );           // one swap
38    }
```

Fig. 7.11 Sorting an array with bubble sort (part 1 of 2).

```
39
40       // swap two elements of an array
41       public void swap( int c[], int first, int second )
42       {
43          int hold;  // temporary holding area for swap
44
45          hold = c[ first ];
46          c[ first ] = c[ second ];
47          c[ second ] = hold;
48       }
49    }
```

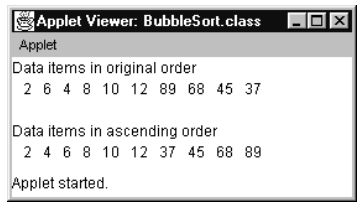
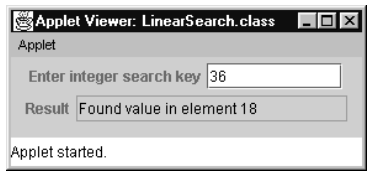


Fig. 7.11 Sorting an array with bubble sort (part 2 of 2).

```
1  // Fig. 7.12: LinearSearch.java
2  // Linear search of an array
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LinearSearch extends JApplet
8                            implements ActionListener {
9     JLabel enterLabel, resultLabel;
10    JTextField enter, result;
11    int a[];
12
13    public void init()
14    {
15       Container c = getContentPane();
16       c.setLayout( new FlowLayout() );
17
18       enterLabel = new JLabel( "Enter integer search key" );
19       c.add( enterLabel );
20
21       enter = new JTextField( 10 );
22       enter.addActionListener( this );
23       c.add( enter );
24
25       resultLabel = new JLabel( "Result" );
26       c.add( resultLabel );
27
28       result = new JTextField( 20 );
29       result.setEditable( false );
30       c.add( result );
31
32       // create array and populate with even integers 0 to 198
33       a = new int[ 100 ];
34
35       for ( int i = 0; i < a.length; i++ )
36          a[ i ] = 2 * i;
37
38    }
39
40    // Search "array" for the specified "key" value
41    public int linearSearch( int array[], int key )
42    {
43       for ( int n = 0; n < a.length; n++ )
44          if ( array[ n ] == key )
45             return n;
46
47       return -1;
48    }
```

Fig. 7.12 Linear search of an array (part 1 of 2).

```
49
50    public void actionPerformed( ActionEvent e )
51    {
52       String searchKey = e.getActionCommand();
53
54       // Array a is passed to linearSearch even though it
55       // is an instance variable. Normally an array will
56       // be passed to a method for searching.
57       int element =
58          linearSearch( a, Integer.parseInt( searchKey ) );
59
60       if ( element != -1 )
61          result.setText( "Found value in element " +
62                          element );
63       else
64          result.setText( "Value not found" );
65    }
66 }
```



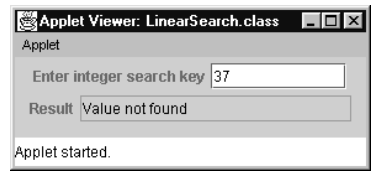


Fig. 7.12 Linear search of an array (part 2 of 2).

```
1  // Fig. 7.13: BinarySearch.java
2  // Binary search of an array
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import java.text.*;
7
8  public class BinarySearch extends JApplet
9                             implements ActionListener {
10    JLabel enterLabel, resultLabel;
11    JTextField enter, result;
12    JTextArea output;
13
14    int a[];
15    String display = "";
16
17    public void init()
18    {
19       Container c = getContentPane();
20       c.setLayout( new FlowLayout() );
21
22       enterLabel = new JLabel( "Enter key" );
23       c.add( enterLabel );
24
25       enter = new JTextField( 5 );
26       enter.addActionListener( this );
27       c.add( enter );
28
29       resultLabel = new JLabel( "Result" );
30       c.add( resultLabel );
31
32       result = new JTextField( 22 );
33       result.setEditable( false );
34       c.add( result );
35
36       output = new JTextArea( 6, 60 );
37       output.setFont(
38          new Font( "Courier", Font.PLAIN, 12 ) );
39       c.add( output );
40
41       // create array and fill with even integers 0 to 28
42       a = new int[ 15 ];
43
44       for ( int i = 0; i < a.length; i++ )
45          a[ i ] = 2 * i;
46    }
47
```

Fig. 7.13 Binary search of a sorted array (part 1 of 3).

```
48    public void actionPerformed( ActionEvent e )
49    {
50       String searchKey = e.getActionCommand();
51
52       // initialize display string for the new search
53       display = "Portions of array searched\n";
54
55       // perform the binary search
56       int element =
57          binarySearch( a, Integer.parseInt( searchKey ) );
58
59       output.setText( display );
60
61       if ( element != -1 )
62          result.setText(
63             "Found value in element " + element );
64       else
65          result.setText( "Value not found" );
66    }
67
68    // Binary search
69    public int binarySearch( int array[], int key )
70    {
71       int low = 0;                    // low subscript
72       int high = array.length - 1;    // high subscript
73       int middle;                     // middle subscript
74
75       while ( low <= high ) {
76          middle = ( low + high ) / 2;
77
78          // The following line is used to display the part
79          // of the array currently being manipulated during
80          // each iteration of the binary search loop.
81          buildOutput( low, middle, high );
82
83          if ( key == array[ middle ] )  // match
84             return middle;
85          else if ( key < array[ middle ] )
86             high = middle - 1;  // search low end of array
87          else
88             low = middle + 1;   // search high end of array
89       }
90
91       return -1;   // searchKey not found
92    }
93
94    // Build one row of output showing the current
95    // part of the array being processed.
96    void buildOutput( int low, int mid, int high )
97    {
98       DecimalFormat twoDigits = new DecimalFormat( "00" );
99
```

Fig. 7.13 Binary search of a sorted array (part 2 of 3).

```
100         for ( int i = 0; i < a.length; i++ ) {
101            if ( i < low || i > high )
102               display += "    ";
103            else if ( i == mid ) // mark middle element in output
104               display += twoDigits.format( a[ i ] ) + "* ";
105            else
106               display += twoDigits.format( a[ i ] ) + "  ";
107         }
108
109         display += "\n";
110      }
111   }
```

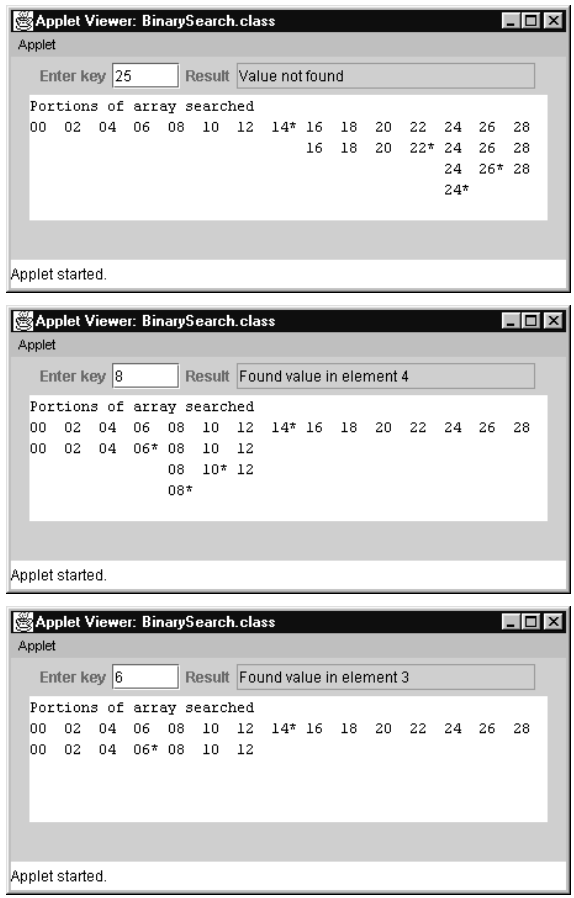


Fig. 7.13 Binary search of a sorted array (part 3 of 3).

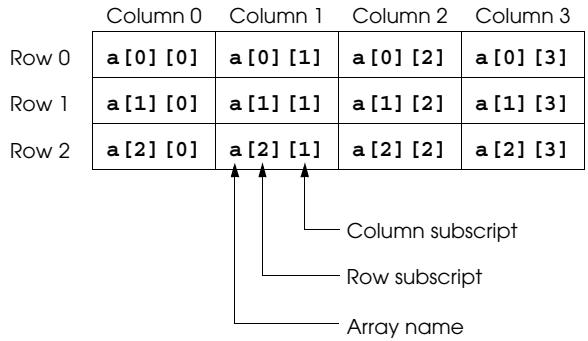


Fig. 7.14 A double-subscripted array with three rows and four columns.

```
1  // Fig. 7.15: InitArray.java
2  // Initializing multidimensional arrays
3  import java.awt.Container;
4  import javax.swing.*;
5
6  public class InitArray extends JApplet {
7     JTextArea outputArea;
8
9     // paint the applet
10    public void init()
11    {
12       outputArea = new JTextArea();
13       Container c = getContentPane();
14       c.add( outputArea );
15
16       int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
17       int array2[][] = { { 1, 2 }, { 3 }, { 4, 5, 6 } };
18
19       outputArea.setText( "Values in array1 by row are\n" );
20       buildOutput( array1 );
21
22       outputArea.append( "\nValues in array2 by row are\n" );
23       buildOutput( array2 );
24    }
25
26    public void buildOutput( int a[][] )
27    {
28       for ( int i = 0; i < a.length; i++ ) {
29
30          for ( int j = 0; j < a[ i ].length; j++ )
31             outputArea.append( a[ i ][ j ] + "  " );
32
33          outputArea.append( "\n" );
34       }
35    }
36 }
```

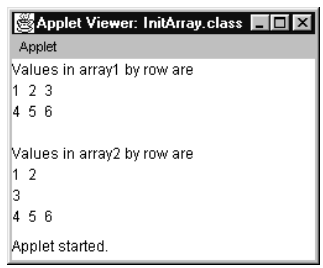


Fig. 7.15 Initializing multidimensional arrays.

```
1  // Fig. 7.16: DoubleArray.java
2  // Double-subscripted array example
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class DoubleArray extends JApplet {
7     int grades[][] = { { 77, 68, 86, 73 },
8                        { 96, 87, 89, 81 },
9                        { 70, 90, 86, 81 } };
```

Fig. 7.16 Example of using double-subscripted arrays (part 1 of 3).

```
10    int students, exams;
11    String output;
12    JTextArea outputArea;
13
14    // initialize instance variables
15    public void init()
16    {
17       students = grades.length;
18       exams = grades[ 0 ].length;
19
20       outputArea = new JTextArea();
21       Container c = getContentPane();
22       c.add( outputArea );
23
24       // build the output string
25       output = "The array is:\n";
26       buildString();
27
28       output += "\n\nLowest grade: " + minimum() +
29                 "\nHighest grade: " + maximum() + "\n";
30
31       for ( int i = 0; i < students; i++ )
32          output += "\nAverage for student " + i + " is " +
33                    average( grades[ i ] );
34
35       outputArea.setFont(
36          new Font( "Courier", Font.PLAIN, 12 ) );
37       outputArea.setText( output );
38    }
39
40    // find the minimum grade
41    public int minimum()
42    {
43       int lowGrade = 100;
44
45       for ( int i = 0; i < students; i++ )
46          for ( int j = 0; j < exams; j++ )
47             if ( grades[ i ][ j ] < lowGrade )
48                lowGrade = grades[ i ][ j ];
49
50       return lowGrade;
51    }
52
```

```
53      // find the maximum grade
54      public int maximum()
55      {
56         int highGrade = 0;
57
58         for ( int i = 0; i < students; i++ )
59            for ( int j = 0; j < exams; j++ )
60               if ( grades[ i ][ j ] > highGrade )
61                  highGrade = grades[ i ][ j ];
62
```

Fig. 7.16 Example of using double-subscripted arrays (part 2 of 3).

```
63         return highGrade;
64      }
65
66      // determine the average grade for a particular
67      // student (or set of grades)
68      public double average( int setOfGrades[] )
69      {
70         int total = 0;
71
72         for ( int i = 0; i < setOfGrades.length; i++ )
73            total += setOfGrades[ i ];
74
75         return ( double ) total / setOfGrades.length;
76      }
77
78      // build output string
79      public void buildString()
80      {
81         output += "            ";  // used to align column heads
82
83         for ( int i = 0; i < exams; i++ )
84            output += "[" + i + "]  ";
85
86         for ( int i = 0; i < students; i++ ) {
87            output += "\ngrades[" + i + "]   ";
88
89            for ( int j = 0; j < exams; j++ )
90               output += grades[ i ][ j ] + "   ";
91         }
92      }
93   }
```

Fig. 7.16 Example of using double-subscripted arrays (part 3 of 3).