Chapter 5 Arrays (Main Page)

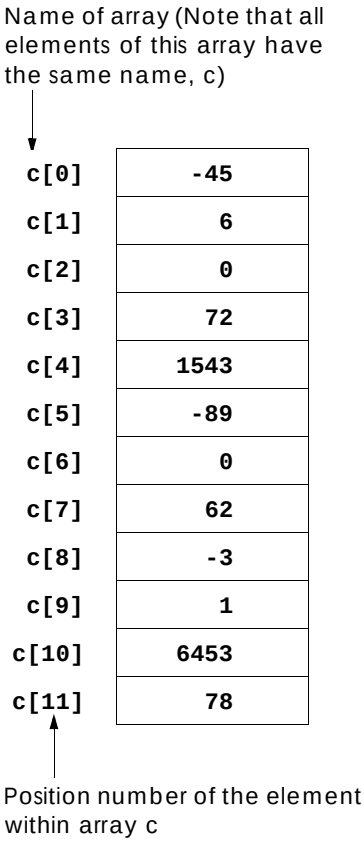


Fig. 5.1 A 12-element array.

Operators	Associativity	Type
`( ) [ ] .`	left to right	parentheses
`++ -- + - !` (*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

Fig. 5.2 Operator precedence and associativity.

Operators	Associativity	Type
&&	left to right	logical AND
||	left to right	logical OR
?:	right to left	conditional
= += -= *= /= %=	right to left	assignment

Fig. 5.2 Operator precedence and associativity.

```
1  // Fig. 5.3: InitArray.java
2  // initializing an array
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class InitArray extends Applet {
7     int n[];        // declare an array of integers
8
9     // initialize instance variables
10    public void init()
11    {
12       n = new int[ 10 ];  // dynamically allocate array
13    }
14
15    // paint the applet
16    public void paint( Graphics g )
17    {
18       int yPosition = 25;  // starting y position on applet
19
20       g.drawString( "Element", 25, yPosition );
21       g.drawString( "Value", 100, yPosition );
22
23       for ( int i = 0; i < n.length; i++ ) {
24          yPosition += 15;
25          g.drawString( String.valueOf( i ), 25, yPosition );
26          g.drawString( String.valueOf( n[ i ] ),
27                        100, yPosition );
28       }
```

Fig. 5.3 Initializing the elements of an array to zeros (part 1 of 2).

```
29      }
30   }
```

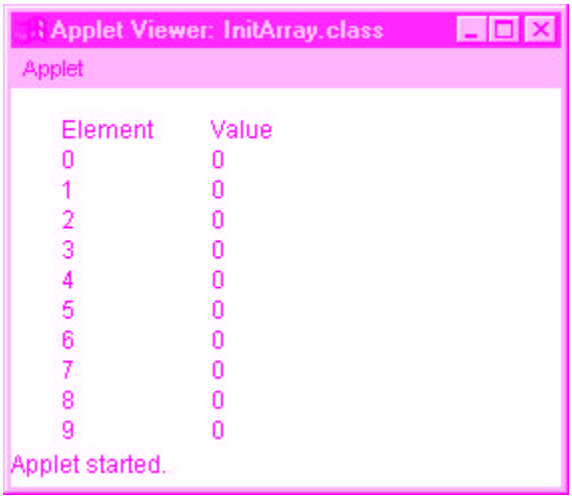


Fig. 5.3 Initializing the elements of an array to zeros (part 2 of 2).

```
1   // Fig. 5.4: InitArray.java
2   // initializing an array with a declaration
3   import java.awt.Graphics;
4   import java.applet.Applet;
5
6   public class InitArray extends Applet {
7      int n[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
8
9      // paint the applet
10     public void paint( Graphics g )
11     {
12        int yPosition = 25;   // starting y position on applet
13
14        g.drawString( "Element", 25, yPosition );
15        g.drawString( "Value", 100, yPosition );
16
17        for ( int i = 0; i < n.length; i++ ) {
18           yPosition += 15;
19           g.drawString( String.valueOf( i ), 25, yPosition );
20           g.drawString( String.valueOf( n[ i ] ),
21                         100, yPosition );
```

Fig. 5.4 Initializing the elements of an array with a declaration (part 1 of 2).

```
22          }
23       }
24    }
```

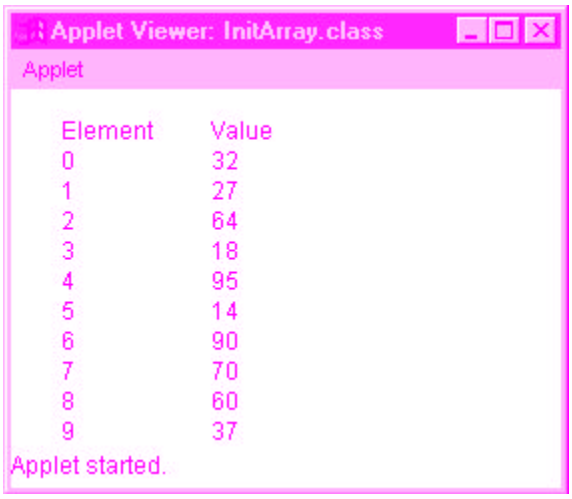


Fig. 5.4 Initializing the elements of an array with a declaration (part 2 of 2).

```
1  // Fig. 5.5: InitArray.java
2  // initialize array s to the even integers from 2 to 20
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class InitArray extends Applet {
7     final int ARRAY_SIZE = 10;
8     int s[];
9
10    // initialize instance variables
11    public void init()
12    {
13       s = new int[ ARRAY_SIZE ];
14
15       // Set the values in the array
16       for ( int i = 0; i < s.length; i++ )
17          s[ i ] = 2 + 2 * i;
18    }
19
20    // paint the applet
21    public void paint( Graphics g )
22    {
23       int yPosition = 25;   // starting y position on applet
24
25       g.drawString( "Element", 25, yPosition );
26       g.drawString( "Value", 100, yPosition );
27
28       for ( int i = 0; i < s.length; i++ ) {
29          yPosition += 15;
30          g.drawString( String.valueOf( i ), 25, yPosition );
31          g.drawString( String.valueOf( s[ i ] ),
32                        100, yPosition );
33       }
```

Fig. 5.5 Generating values to be placed into elements of an array (part 1 of 2).

```
34      }
35   }
```

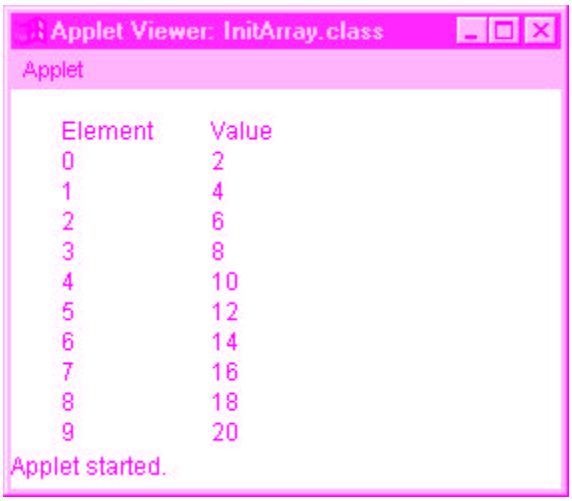


Fig. 5.5 Generating values to be placed into elements of an array (part 2 of 2).

```
1  // Fig. 5.6: FinalTest.java
2  // a final object must be initialized
3  import java.applet.Applet;
4
5  public class FinalTest extends Applet {
6     final int x;  // Error: Final variables must be initialized
7  }
```

```
FinalTest.java:5: Blank final variable 'x' may not have
   been initialized. It must be assigned a value in an
   initializer, or in every constructor.
public class FinalTest extends Applet {
             ^
1 error
```

Fig. 5.6 A `final` object must be initialized.

```
1  // Fig. 5.7: FinalTest.java
2  // using a properly initialized constant variable
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class FinalTest extends Applet {
7     final int x = 7;  // initialize constant variable
8
9     public void paint( Graphics g )
10    {
11       g.drawString( "The value of x is: " + x, 25, 25 );
```

Fig. 5.7 Correctly initializing and using a constant variable (part 1 of 2).

```
12      }
13   }
14
```

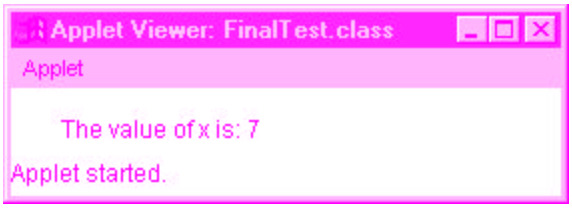


Fig. 5.7 Correctly initializing and using a constant variable (part 2 of 2).

```
1   // Fig. 5.8: SumArray.java
2   // Compute the sum of the elements of the array
3   import java.awt.Graphics;
4   import java.applet.Applet;
5
6   public class SumArray extends Applet {
7      int a[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
8      int total;
9
10     // initialize instance variables
11     public void init()
12     {
13        total = 0;
14
15        for ( int i = 0; i < a.length; i++ )
16           total += a[ i ];
17     }
18
19     // paint the applet
20     public void paint( Graphics g )
21     {
22        g.drawString( "Total of array elements: " + total,
23                      25, 25 );
24     }
25  }
```

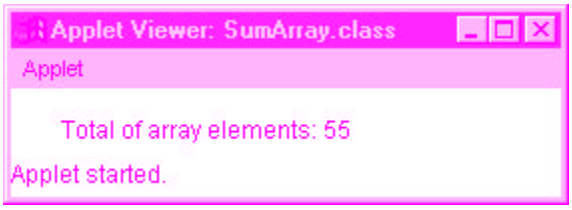


Fig. 5.8 Computing the sum of the elements of an array .

.

```
1    // Fig. 5.9: StudentPoll.java
2   // Student poll program
3   import java.awt.Graphics;
4   import java.applet.Applet;
```

Fig. 5.9 A simple student-poll analysis program (part 1 of 2).

```
5
6  public class StudentPoll extends Applet {
7     int responses[] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
8                         1, 6, 3, 8, 6, 10, 3, 8, 2, 7,
9                         6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
10                        5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };
11    int frequency[];
12
13    // initialize instance variables
14    public void init()
15    {
16       frequency = new int[ 11 ];
17
18      for ( int answer = 0; answer < responses.length; answer++ )
19          ++frequency[ responses[ answer ] ];
20    }
21
22    // paint the applet
23    public void paint( Graphics g )
24    {
25       int yPosition = 25;   // starting y position on applet
26
27       g.drawString( "Rating", 25, yPosition );
28       g.drawString( "Frequency", 100, yPosition );
29
30       for ( int rating = 1;
31             rating < frequency.length; rating++ ) {
32          yPosition += 15;
33          g.drawString( String.valueOf( rating ),
34                        25, yPosition );
35          g.drawString( String.valueOf( frequency[ rating ] ),
36                        100, yPosition );
37       }
38    }
39 }
```

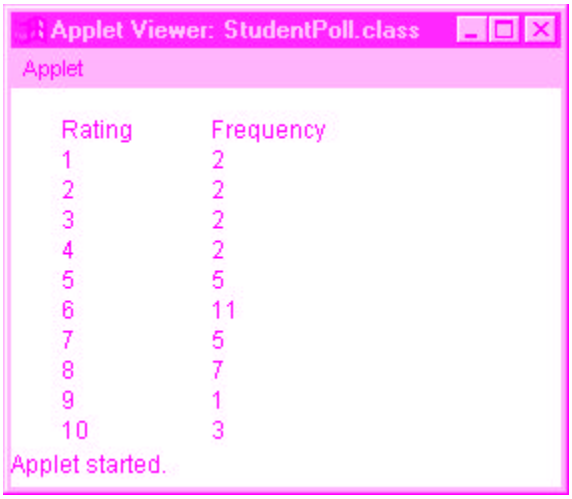


Fig. 5.9 A simple student-poll analysis program (part 2 of 2).

```
1  // Fig. 5.10: Histogram.java
2  // Histogram printing program
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class Histogram extends Applet {
7     int n[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
8
9     // paint the applet
10    public void paint( Graphics g )
11    {
12       int xPosition;       // position of * in histogram
13       int yPosition = 25;  // vertical position in applet
14
15       g.drawString( "Element", 25, yPosition );
16       g.drawString( "Value", 100, yPosition );
17       g.drawString( "Histogram", 175, yPosition );
18
19       for ( int i = 0; i < n.length; i++ ) {
20          yPosition += 15;
21          g.drawString( String.valueOf( i ), 25, yPosition );
22          g.drawString( String.valueOf( n[ i ] ),
23                        100, yPosition );
24          xPosition = 175;
25
26          for ( int j = 1; j <= n[ i ]; j++ ) { // print one bar
27             g.drawString( "*", xPosition, yPosition );
28             xPosition += 7;
29          }
30       }
31    }
32 }
33
```

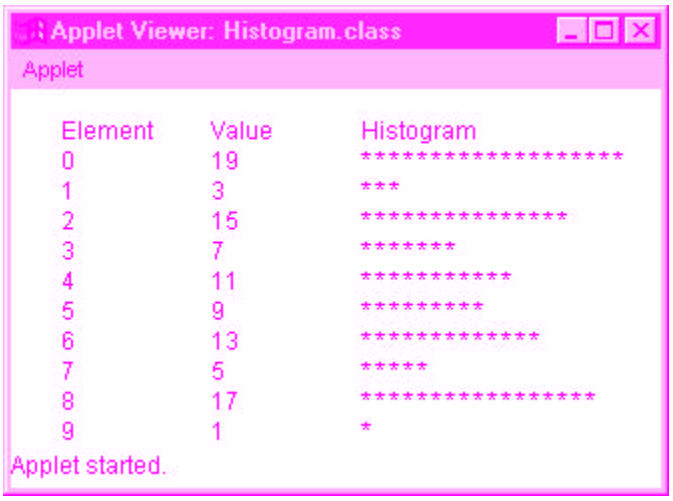


Fig. 5.10 A program that prints histograms .

```
1  // Fig. 5.11: RollDie.java
2  // Roll a six-sided die 6000 times
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class RollDie extends Applet {
7     int face;
8     int frequency[];
9
10    // initialize instance variables
11    public void init()
12    {
13       frequency = new int[ 7 ];
14
15       for ( int roll = 1; roll <= 6000; roll++ ) {
16          face = 1 + (int) ( Math.random() * 6 );
17          ++frequency[ face ];
18       }
19    }
20
21    // paint the applet
22    public void paint( Graphics g )
23    {
24       int yPosition = 25;
25
26       g.drawString( "Face", 25, yPosition );
27       g.drawString( "Frequency", 100, yPosition );
28
29       for ( face = 1; face < frequency.length; face++ ) {
30          yPosition += 15;
31          g.drawString( String.valueOf( face ), 25, yPosition );
32          g.drawString( String.valueOf( frequency[ face ] ),
33                        100, yPosition );
34       }
35    }
36 }
```

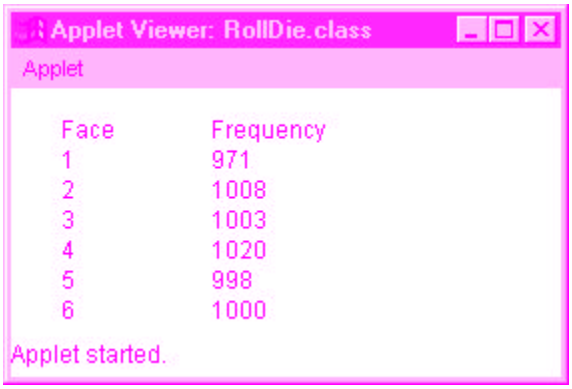


Fig. 5.11 Dice-rolling program using arrays instead of `switch`.

```
1  // Fig. 5.12: PassArray.java
2  // Passing arrays and individual array elements to methods
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class PassArray extends Applet {
7     int a[] = {0, 1, 2, 3, 4};
8
9     public void paint( Graphics g )
10    {
11       int xPosition = 25, yPosition = 25;
12
13       g.drawString(
14          "Effects of passing entire array call-by-reference:",
15          xPosition, yPosition );
16       yPosition += 15;
17       g.drawString( "The values of the original array are:",
18                     xPosition, yPosition );
19       xPosition += 15;
20       yPosition += 15;
21
22       for ( int i = 0; i < a.length; i++ ) {
23          g.drawString( String.valueOf( a[ i ] ),
24                        xPosition, yPosition );
25          xPosition += 15;
26       }
27
28       xPosition = 25;
29       yPosition += 30;
30
31       modifyArray( a );  // array a passed call-by-reference
32
33       g.drawString( "The values of the modified array are:",
34                     xPosition, yPosition );
35       xPosition += 15;
36       yPosition += 15;
37
38       for ( int i = 0; i < a.length; i++ ) {
39          g.drawString( String.valueOf( a[ i ] ),
40                        xPosition, yPosition );
41          xPosition += 15;
42       }
43
44       xPosition = 25;
45       yPosition += 30;
46
47       g.drawString(
48          "Effects of passing array element call-by-value:",
49          xPosition, yPosition );
50       yPosition += 15;
51       g.drawString( "a[3] before modifyElement: " + a[ 3 ],
52                     xPosition, yPosition );
53       yPosition += 15;
54
55       modifyElement( a[ 3 ] );
56
57       g.drawString( "a[3] after modifyElement: " + a[ 3 ],
58                     xPosition, yPosition );
59    }
60
```

Fig. 5.12 Passing arrays and individual array elements to methods (part 1 of 2).

```
61     public void modifyArray( int b[] )
62     {
63        for ( int j = 0; j < b.length; j++ )
64           b[ j ] *= 2;
65     }
66
67     public void modifyElement( int e )
68     {
69        e *= 2;
70     }
71  }
```

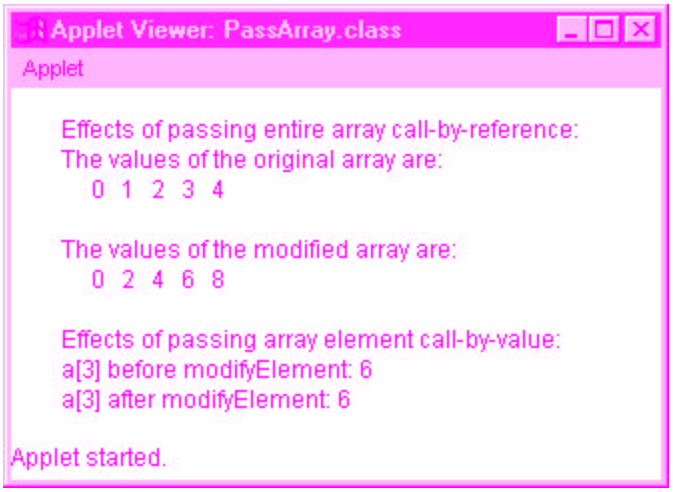


Fig. 5.12 Passing arrays and individual array elements to methods (part 2 of 2).

.

```
1  // Fig. 5.13: BubbleSort.java
2  // This program sorts an array's values into
3  // ascending order
4  import java.awt.Graphics;
5  import java.applet.Applet;
6
7  public class BubbleSort extends Applet {
8     int a[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
9
10    public void paint( Graphics g )
11    {
12       print( g, "Data items in original order", a, 25, 25 );
13
14       sort();
15
16       print( g, "Data items in ascending order", a, 25, 55 );
17    }
18
19    public void sort()
20    {
21       int hold;  // temporary holding area for swap
22
23       for ( int pass = 1; pass < a.length; pass++ ) // passes
24          for ( int i = 0; i < a.length - 1; i++ ) // one pass
```

Fig. 5.13 Sorting an array with bubble sort (part 1 of 2).

```
25             if ( a[ i ] > a[ i + 1 ] ) {      // one comparison
26                hold = a[i];                    // one swap
27                a[ i ] = a[ i + 1 ];
28                a[ i + 1 ] = hold;
29             }
30     }
31
32     public void print( Graphics g, String head, int b[],
33                        int x, int y )
34     {
35        g.drawString( head, x, y );
36        x += 15;
37        y += 15;
38
39        for ( int i = 0; i < b.length; i++ ) {
40           g.drawString( String.valueOf( b[ i ] ), x, y );
41           x += 20;
42        }
43     }
44  }
```

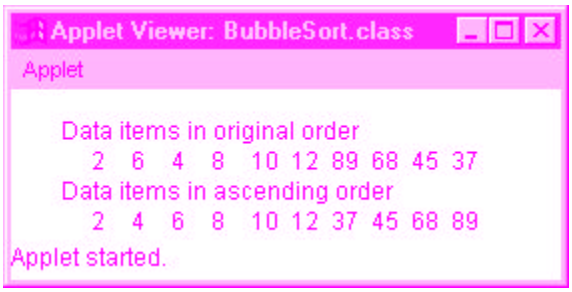


Fig. 5.13 Sorting an array with bubble sort (part 2 of 2).

```
1  // Fig. 5.14: LinearSearch.java
2  // Linear search of an array
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.applet.Applet;
6
7  public class LinearSearch extends Applet
8                implements ActionListener {
9     int a[];
10    int element;
11    String searchKey;
12    Label enterLabel, resultLabel;
13    TextField enter, result;
```

Fig. 5.14 Linear search of an array (part 1 of 3).

```
14
15    public void init()
16    {
17       a = new int[ 100 ];
18
19       for ( int i = 0; i < a.length; i++ ) // create data
20          a[ i ] = 2 * i;
21
22       enterLabel = new Label( "Enter integer search key" );
23       add( enterLabel );
```

```
24
25          enter = new TextField( 10 );
26          enter.addActionListener( this );
27          add( enter );
28
29          resultLabel = new Label( "Result" );
30          add( resultLabel );
31
32          result = new TextField( 25 );
33          result.setEditable( false );
34          add( result );
35       }
36
37       public int linearSearch( int key )
38       {
39          for ( int n = 0; n < a.length; n++ )
40             if ( a[ n ] == key )
41                return n;
42
43          return -1;
44       }
45
46       public void actionPerformed( ActionEvent e )
47       {
48          searchKey = e.getActionCommand();
49          element =
50             linearSearch( Integer.parseInt( searchKey ) );
51
52          if ( element != -1 )
53             result.setText( "Found value in element " +
54                             element );
55          else
56             result.setText( "Value not found" );
57       }
58    }
```

Fig. 5.14 Linear search of an array (part 2 of 3).

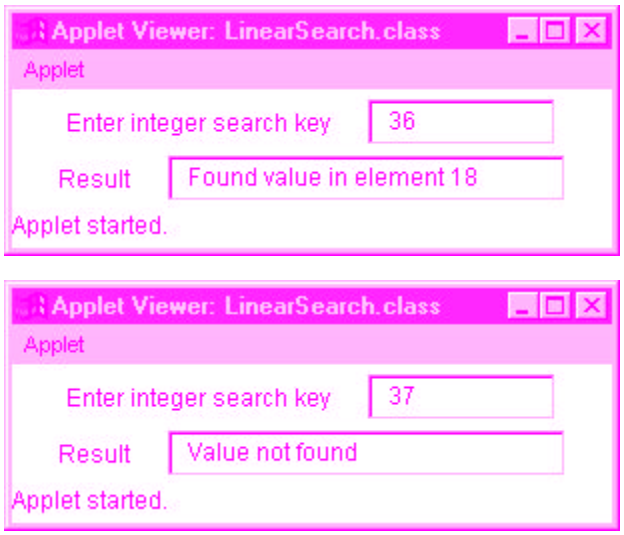


Fig. 5.14 Linear search of an array (part 3 of 3).

```
1  // Fig. 5.15: BinarySearch.java
2  // Binary search of an array
3  import java.awt.*;
4  import java.applet.Applet;
5  import java.awt.event.*;
6
7  public class BinarySearch extends Applet
8                implements ActionListener {
9     int a[];
10    int element;
11    String searchKey;
12    int xPosition;  // applet horizontal drawing position
13    int yPosition;  // applet vertical drawing position
14    Label enterLabel, resultLabel;
15    TextField enter, result;
16    boolean timeToSearch = false;
17
18    public void init()
19    {
20       a = new int[ 15 ];
21
22       for ( int i = 0; i < a.length; i++ ) // create data
23          a[ i ] = 2 * i;
24
25       enterLabel = new Label( "Enter key" );
26       add( enterLabel );
27
28       enter = new TextField( 5 );
29       enter.addActionListener( this );
30       add( enter );
31
32       resultLabel = new Label( "Result" );
33       add( resultLabel );
34
35       result = new TextField( 22 );
36       result.setEditable( false );
37       add( result );
38    }
```

Fig. 5.15 Binary search of a sorted array (part 1 of 4).

```
39
40    public void paint( Graphics g )
41    {
42       if ( timeToSearch ) { // prevents search 1st time called
43          element = binarySearch(
44             Integer.parseInt( searchKey ), g );
45
46          if ( element != -1 )
47             result.setText(
48                "Found value in element " + element );
49          else
50             result.setText( "Value not found" );
51       }
52    }
53
54    public void actionPerformed( ActionEvent event )
55    {
56       timeToSearch = true;
57       xPosition = 25;
58       yPosition = 75;
```

```
59          searchKey = event.getActionCommand().toString();
60          repaint();  // call paint to start search and output
61       }
62
63       // Binary search
64       public int binarySearch( int key, Graphics gg )
65       {
66          gg.drawString( "Portions of array searched",
67                         xPosition, yPosition );
68          yPosition += 15;
69
70          int low = 0;              // low subscript
71          int high = a.length - 1;  // high subscript
72          int middle;               // middle subscript
73
74          while ( low <= high ) {
75             middle = ( low + high ) / 2;
76
77             printRow( low, middle, high, gg );
78
79             if ( key == a[ middle ] )  // match
80                return middle;
81             else if ( key < a[ middle ] )
82                high = middle - 1;  // search low end of array
83             else
84                low = middle + 1;   // search high end of array
85          }
86
87          return -1;   // searchKey not found
88       }
89
```

Fig. 5.15 Binary search of a sorted array (part 2 of 4).

```
90       // Print one row of output showing the current
91       // part of the array being processed.
92       void printRow( int low, int mid, int high, Graphics gg )
93       {
94          xPosition = 25;
95
96          for ( int i = 0; i < a.length; i++ ) {
97             if ( i < low || i > high )
98                gg.drawString( "", xPosition, yPosition );
99             else if ( i == mid )   // mark middle value
100               gg.drawString( String.valueOf( a[ i ] ) + "*",
101                              xPosition, yPosition );
102            else
103               gg.drawString( String.valueOf( a[ i ] ),
104                              xPosition, yPosition );
105
106            xPosition += 20;
107         }
108
109         yPosition += 15;
110      }
111   }
```

Fig. 5.15 Binary search of a sorted array (part 3 of 4).

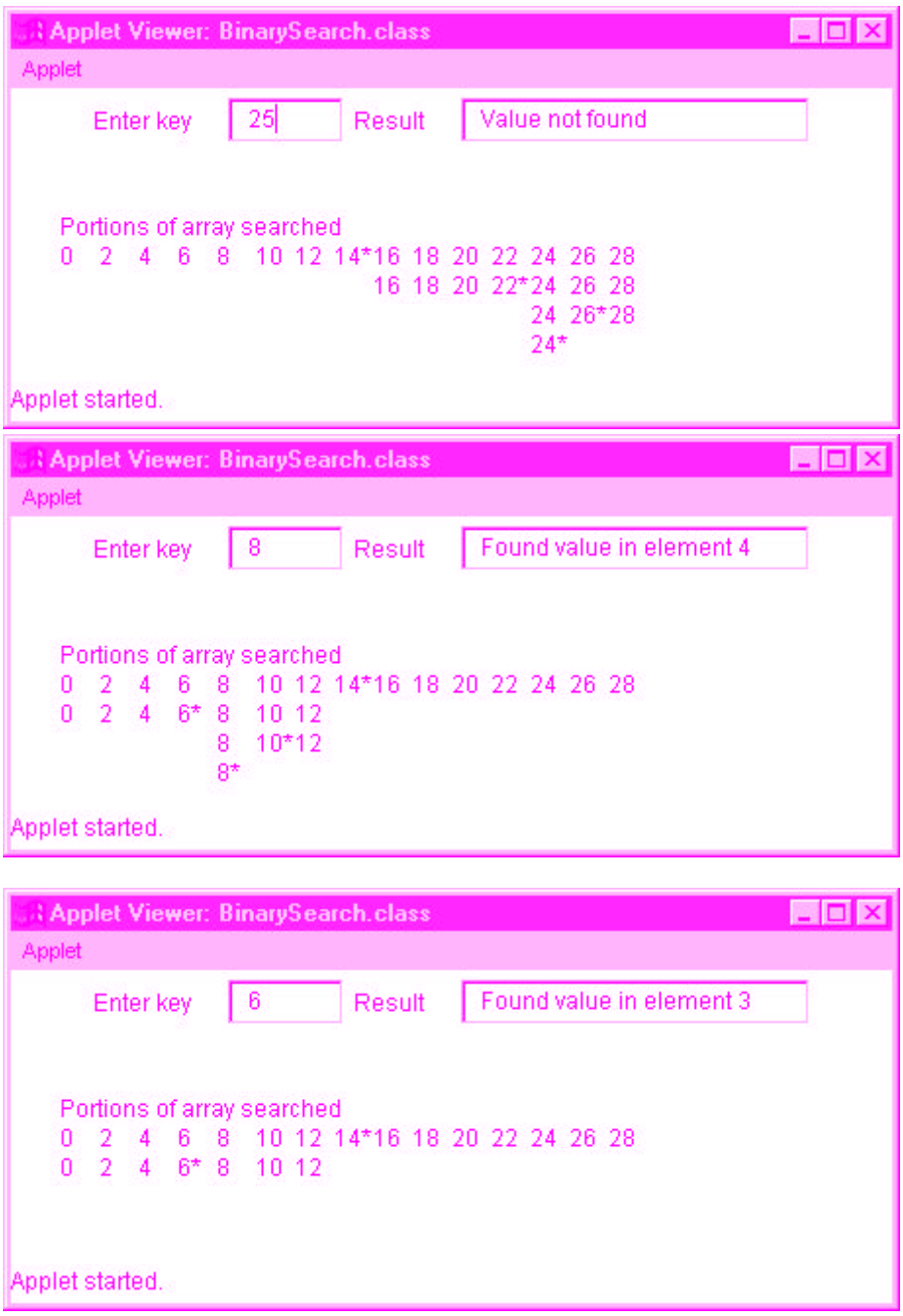


Fig. 5.15 Binary search of a sorted array (part 4 of 4).

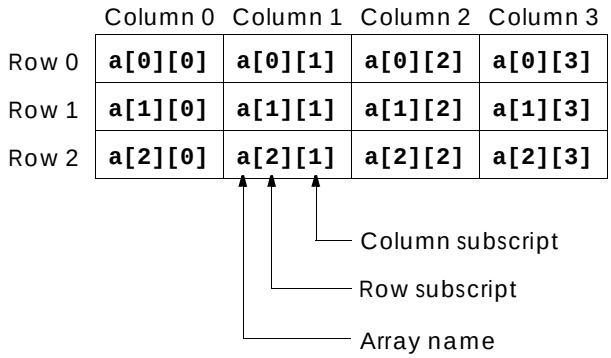
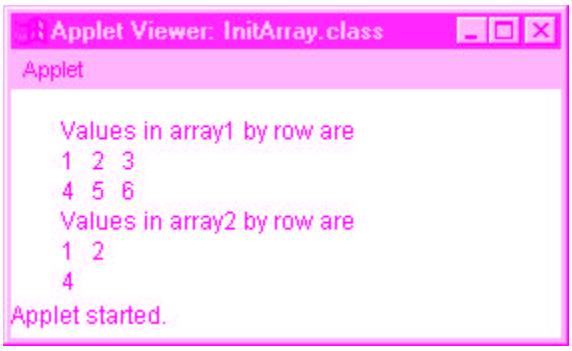


Fig. 5.16 A double-subscripted array with three rows and four columns.

```
1  // Fig. 5.17: InitArray.java
2  // Initializing multidimensional arrays
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class InitArray extends Applet {
7
8     // paint the applet
9     public void paint( Graphics g )
10    {
11       int array1[][] = { { 1, 2, 3 }, { 4, 5, 6 } };
12       int array2[][] = { { 1, 2 }, { 4 } };
13
14       g.drawString( "Values in array1 by row are", 25, 25 );
15       printArray( array1, g, 40 );
16
17       g.drawString( "Values in array2 by row are", 25, 70 );
18       printArray( array2, g, 85 );
19    }
20
21    public void printArray( int a[][], Graphics g, int y )
22    {
23       int x = 25;
24
25       for ( int i = 0; i < a.length; i++ ) {
26
27          for ( int j = 0; j < a[ i ].length; j++ )  {
28             g.drawString( String.valueOf( a[ i ][ j ] ), x, y );
29             x += 15;
30          }
31
32          x = 25;
33          y += 15;
34       }
35    }
36 }
```

Fig. 5.17 Initializing multidimensional arrays.




```
1  // Fig. 5.18: DoubleArray.java
2  // Double-subscripted array example
3  import java.awt.Graphics;
4  import java.applet.Applet;
5
6  public class DoubleArray extends Applet {
7     int grades[][] = { { 77, 68, 86, 73 },
8                        { 96, 87, 89, 81 },
9                        { 70, 90, 86, 81 } };
10    int students, exams;
11    int xPosition, yPosition;
12
13    // initialize instance variables
14    public void init()
15    {
16       students = grades.length;
17       exams = grades[ 0 ].length;
18    }
19
20    // paint the applet
21    public void paint( Graphics g )
22    {
23       xPosition = 25;
24       yPosition = 25;
25
26       g.drawString( "The array is:", xPosition, yPosition );
27       yPosition += 15;
28       printArray( g );
29       xPosition = 25;
30       yPosition += 30;
31       g.drawString( "Lowest grade:", xPosition, yPosition );
32       int min = minimum();
33       g.drawString( String.valueOf( min ),
34                     xPosition + 85, yPosition );
35       yPosition += 15;
36       g.drawString( "Highest grade:", xPosition, yPosition );
37       int max = maximum();
38       g.drawString( String.valueOf( max ),
39                     xPosition + 85, yPosition );
40       yPosition += 15;
41
42       for ( int i = 0; i < students; i++ ) {
43          g.drawString( "Average for student " + i + " is ",
44                        25, yPosition );
45          double ave = average( grades[ i ] );
46          g.drawString( String.valueOf( ave ), 165, yPosition );
47          yPosition += 15;
```

Fig. 5.18 Example of using double-subscripted arrays (part 1 of 3).

```
48          }
49       }
50
51       // find the minimum grade
52       public int minimum()
53       {
54          int lowGrade = 100;
55
56          for ( int i = 0; i < students; i++ )
57             for ( int j = 0; j < exams; j++ )
58                if ( grades[ i ][ j ] < lowGrade )
59                   lowGrade = grades[ i ][ j ];
60
61          return lowGrade;
62       }
63
64       // find the maximum grade
65       public int maximum()
66       {
67          int highGrade = 0;
68
69          for ( int i = 0; i < students; i++ )
70             for ( int j = 0; j < exams; j++ )
71                if ( grades[ i ][ j ] > highGrade )
72                   highGrade = grades[ i ][ j ];
73
74          return highGrade;
75       }
76
77       // determine the average grade for a particular
78       // student (or set of grades)
79       public double average( int setOfGrades[] )
80       {
81          int total = 0;
82
83          for ( int i = 0; i < setOfGrades.length; i++ )
84             total += setOfGrades[ i ];
85
86          return (double) total / setOfGrades.length;
87       }
88
89       // print the array
90       public void printArray( Graphics g )
91       {
92          xPosition = 80;
93
94          for ( int i = 0; i < exams; i++ ) {
95             g.drawString( "[" + i + "]", xPosition, yPosition );
96             xPosition += 30;
97          }
98
99          for ( int i = 0; i < students; i++ ) {
100            xPosition = 25;
101            yPosition += 15;
102            g.drawString( "grades[" + i + "]",
103                          xPosition, yPosition );
104            xPosition = 80;
105
106            for ( int j = 0; j < exams; j++ )  {
107               g.drawString( String.valueOf( grades[ i ][ j ] ),
108                             xPosition, yPosition );
109               xPosition += 30;
```

Fig. 5.18 Example of using double-subscripted arrays (part 2 of 3).

```
110            }
111         }
112      }
113   }
```

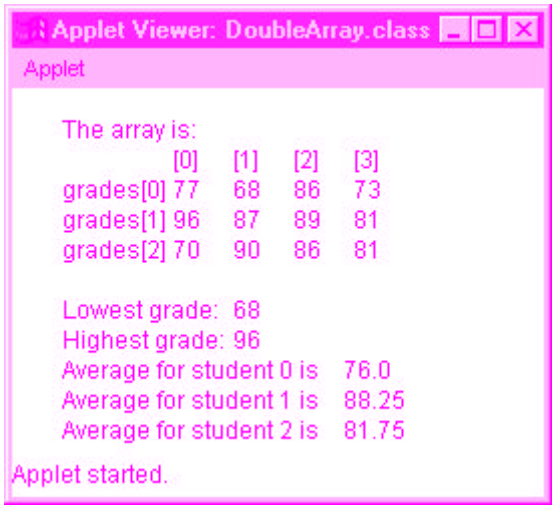


Fig. 5.18 Example of using double-subscripted arrays (part 3 of 3).