CS 335 JAVA-ARRAY

OBJECTIVES

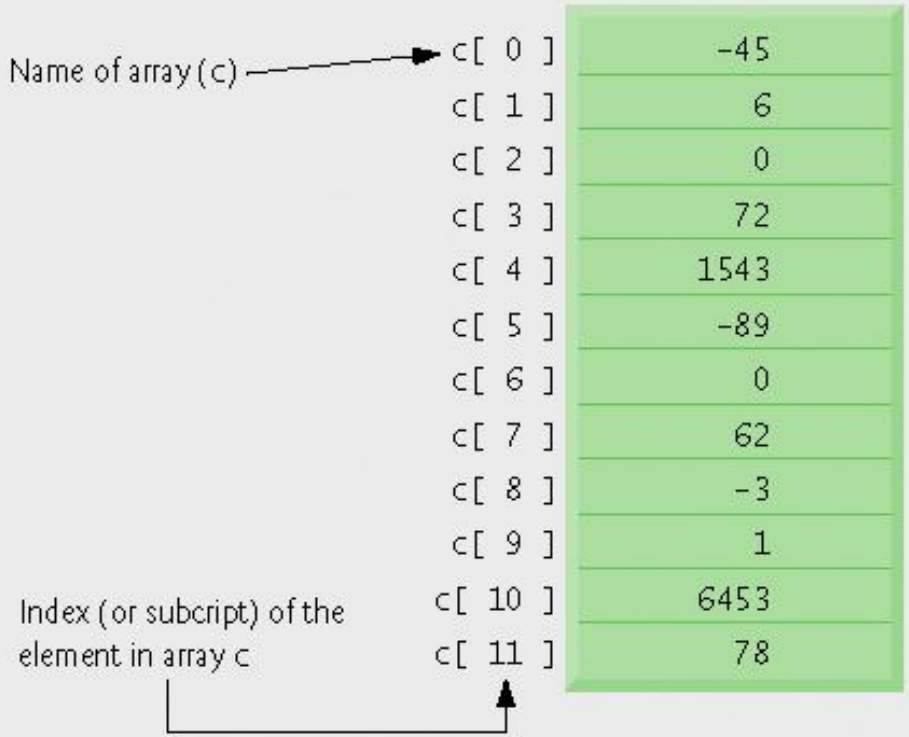
In this chapter you will learn:

- 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.

Introduction

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

12-element array.



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

- Using a value of type `long`
- Must be `int` (or)

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

- `int c[ 12 ];` → a syntax error.

Common Programming Error /Tricky Interview Question

- `int[] a, b, c;`
- `int a[], b, c;`

Examples Using Arrays

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

7.4 Examples Using Arrays

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

```java
// Fig. 7.2: InitArray.java
// Creating an array.

public class InitArray
{
   public static void main( String args[] )
   {
      int array[]; // declare array named array

      array = new int[ 10 ]; // create the space for array

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // 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
```

Examples Using Arrays

- 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`

```java
// Fig. 7.3: InitArray.java
// Initializing the elements of an array with an array initializer.

public class InitArray
{
   public static void main( String args[] )
   {
      // initializer list specifies the value for each element
      int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

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

Error-Prevention Tip

- When a program attempts to access an element outside the array bounds, an `ArrayIndexOutOfBoundsException` occurs. Exception handling is discussed later

Error-Prevention Tip

- When writing code to loop through an array, ensure that the array index is always:
  - >= 0 and < array.length

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

- 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

```java
// Fig. 7.10: DeckOfCards.java
// DeckOfCards class represents a deck of playing cards.
import java.util.Random;

public class DeckOfCards
{
   private Card deck[]; // array of Card objects
   private int currentCard; // index of next Card to be dealt
   private final int NUMBER_OF_CARDS = 52; // constant number of Cards
   private Random randomNumbers; // random number generator

   // constructor fills deck of Cards
   public DeckOfCards()
   {
      String faces[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
         "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
      String suits[] = { "Hearts", "Diamonds", "Clubs", "Spades" };

      deck = new Card[ NUMBER_OF_CARDS ]; // create array of Card objects
      currentCard = 0; // set currentCard so first Card dealt is deck[ 0 ]
      randomNumbers = new Random(); // create random number

      // populate deck with Card objects
      for ( int count = 0; count < deck.length; count++ )
         deck[ count ] =
            new Card( faces[ count % 13 ], suits[ count / 13 ] );
   } // end DeckOfCards constructor
```

Constant NUMBER_OF_CARDS indicates the number of Cards in the deck

Fill the deck array with Cards

```
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

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
```


Outline

- DeckOfCardsTest.java
- (1 of 2)

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

```
1  // Fig. 7.12: EnhancedForTest.java
2  // Using enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main( String args[] )
7     {
8        int array[] = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for ( int number : array )
13          total += number;
14
15       System.out.printf( "Total of array elements: %d\n", total );
16    } // end main
17 } // end class EnhancedForTest
```

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

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

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

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(int array [] );
```

    - Passes array `hourlyTemperatures` to method `modifyArray`

Passing Arrays to Methods

- 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

- Passing arrays by reference
  - No copy, fast

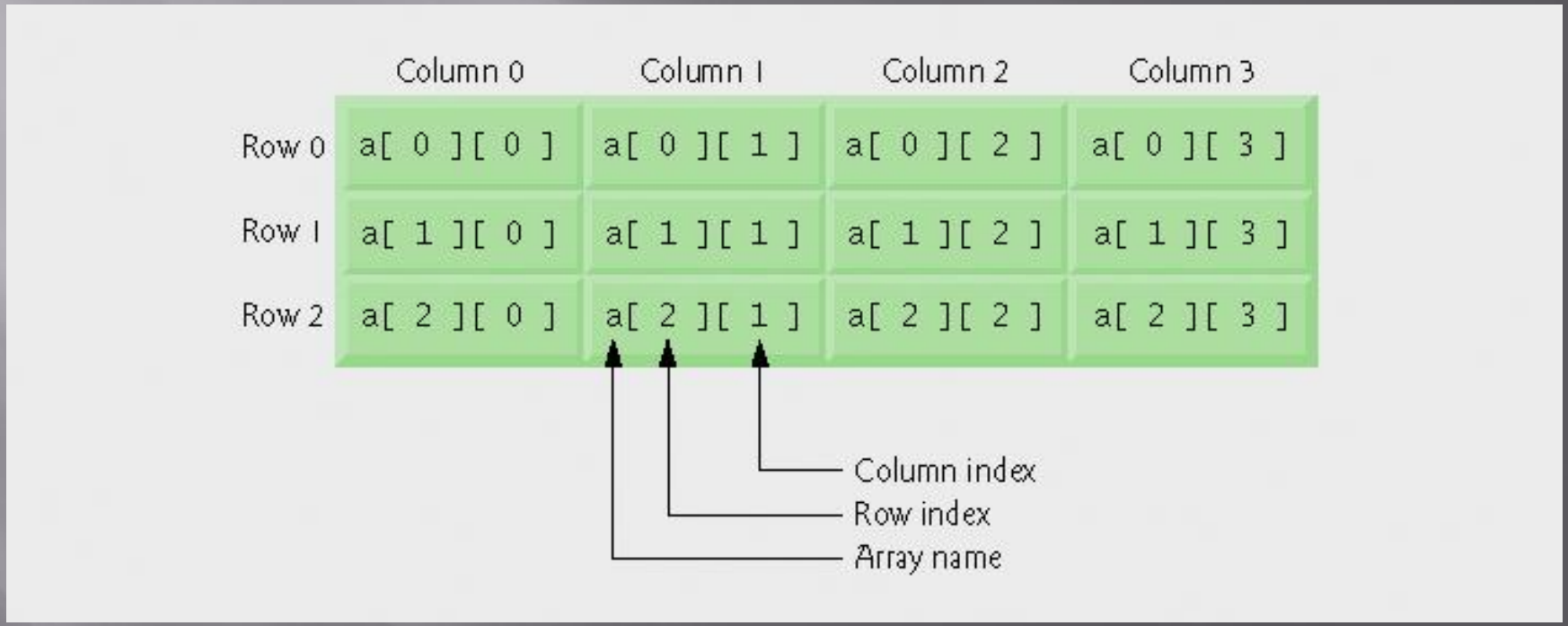
Case Study: Class GradeBook Using an Array to Store Grades

- Check the example in your textbook

Multidimensional Arrays

- Multidimensional arrays
  - Tables with rows and columns
    - Two-dimensional array
    - m-by-n array

Two-dimensional array with three rows and four columns.



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

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 } };`

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

```
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

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```

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;
```

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

Programming

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
25    } // end main
26 } // end class GradeBookTest
```

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

Implement

- Print out the following to the console

```
            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
    Avg        ???     ???     ???
```

Variable-Length Argument Lists

- Variable-length argument lists
  - Unspecified number of arguments
  - Use ellipsis (...) in method's parameter list
    - Can occur only once in parameter list
    - Must be placed at the end of parameter list
  - Array whose elements are all of the same type

Outline

```
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 enhance
12       for ( double d : numbers )
13          total += d;
14
15       return total / numbers.length;
16    } // end method average
17
18    public static void main( String args[]
19    {
20       double d1 = 10.0;
21       double d2 = 20.0;
22       double d3 = 30.0;
23       double d4 = 40.0;
24
```

Method `average` receives a variable length sequence of `double`s

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

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 d3 is %.1f\n",
31          average( d1, d2, d3 ) );
32       System.out.printf( "Average of d1, d2, d3 and d4 is %.1f\n",
33          average( d1, d2, d3, d4 ) );
34    } // end main
35 } // end class VarargsTest
```

```
d1 = 10.0
d2 = 20.0
d3 = 30.0
d4 = 40.0

Average of d1 and d2 is 15.0
Average of d1, d2 and d3 is 20.0
Average of d1, d2, d3 and d4 is 25.0
```

Common Programming Error

- An ellipsis may be placed only at the end of the parameter list.
  - in the middle: a syntax error.

Using Command-Line Arguments

- Command-line arguments
  - Pass arguments from the command line
    - `String args[]`
  - Appear after the class name in the `java` command
    - `java MyClass a b`
  - Number of arguments passed in from command line
    - `args.length`
  - First command-line argument
    - `args[ 0 ]`

Outline

```
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 the entire command, including\n" +
12             "an array size, initial value and increment." );
13       else
14       {
15          // get array size from first command-line argument
16          int arrayLength = Integer.parseInt( args[ 0 ] );
17          int array[] = new int[ arrayLength ]; // create array
18
19          // get initial value and increment from command-line argument
20          int initialValue = Integer.parseInt( args[ 1 ] );
21          int increment = Integer.parseInt( args[ 2 ] );
22
23          // calculate value for each array element
24          for ( int counter = 0; counter < array.length; counter++ )
25             array[ counter ] = initialValue + increment * counter;
26
27          System.out.printf( "%s%8s\n", "Index", "Value" );
28
```

Outline

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

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

```
java InitArray 5 0 4
Index   Value
    0       0
    1       4
    2       8
    3      12
    4      16
```

```
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
```

GUI and Graphics Case Study: Drawing Arcs

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

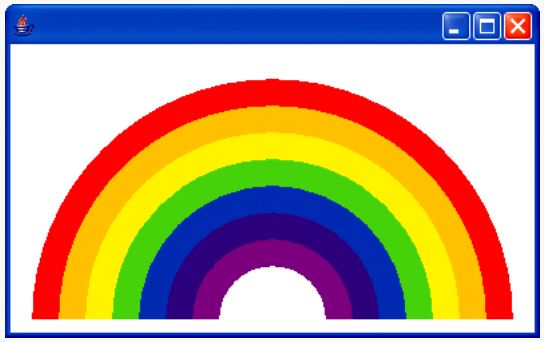
DrawRainbow

```
1  // Fig. 7.22: DrawRainbow.java
2  // Demonstrates using colors in an array.
3  import java.awt.Color;
4  import java.awt.Graphics;
5  import javax.swing.JPanel;
6
7  public class DrawRainbow extends JPanel
8  {
9     // Define indigo and violet
10    final Color VIOLET = new Color( 128, 0, 128 );
11    final Color INDIGO = new Color( 75, 0, 130 );
12
13    // colors to use in the rainbow, starting from the innermost
14    // The two white entries result in an empty arc in the center
15    private Color colors[] =
16       { Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE,
17         Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED };
18
19    // constructor
20    public DrawRainbow()
21    {
22       setBackground( Color.WHITE ); // set the background to white
23    } // end DrawRainbow constructor
24
25    // draws a rainbow using concentric circles
26    public void paintComponent( Graphics g )
27    {
28       super.paintComponent( g );
29
30       int radius = 20; // radius of an arch
```

Outline

```
31
32       // draw the rainbow near the bottom-center
33       int centerX = getWidth() / 2;
34       int centerY = getHeight() - 10;
35
36       // draws filled arcs starting with the outermost
37       for ( int counter = colors.length; counter > 0; counter-- )
38       {
39          // set the color for the current arc
40          g.setColor( colors[ counter - 1 ] );
41
42          // fill the arc from 0 to 180 degrees
43          g.fillArc( centerX - counter * radius,
44             centerY - counter * radius,
45             counter * radius * 2, counter * radius * 2, 0, 180 );
46       } // end for
47    } // end method paintComponent
48 } // end class DrawRainbow
```

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



Drawing a spiral using `drawLine` (left) and `drawArc` (right).