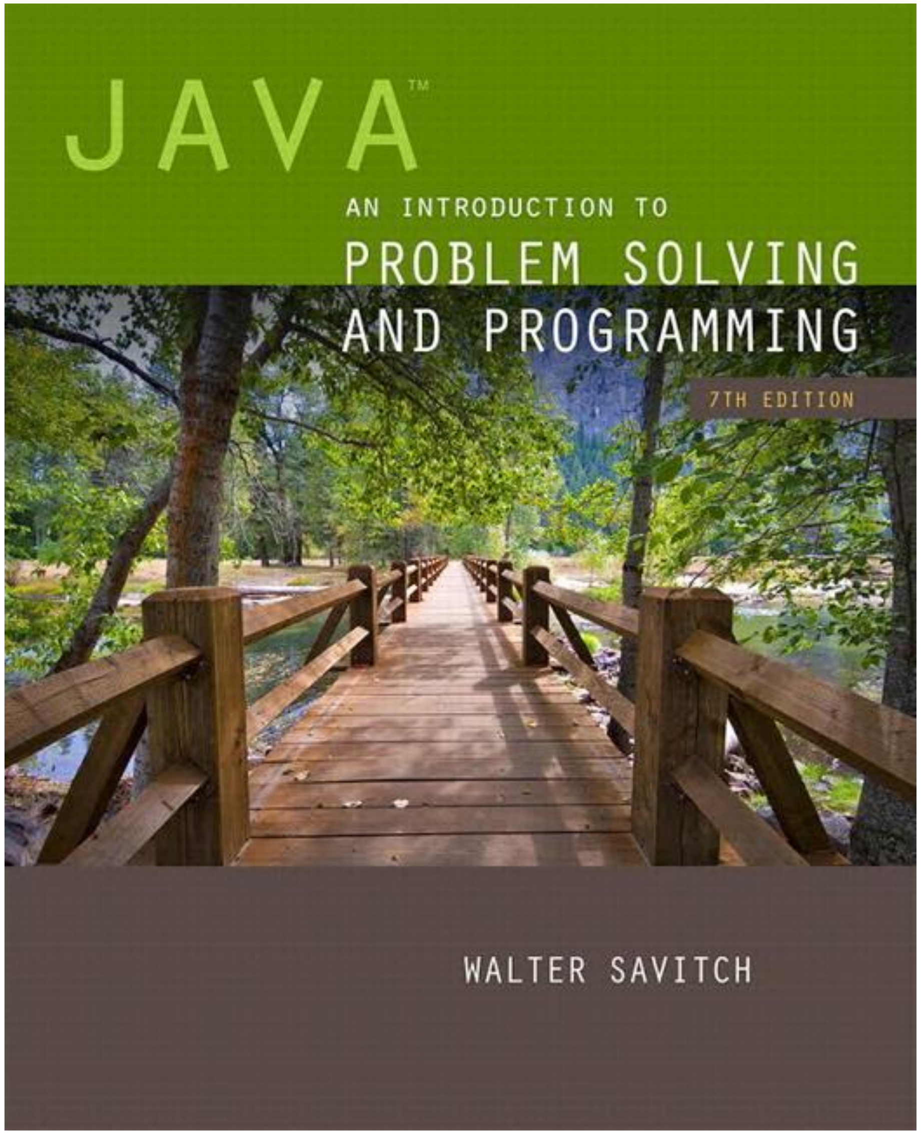
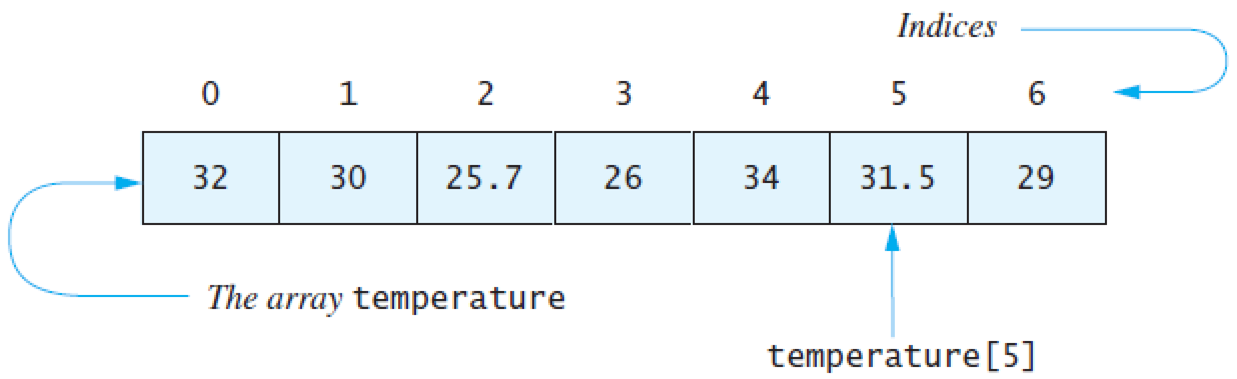

JAVA™
AN INTRODUCTION TO
PROBLEM SOLVING
AND PROGRAMMING
7TH EDITION
WALTER SAVITCH

Arrays 7

FIGURE 7.1 A Common Way to Visualize an Array



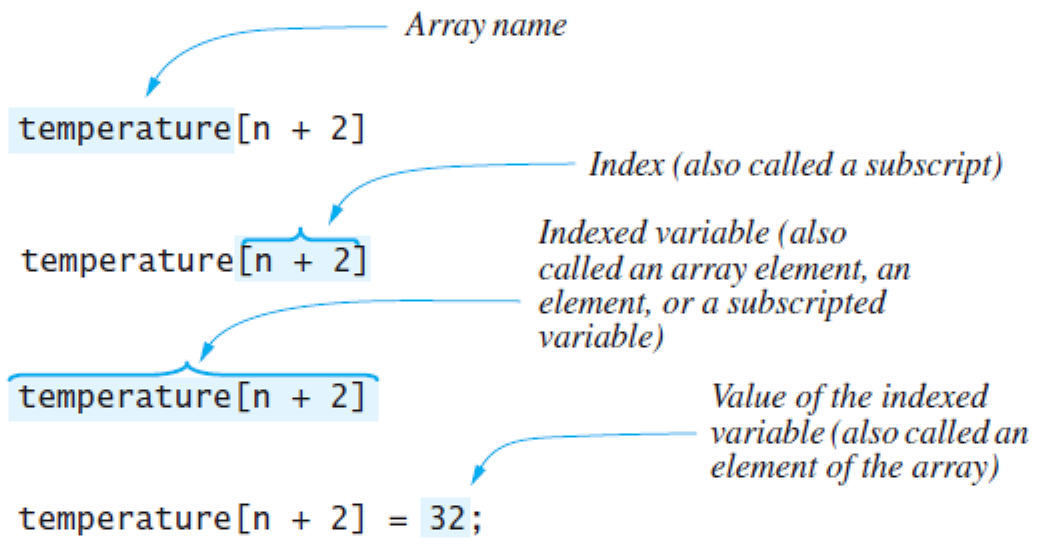
LISTING 7.1 An Array of Temperatures *(part 1 of 2)*

```
/**
Reads 7 temperatures from the user and shows which are above
and which are below the average of the 7 temperatures.
*/
import java.util.Scanner;

public class ArrayOfTemperatures
{
    public static void main(String[] args)
    {
        double[] temperature = new double[7];

        // Read temperatures and compute their average:
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter 7 temperatures:");
        double sum = 0;
        for (int index = 0; index < 7; index++)
        {
            temperature[index] = keyboard.nextDouble();
            sum = sum + temperature[index];
        }
        double average = sum / 7;
        System.out.println("The average temperature is " +
                                average);

        // Display each temperature and its relation to the average:
        System.out.println("The temperatures are");
        for (int index = 0; index < 7; index++)
        {
            if (temperature[index] < average)
                System.out.println(temperature[index] +
                                   " below average");
            else if (temperature[index] > average)
                System.out.println(temperature[index] +
                                   " above average");
            else //temperature[index] == average
                System.out.println(temperature[index] +
                                   " the average");
        }

        System.out.println("Have a nice week.");
    }
}
```

Sample Screen Output

```
Enter 7 temperatures:
32
30
25.7
26
34
31.5
29
The average temperature is 29.7428
The temperatures are
32.0 above average
30.0 above average
25.7 below average
26.0 below average
34.0 above average
31.5 above average
29.0 below average
Have a nice week.
```

FIGURE 7.2 Array Terminology


Array name
temperature[n + 2]
Index (also called a subscript)
temperature[n + 2]
Indexed variable (also called an array element, an element, or a subscripted variable)
temperature[n + 2]
Value of the indexed variable (also called an element of the array)
temperature[n + 2] = 32;

LISTING 7.2 An Array of Temperatures—Revised *(part 1 of 2)*

```
/**
Reads temperatures from the user and shows which are above
and which are below the average of all the temperatures.
*/
import java.util.Scanner;

public class ArrayOfTemperatures2
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("How many temperatures do you have?");
        int size = keyboard.nextInt( );
        double[] temperature = new double[size];

        // Read temperatures and compute their average:
        System.out.println("Enter " + temperature.length +
                           " temperatures:");
        double sum = 0;
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " +
                            average);
        // Display each temperature and its relation to the
        // average:
        System.out.println("The temperatures are");
        for (int index = 0; index < temperature.length; index++)
        {
            if (temperature[index] < average)
                System.out.println(temperature[index] +
                                   " below average");
            else if (temperature[index] > average)
                System.out.println(temperature[index] +
                                   " above average");
            else //temperature[index] == average
                System.out.println(temperature[index] +
                                   " the average");
        }

        System.out.println("Have a nice week.");
    }
}
```

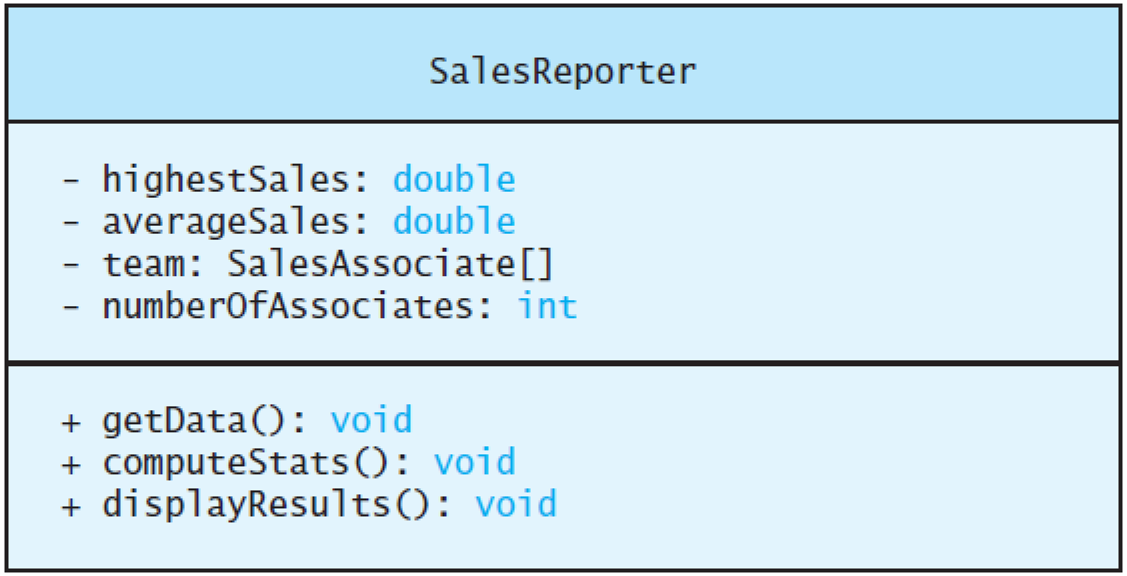
Sample Screen Output

```
How many temperatures do you have?
3
Enter 3 temperatures:
32
26.5
27
The average temperature is 28.5
The temperatures are
32.0 above average
26.5 below average
27.0 below average
Have a nice week.
```

LISTING 7.3 Sales Associate Class

```
import java.util.Scanner;
/**
 Class for sales associate records.
*/
public class SalesAssociate
{
    private String name;
    private double sales;

    public SalesAssociate()
    {
        name = "No record";
        sales = 0;
    }

    public SalesAssociate(String initialName, double initialSales)
    {
        set(initialName, initialSales);
    }

    public void set(String newName, double newSales)
    {
        name = newName;
        sales = newSales;
    }
    public void readInput()
    {
        System.out.print("Enter name of sales associate: ");
        Scanner keyboard = new Scanner(System.in);
        name = keyboard.nextLine();

        System.out.print("Enter associate's sales: $");
        sales = keyboard.nextDouble();
    }

    public void writeOutput()
    {
        System.out.println("Name: " + name);
        System.out.println("Sales: $" + sales);
    }

    public String getName()
    {
        return name;
    }

    public double getSales()
    {
        return sales;
    }
}
```

FIGURE 7.3 **Class Diagram for the Class** `Sales Reporter`



LISTING 7.4 A Sales Report Program *(part 1 of 3)*

```
import java.util.Scanner;
/**
 Program to generate sales report.
*/
public class SalesReporter
{
    private double highestSales;
    private double averageSales;
    private SalesAssociate[] team;  //The array object is
                                    //created in getData.
    private int numberOfAssociates; //Same as team.length
    /**
     Reads the number of sales associates and data for each one.
    */
    public void getData()
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter number of sales associates:");
        numberOfAssociates = keyboard.nextInt();
        team = new SalesAssociate[numberOfAssociates + 1];
        for (int i = 1; i <= numberOfAssociates; i++)
        {
            team[i] = new SalesAssociate();
            System.out.println("Enter data for associate " + i);
            team[i].readInput();
            System.out.println();
        }
    }
```

The `main` method is at the end of the class.

Array object created here.

`SalesAssociate` objects created here.

```java
/**
 Computes the average and highest sales figures.
 Precondition: There is at least one salesAssociate.
*/
public void computeStats()
{
    double nextSales = team[1].getSales();
    highestSales = nextSales;
    double sum = nextSales;
    for (int i = 2; i <= numberOfAssociates; i++)
    {
        nextSales = team[i].getSales();
        sum = sum + nextSales;
        if (nextSales > highestSales)
            highestSales = nextSales; //highest sales so far.
    }
    averageSales = sum / numberOfAssociates;
}
```

Already processed `team[1]`*, so the loop starts with* `team[2]`*.*

```
/**
 Displays sales report on the screen.
*/
public void displayResults()
{
    System.out.println("Average sales per associate is $" +
                       averageSales);
    System.out.println("The highest sales figure is $" +
                       highestSales);
    System.out.println();
    System.out.println("The following had the highest sales:");
    for (int i = 1; i <= numberOfAssociates; i++)
    {
        double nextSales = team[i].getSales();
        if (nextSales == highestSales)
        {
            team[i].writeOutput();
            System.out.println("$" + (nextSales - averageSales)
                               + " above the average.");
            System.out.println();
        }
    }
```

```
        System.out.println("The rest performed as follows:");
        for (int i = 1; i <= numberOfAssociates; i++)
        {
            double nextSales = team[i].getSales();
            if (team[i].getSales() != highestSales)
            {
                team[i].writeOutput();
                if (nextSales >= averageSales)
                    System.out.println("$" + (nextSales -
                           averageSales) + " above the average.");
                else
                    System.out.println("$" + (averageSales -
                              nextSales) + " below the average.");
                System.out.println();
            }
        }
    }
    public static void main(String[] args)
    {
        SalesReporter clerk = new SalesReporter();
        clerk.getData();
        clerk.computeStats();
        clerk.displayResults();
    }
}
```

Sample Screen Output

```
Enter number of sales associates:
3
Enter data for associate number 1
Enter name of sales associate: Dusty Rhodes
Enter associate's sales: $36000
Enter data for associate number 2
Enter name of sales associate: Natalie Dressed
Enter associate's sales: $50000
Enter data for associate number 3
Enter name of sales associate: Sandy Hair
Enter associate's sales: $10000
Average sales per associate is $32000.0
The highest sales figure is $50000.0
The following had the highest sales:
Name: Natalie Dressed
Sales: $50000.0
$18000.0 above the average.
The rest performed as follows:
Name: Dusty Rhodes
Sales: $36000.0
$4000.0 above the average.
Name: Sandy Hair
Sales: $10000.0
$22000.0 below the average.
```

LISTING 7.5 Indexed Variables as Arguments

```java
import java.util.Scanner;

/**
 A demonstration of using indexed variables as arguments.
*/
public class ArgumentDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt();
        int[] nextScore = new int[3];

        for (int i = 0; i < nextScore.length; i++)
            nextScore[i] = firstScore + 5 * i;

        for (int i = 0; i < nextScore.length; i++)
        {
            double possibleAverage =
                            getAverage(firstScore, nextScore[i]);
            System.out.println("If your score on exam 2 is " +
                                nextScore[i]);
            System.out.println("your average will be " +
                                possibleAverage);
        }
    }

    public static double getAverage(int n1, int n2)
    {
        return (n1 + n2) / 2.0;
    }
}
```

Sample Screen Output

```
Enter your score on exam 1:
80
If your score on exam 2 is 80
your average will be 80.0
If your score on exam 2 is 85
your average will be 82.5
If your score on exam 2 is 90
your average will be 85.0
```

LISTING 7.6 Two Kinds of Equality *(part 1 of 2)*

```
/**
 A demonstration program to test two arrays for equality.
*/
public class TestEquals
{
    public static void main(String[] args)
    {
        int[] a = new int[3];
        int[] b = new int[3];
        setArray(a);
        setArray(b);

        if (b == a)
            System.out.println("Equal by ==.");
        else
            System.out.println("Not equal by ==.");

        if (equals(b, a))
            System.out.println("Equal by the equals method.");
        else
            System.out.println("Not equal by the equals method.");
    }
```

The arrays a *and* b *contain the same integers in the same order.*

```
    public static boolean equals(int[] a, int[] b)
    {
        boolean elementsMatch = true;//tentatively
        if (a.length != b.length)
            elementsMatch = false;
        else
        {
            int i = 0;
            while (elementsMatch && (i < a.length))
            {
                if (a[i] != b[i])
                    elementsMatch = false;
                i++;
            }
         }
         return elementsMatch;
    }

    public static void setArray(int[] array)
    {
         for (int i = 0; i < array.length; i++)
              array[i] = i;
    }
}
```

Screen Output

```
Not equal by ==.
Equal by the equals method.
```

LISTING 7.7 A Method That Returns an Array

```java
import java.util.Scanner;
/**
 A demonstration of a method that returns an array.
*/
public class ReturnArrayDemo
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt();
        int[] nextScore = new int[3];

        for (int i = 0; i < nextScore.length; i++)
            nextScore[i] = firstScore + 5 * i;

        double[] averageScore =
                 getArrayOfAverages(firstScore, nextScore);
        for (int i = 0; i < nextScore.length; i++)
        {
            System.out.println("If your score on exam 2 is " +
                               nextScore[i]);
            System.out.println("your average will be " +
                               averageScore[i]);
        }
    }

    public static double[] getArrayOfAverages(int firstScore,
                                              int[] nextScore)
    {
        double[] temp = new double[nextScore.length];
        for (int i = 0; i < temp.length; i++)
            temp[i] = getAverage(firstScore, nextScore[i]);

        return temp;
    }

    public static double getAverage(int n1, int n2)
    {
        return (n1 + n2) / 2.0;
    }
}
```

The sample screen output is the same as in Listing 7.5.

LISTING 7.8 Using the Class `OneWayNoRepeatsList` *(part 1 of 2)*

```
import java.util.Scanner;

public class ListDemo
{
    public static final int MAX_SIZE = 3; //Assumed > 0

    public static void main(String[] args)
    {
        OneWayNoRepeatsList toDoList =
                                new OneWayNoRepeatsList(MAX_SIZE);
        System.out.println(
                    "Enter items for the list, when prompted.");
        boolean moreEntries = true;
        String next = null;
        Scanner keyboard = new Scanner(System.in);

        while (moreEntries && !toDoList.isFull())
        {
            System.out.println("Enter an item:");
            next = keyboard.nextLine();
            toDoList.addItem(next);

            if (toDoList.isFull())
            {
                System.out.println("List is now full.");
            }
            else
            {
                System.out.print("More items for the list? ");
                String ans = keyboard.nextLine();
                if (ans.trim().equalsIgnoreCase("no"))
                    moreEntries = false; //User says no more
            }
        }
```

```
            System.out.println("The list contains:");
            int position = toDoList.START_POSITION;
            next = toDoList.getEntryAt(position);
            while (next != null) //null indicates end of list
            {
                System.out.println(next);
                position++;
                next = toDoList.getEntryAt(position);
            }
        }
    }
```

Sample Screen Output

```
Enter items for the list, when prompted.
Enter an item:
Buy milk
More items for the list? yes
Enter an item:
Walk dog
More items for the list? yes
Enter an item:
Buy milk
More items for the list? yes
Enter an item:
Write program
The list is now full.
The list contains:
Buy milk
Walk dog
Write program
```

LISTING 7.9 An Array Wrapped in a Class to Represent a List *(part 1 of 3)*

```java
/**
 An object of this class is a special kind of list of strings.
 You can write the list only from beginning to end. You can add
 only to the end of the list. You cannot change individual en-
 tries, but you can erase the entire list and start over. No
 entry may appear more than once on the list. You can use int
 variables as position markers into the list. Position markers
 are similar to array indices, but are numbered starting with 1.
*/
public class OneWayNoRepeatsList
{
    public static int START_POSITION = 1;
    public static int DEFAULT_SIZE = 50;

    //entry.length is the total number of items you have room
    //for on the list (its capacity); countOfEntries is the number of
    //items currently on the list.
    private int countOfEntries; //can be less than entry.length.
    private String[] entry;

    /**
     Creates an empty list with a given capacity.
    */
    public OneWayNoRepeatsList(int maximumNumberOfEntries)
    {
        entry = new String[maximumNumberOfEntries];
        countOfEntries = 0;
    }
```

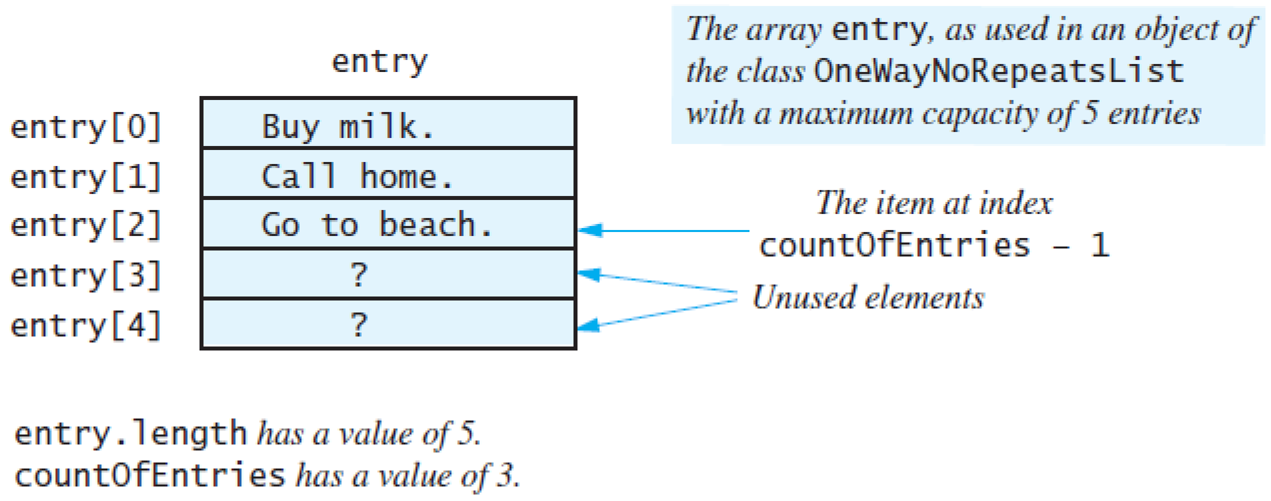
```
/**
 Creates an empty list with a capacity of DEFAULT_SIZE.
*/
public OneWayNoRepeatsList()
{
    entry = new String[DEFAULT_SIZE];
    countOfEntries = 0;
// or replace these two statements with this(DEFAULT_SIZE);
}

public boolean isFull()
{
    return countOfEntries == entry.length;
}

public boolean isEmpty()
{
    return countOfEntries == 0;
}
```

```java
/**
 Precondition: List is not full.
 Postcondition: If item was not on the list,
 it has been added to the list.
*/
public void addItem(String item)
{
    if (!isOnList(item))
    {
        if (countOfEntries == entry.length)
        {
            System.out.println("Adding to a full list!");
            System.exit(0);
        }
        else
        {
            entry[countOfEntries] = item;
            countOfEntries++;
        }
    } //else do nothing. Item is already on the list.
}
```

```java
/**
 If the argument indicates a position on the list,
 the entry at that specified position is returned;
 otherwise, null is returned.
*/
public String getEntryAt(int position)
{
    String result = null;
    if ((1 <= position) && (position <= countOfEntries))
        result = entry[position - 1];

    return result;
}

/**
 Returns true if position indicates the last item
 on the list; otherwise, returns false.
*/
public boolean atLastEntry(int position)
{
    return position == countOfEntries;
}
```

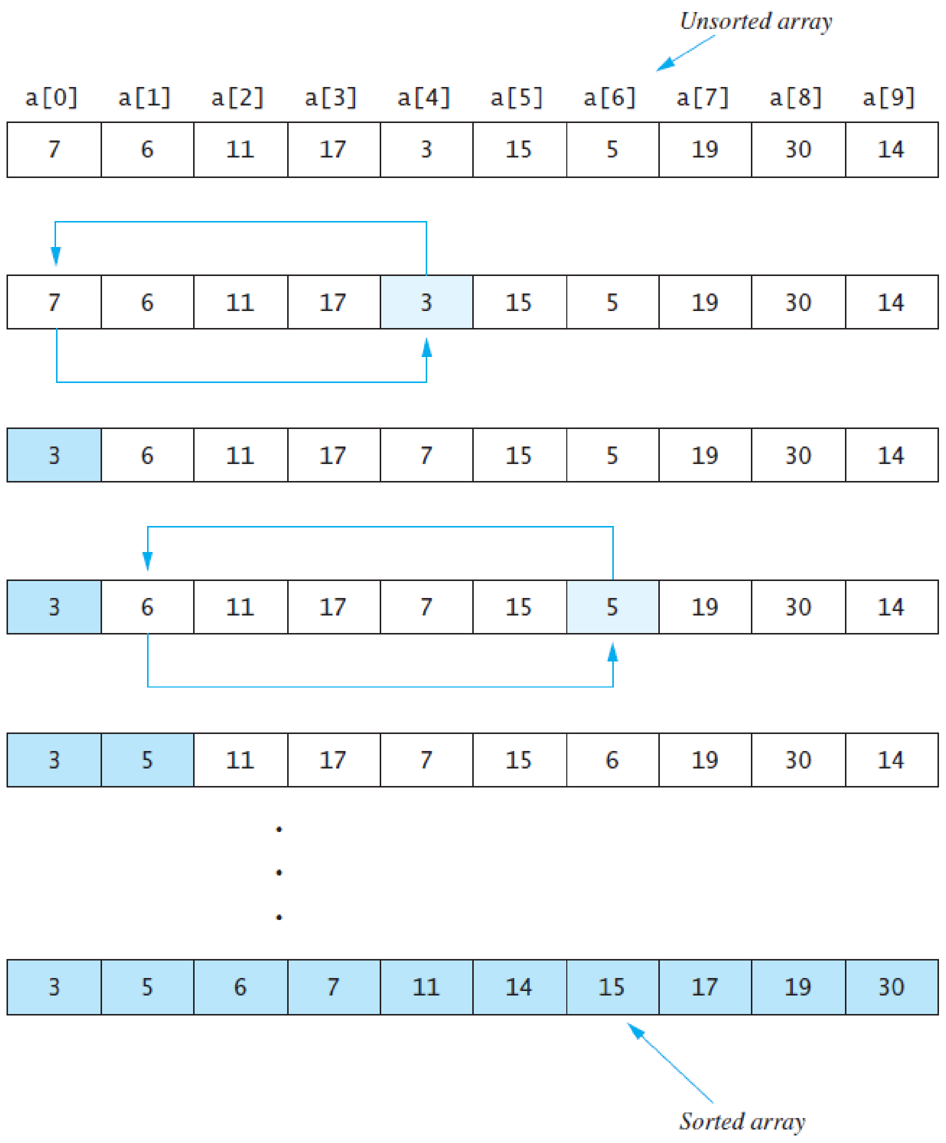
```
    /**
     Returns true if item is on the list;
     otherwise, returns false. Does not differentiate
     between uppercase and lowercase letters.
    */
    public boolean isOnList(String item)
    {
        boolean found = false;
        int i = 0;
        while (!found && (i < countOfEntries))
        {
            if (item.equalsIgnoreCase(entry[i]))
                found = true;
            else
                i++;
        }

        return found;
    }

    public int getMaximumNumberOfEntries()
    {
        return entry.length;
    }

    public int getNumberOfEntries()
    {
        return countOfEntries;
    }

    public void eraseList()
    {
        countOfEntries = 0;
    }
}
```

FIGURE 7.4 A Partially Filled Array



entry
entry[0]
entry[1]
entry[2]
entry[3]
entry[4]
Buy milk.
Call home.
Go to beach.
?
?
The array entry, as used in an object of the class OneWayNoRepeatsList with a maximum capacity of 5 entries
The item at index countOfEntries - 1
Unused elements
entry.length has a value of 5.
countOfEntries has a value of 3.

FIGURE 7.5 **Selection Sort**



LISTING 7.10 Implementation of the Selection Sort *(part 1 of 2)*

```
/**
 Class for sorting an array of base type int from smallest to largest.
*/
public class ArraySorter
{
    /**
     Precondition: Every element in anArray has a value.
     Action: Sorts the array into ascending order.
    */
    public static void selectionSort(int[] anArray)
    {
        for (int index = 0; index < anArray.length - 1; index++)
        {   // Place the correct value in anArray[index]
            int indexOfNextSmallest = getIndexOfSmallest(index, anArray);
            interchange(index, indexOfNextSmallest, anArray);
            //Assertion:anArray[0] <= anArray[1] <=...<= anArray[index]
            //and these are the smallest of the original array elements.
            //The remaining positions contain the rest of the original
            //array elements.
        }
    }
```

```java
/**
 Returns the index of the smallest value in the portion of the
 array that begins at the element whose index is startIndex and
 ends at the last element.
*/
private static int getIndexOfSmallest(int startIndex, int[] a)
{
    int min = a[startIndex];
    int indexOfMin = startIndex;
    for (int index = startIndex + 1; index < a.length; index++)
    {
        if (a[index] < min)
        {
            min = a[index];
            indexOfMin = index;
            //min is smallest of a[startIndex] through a[index]
        }
    }
    return indexOfMin;
}
```

```
    /**
     Precondition: i and j are valid indices for the array a.
     Postcondition: Values of a[i] and a[j] have been interchanged.
    */
    private static void interchange(int i, int j, int[] a)
    {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp; //original value of a[i]
    }
}
```

LISTING 7.11 Demonstration of the Method selectionSort

```java
public class SelectionSortDemo
{
    public static void main(String[] args)
    {
        int[] b = {7, 5, 11, 2, 16, 4, 18, 14, 12, 30};

        display(b, "before");
        ArraySorter.selectionSort(b);
        display(b, "after");
    }

    public static void display(int[] array, String when)
    {
        System.out.println("Array values " + when + " sorting:");
        for (int i = 0; i < array.length; i++)
            System.out.print(array[i] + " ");
        System.out.println( );
    }
}
```

Screen Output

```
Array values before sorting:
7 5 11 2 16 4 18 14 12 30
Array values after sorting:
2 4 5 7 11 12 14 16 18 30
```

FIGURE 7.6 A Table of Values

Savings Account Balances for Various Interest Rates Compounded Annually (Rounded to Whole Dollar Amounts)						
Year	5.00%	5.50%	6.00%	6.50%	7.00%	7.50%
1	$1050	$1055	$1060	$1065	$1070	$1075
2	$1103	$1113	$1124	$1134	$1145	$1156
3	$1158	$1174	$1191	$1208	$1225	$1242
4	$1216	$1239	$1262	$1286	$1311	$1335
5	$1276	$1307	$1338	$1370	$1403	$1436
6	$1340	$1379	$1419	$1459	$1501	$1543
7	$1407	$1455	$1504	$1554	$1606	$1659
8	$1477	$1535	$1594	$1655	$1718	$1783
9	$1551	$1619	$1689	$1763	$1838	$1917
10	$1629	$1708	$1791	$1877	$1967	$2061

FIGURE 7.7 Row and Column Indices for an Array Named `table`

Row index 3

Column index 2

Indices	0	1	2	3	4	5
0	$1050	$1055	$1060	$1065	$1070	$1075
1	$1103	$1113	$1124	$1134	$1145	$1156
2	$1158	$1174	$1191	$1208	$1225	$1242
3	$1216	$1239	$1262	$1286	$1311	$1335
4	$1276	$1307	$1338	$1370	$1403	$1436
5	$1340	$1379	$1419	$1459	$1501	$1543
6	$1407	$1455	$1504	$1554	$1606	$1659
7	$1477	$1535	$1594	$1655	$1718	$1783
8	$1551	$1619	$1689	$1763	$1838	$1917
9	$1629	$1708	$1791	$1877	$1967	$2061

`table[3][2]` *has a value of* `1262`

LISTING 7.12 Using a Two-Dimensional Array *(part 1 of 2)*

A real application would do something more with the array `table`*. This is just a demonstration program.*

```
/**
 Displays a two-dimensional table showing how
 interest rates affect bank balances.
*/
public class InterestTable
{
    public static void main(String[] args)
    {
        int[][] table = new int[10][6];
        for (int row = 0; row < 10; row++)
            for (int column = 0; column < 6; column++)
                table[row][column] =
                    getBalance(1000.00, row + 1, (5 + 0.5 *
                                                   column));
        System.out.println("Balances for Various Interest Rates " +
                                  "Compounded Annually");
        System.out.println("(Rounded to Whole Dollar Amounts)");
        System.out.println();
        System.out.println("Years 5.00% 5.50% 6.00% 6.50% " +
                                  "7.00% 7.50%");
        for (int row = 0; row < 10; row++)
        {
            System.out.print((row + 1) + "  ");
            for (int column = 0; column < 6; column++)
                System.out.print("$" + table[row][column] + " ");
            System.out.println();
        }
    }
    /**
    Returns the balance in an account after a given number of years
    and interest rate with an initial balance of startBalance.
    Interest is compounded annually. The balance is rounded
    to a whole number.
    */
    public static int getBalance(double startBalance, int years,
                                 double rate)
    {
        double runningBalance = startBalance;
        for (int count = 1; count <= years; count++)
            runningBalance = runningBalance * (1 + rate / 100);
        return (int)(Math.round(runningBalance));
    }
}
```

Sample Screen Output

```
Balances for Various Interest Rates Compounded Annually
(Rounded to Whole Dollar Amounts)

Years    5.00%    5.50%    6.00%    6.50%    7.00%    7.50%
1        $1050    $1055    $1060    $1065    $1070    $1075
2        $1103    $1113    $1124    $1134    $1145    $1156
3        $1158    $1174    $1191    $1208    $1225    $1242
4        $1216    $1239    $1262    $1286    $1311    $1335
5        $1276    $1307    $1338    $1370    $1403    $1436
6        $1340    $1379    $1419    $1459    $1501    $1543
7        $1407    $1455    $1504    $1554    $1606    $1659
8        $1477    $1535    $1594    $1655    $1718    $1783
9        $1551    $1619    $1689    $1763    $1838    $1917
10        $1629    $1708    $1791    $1877    $1967    $2061
```

The last line is out of alignment because 10 has two digits. This is easy to fix, but that would clutter the discussion of arrays with extraneous concerns.

LISTING 7.13 A Multidimensional-Array Parameter

```
/**
 Displays a two-dimensional table showing how interest
 rates affect bank balances.
*/
public class InterestTable2
{
    public static final int ROWS = 10;
    public static final int COLUMNS = 6;

    public static void main(String[] args)
    {
        int[][] table = new int[ROWS][COLUMNS];
        for (int row = 0; row < ROWS; row++)
            for (int column = 0; column < COLUMNS; column++)
             table[row][column] =
                 getBalance(1000.00, row + 1, (5 + 0.5 * column));

        System.out.println("Balances for Various Interest Rates " +
                           "Compounded Annually");
        System.out.println("(Rounded to Whole Dollar Amounts)");
        System.out.println();
        System.out.println("Years 5.00% 5.50% 6.00% 6.50% 7.00% 7.50%");

        showTable(table);
    }
    /**
    Precondition: The array anArray has ROWS rows and COLUMNS columns.
    Postcondition: The array contents are displayed with dollar signs.
    */
    public static void showTable(int[][] anArray)
    {
        for (int row = 0; row < ROWS; row++)
        {
            System.out.print((row + 1) + " ");
            for (int column = 0; column < COLUMNS; column++)
                System.out.print("$" + anArray[row][column] + " ");
            System.out.println();
        }
    }

    public static int getBalance(double startBalance, int years, double rate)
    <The rest of the definition of getBalance is the same as in Listing 7.12.>
}
```

A better definition of `showTable` *is possible, as you will see.*

The output is the same as in Listing 7.12.

LISTING 7.14 A Timekeeping Program *(part 1 of 4)*

```
/**
 Class that records the time worked by each of a
 company's employees during one five-day week.
 A sample application is in the main method.
*/
public class TimeBook
{
    private int numberOfEmployees;
    private int[][] hours;   //hours[i][j] has the hours for
                             //employee j on day i.
    private int[] weekHours; //weekHours[i] has the week's
                             //hours worked for employee i + 1.
    private int[] dayHours;  //dayHours[i] has the total hours
                             //worked by all employees on day i.
    private static final int NUMBER_OF_WORKDAYS = 5;
    private static final int MON = 0;
    private static final int TUE = 1;
    private static final int WED = 2;
    private static final int THU = 3;
    private static final int FRI = 4;
```

```
/**
 Reads hours worked for each employee on each day of the
 work week into the two-dimensional array hours. (The method
 for input is just a stub in this preliminary version.)
 Computes the total weekly hours for each employee and
 the total daily hours for all employees combined.
*/
public static void main(String[] args)
{
    private static final int NUMBER_OF_EMPLOYEES = 3;
    TimeBook book = new TimeBook(NUMBER_OF_EMPLOYEES);
    book.setHours();
    book.update();
    book.showTable();
}
```

A class generally has more methods. We have defined only the methods used in `main`*.*

```
public TimeBook(int theNumberOfEmployees)
{
    numberOfEmployees = theNumberOfEmployees;
    hours = new int[NUMBER_OF_WORKDAYS][numberOfEmployees];
    weekHours = new int[numberOfEmployees];
    dayHours = new int[NUMBER_OF_WORKDAYS];
}
```

*The **final** program would replace the stub* `setHours` *with a complete method to obtain the employee data from the user.*

```
public void setHours() //This is a stub.
{
    hours[0][0] = 8; hours[0][1] = 0; hours[0][2] = 9;
    hours[1][0] = 8; hours[1][1] = 0; hours[1][2] = 9;
    hours[2][0] = 8; hours[2][1] = 8; hours[2][2] = 8;
    hours[3][0] = 8; hours[3][1] = 8; hours[3][2] = 4;
    hours[4][0] = 8; hours[4][1] = 8; hours[4][2] = 8;
}

public void update()
{
    computeWeekHours();
    computeDayHours();
}
```

```
private void computeWeekHours()
{
    for (employeeNumber = 1; employeeNumber <=
            numberOfEmployees; employeeNumber++)
    {//Process one employee:
        int sum = 0;
        for (int day = MON; day <= FRI; day++)
            sum = sum + hours[day][employeeNumber - 1];
            //sum contains the sum of all the hours worked in
            //one
            //week by the employee with number employeeNumber.
        weekHours[employeeNumber - 1] = sum;
    }
}

private void computeDayHours()
{
    for (int day = MON; day <= FRI; day++)
    {//Process one day (for all employees):
        int sum = 0;
        for (int employeeNumber = 1;
                 employeeNumber <= numberOfEmployees;
                 employeeNumber++)
            sum = sum + hours[day][employeeNumber - 1];
            //sum contains the sum of all hours worked by all
            //employees on one day.
        dayHours[day] = sum;
    }
}
```

The method **showTable** *can and should be made more robust. See Programming Project 6.*

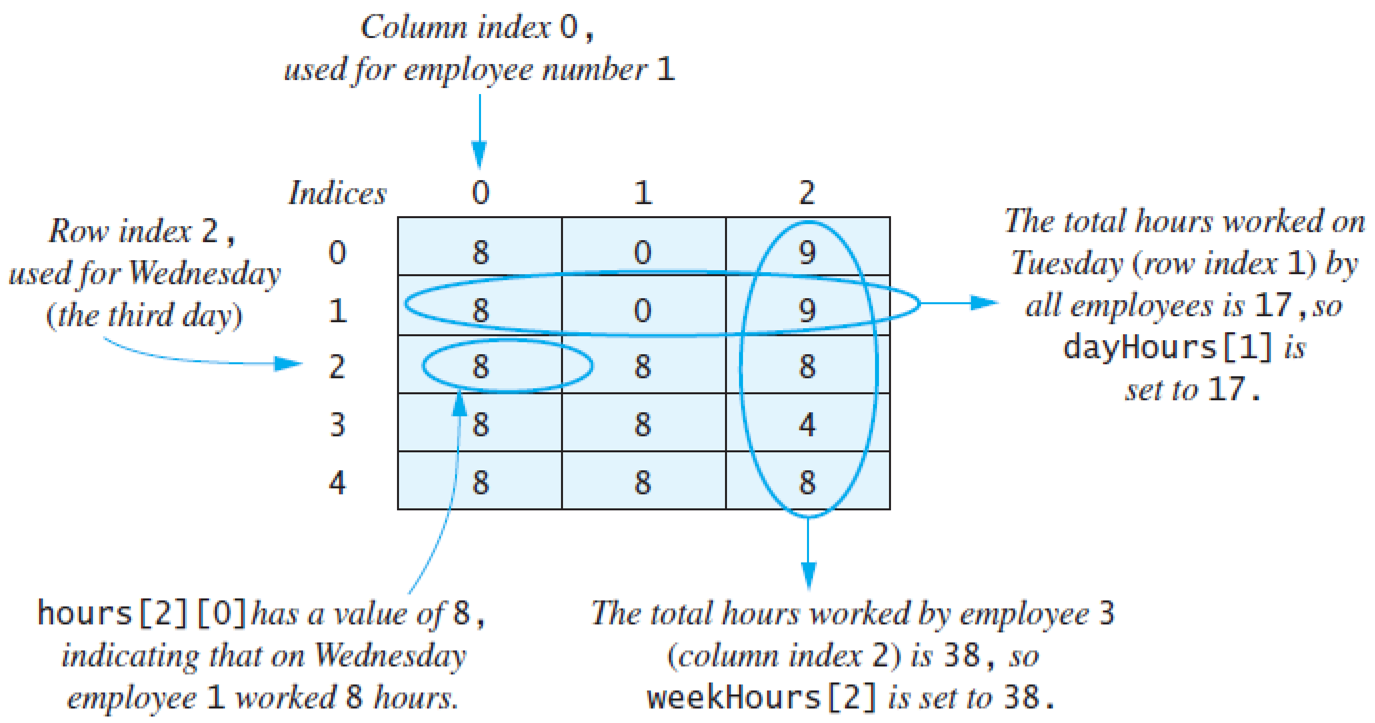
```java
public void showTable()
{
    // heading
    System.out.print("Employee ");
    for (int employeeNumber = 1;
             employeeNumber <= numberOfEmployees;
             employeeNumber++)
        System.out.print(employeeNumber + " ");
    System.out.println("Totals");
    System.out.println( );

    // row entries
    for (int day = MON; day <= FRI; day++)
    {
        System.out.print(getDayName(day) + " ");
        for (int column = 0; column < hours[day].length;
                 column++)
            System.out.print(hours[day][column] + " ");
        System.out.println(dayHours[day]);
    }
```

```
            System.out.println( );

            System.out.print("Total = ");
            for (int column = 0; column < numberOfEmployees; column++)
                System.out.print(weekHours[column] + " ");
            System.out.println( );
        }

        //Converts 0 to "Monday", 1 to "Tuesday", etc.
        //Blanks are inserted to make all strings the same length.
        private String getDayName(int day)
        {
            String dayName = null;
            switch (day)
            {
                case MON:
                    dayName = "Monday   ";
                    break;
                case TUE:
                    dayName = "Tuesday  ";
                    break;
                case WED:
                    dayName = "Wednesday";
                    break;
                case THU:
                    dayName = "Thursday ";
                    break;
                case FRI:
                    dayName = "Friday   ";
                    break;
                default:
                    System.out.println("Fatal Error.");
                    System.exit(0);
                    break;
            }
            return dayName;
        }
    }
```

Sample Screen Output

```
Employee   1   2   3   Totals
Monday     8   0   9   17
Tuesday    8   0   9   17
Wednesday  8   8   8   24
Thursday   8   8   4   20
Friday     8   8   8   24
Total  =   40  24  38
```

FIGURE 7.8 Arrays for the Class `TimeBook`


Column index `0`, used for employee number `1`
Indices
0 1 2
0 1 2 3 4
8 0 9
8 0 9
8 8 8
8 8 4
8 8 8
Row index `2`, used for Wednesday (the third day)
The total hours worked on Tuesday (row index `1`) by all employees is `17`, so `dayHours[1]` is set to `17`.
`hours[2][0]` has a value of `8`, indicating that on Wednesday employee `1` worked `8` hours.
The total hours worked by employee `3` (column index `2`) is `38`, so `weekHours[2]` is set to `38`.

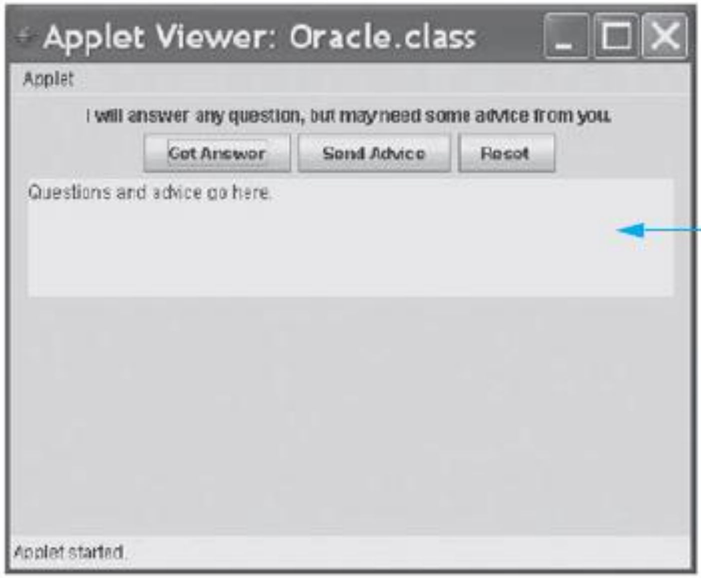
LISTING 7.15 An Applet with a Text Area *(part 1 of 2)*

```
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Oracle extends JApplet implements ActionListener
{
    public static int LINES = 5;
    public static int CHAR_PER_LINE = 40;

    private JTextArea theText;
    private String question;
    private String answer;
    private String advice;
```

```
public void int()
{
    Container contentPane = getContentPane();
    contentPane.setLayout(new FlowLayout());

    JLabel instructions=
        new JLabel("I will answer any question, " +
                   "but may need some advice from you.");
    contentPane.add(instructions);

    JButton getAnswerButton = new JButton("Get Answer");
    getAnswerButton.addActionListener(this);
    contentPane.add(getAnswerButton);

    JButton sendAdviceButton = new JButton("Send Advice");
    sendAdviceButton.addActionListener(this);
    contentPane.add(sendAdviceButton);

    JButton resetButton = new JButton("Reset");
    resetButton.addActionListener(this);
    contentPane.add(resetButton);

    theText = new JTextArea(LINES, CHAR_PER_LINE);
    theText.setText("Questions and advice go here.");
    contentPane.add(theText);
    answer = "The answer is: Look around."; //first answer
}
```

```
        public void actionPerformed(ActionEvent e)
        {
            String actionCommand = e.getActionCommand();
            if (actionCommand.equals("Get Answer"))
            {
                question = theText.getText();
                theText.setText("That is a difficult question.\n" +
                                "Please give me some advice\n" +
                                "and click the Send Advice button.");
            }
            else if (actionCommand.equals("Send Advice"))
            {
                advice = theText.getText();
                theText.setText("That advice helped.\n" +
                                "You asked the question: " + question +
                                "\n" + answer +
                                "\nClick the Reset button and" +
                                "\nleave the program on for others.");
                answer = "The answer is: " + advice;
            }
            else if (actionCommand.equals("Reset"))
            {
                theText.setText("Questions and advice go here.");
            }
            else
                theText.setText("Error");
        }
    }
```

Applet Output



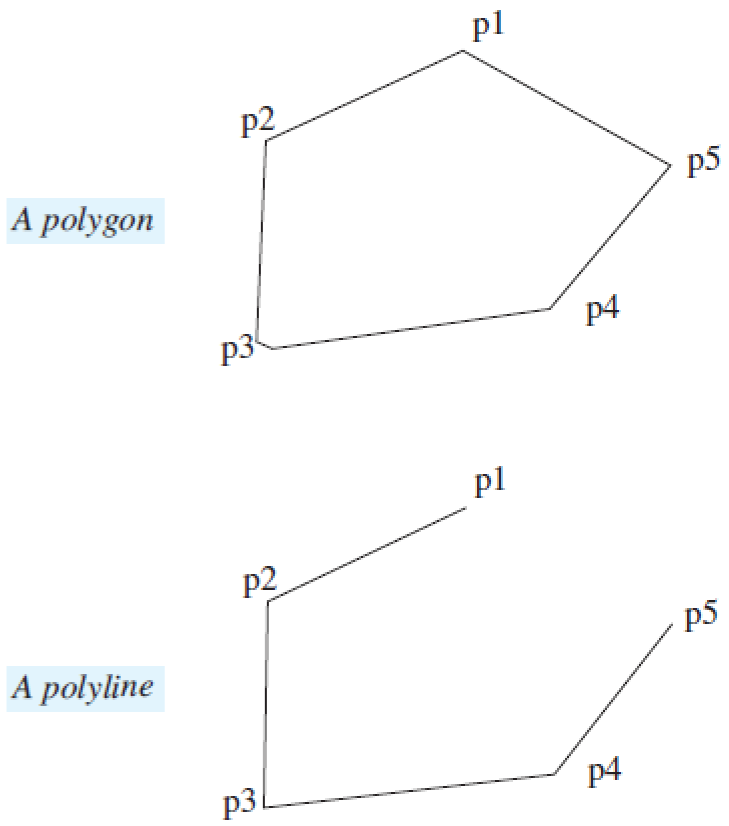
I will answer any question, but may need some advice from you.

Questions and advice go here.

This is the text area.

The positions of the buttons will vary depending on local conditions on your computer. To clean up the display, use your mouse to resize the applet window.

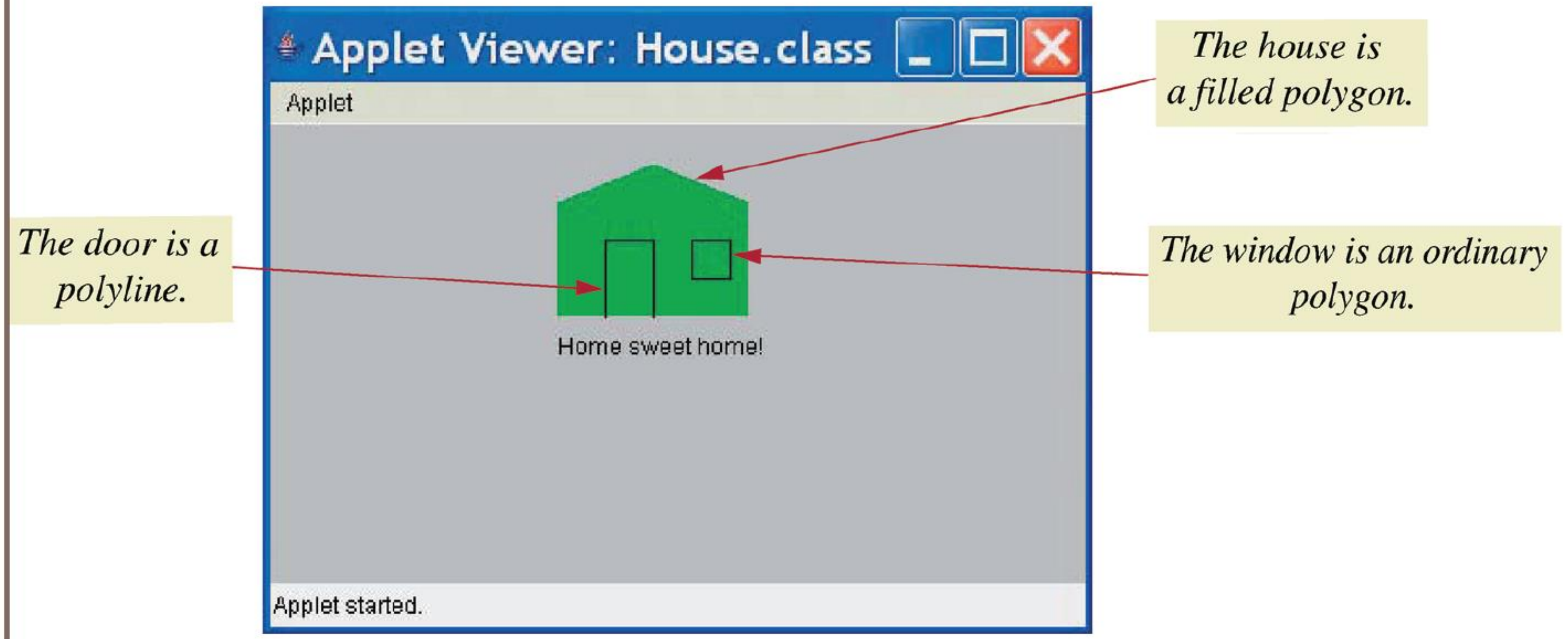
FIGURE 7.9 **A Polygon and a Polyline**



LISTING 7.16 An Applet with Polygons and a Polyline

```
import javax.swing.JApplet;
import java.awt.Color;
import java.awt.Graphics;

public class House extends JApplet
{
    private int[] xHouse = {150, 150, 200, 250, 250};
    private int[] yHouse = {100, 40, 20, 40, 100};
    private int[] xDoor = {175, 175, 200, 200};
    private int[] yDoor = {100, 60, 60, 100};
    private int[] xWindow = {220, 220, 240, 240};
    private int[] yWindow = {60, 80, 80, 60};

    public void paint(Graphics canvas)
    {
        super.paint(canvas);
        this.setBackground(Color.LIGT_GRAY);
        canvas.setColor(Color.GREEN);
        canvas.fillPolygon(xHouse, yHouse, xHouse.length);
        canvas.setColor(Color.BLACK);
        canvas.drawPolyline(xDoor, yDoor, xDoor.length);
        canvas.drawPolygon(xWindow, yWindow, xWindow.length);
        canvas.drawString("Home sweet home!", 150, 120);
    }
}
```

Applet Output


Applet Viewer: House.class
Applet
Home sweet home!
Applet started.
The house is a filled polygon.
The door is a polyline.
The window is an ordinary polygon.