# Chapter 7 Sample Programs

```
//********************************************************************
//  Words.java       Author: Lewis and Loftus
//  Demonstrates the use of an inherited method.
//********************************************************************
class Words
{
   //-----------------------------------------------------------------
   //  Instantiates a derived class and invokes its inherited and
   //  local methods.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Dictionary webster = new Dictionary ();

      webster.pageMessage();
      webster.definitionMessage();
   }
}

//********************************************************************
//  Book.java       Author: Lewis and Loftus
//  Represents a book.
//********************************************************************
class Book
{
   protected int pages = 1500;

   //-----------------------------------------------------------------
   //  Prints a message concerning the pages of this book.
   //-----------------------------------------------------------------
   public void pageMessage ()
   {
      System.out.println ("Number of pages: " + pages);
   }
}

//********************************************************************
//  Dictionary.java       Author: Lewis and Loftus
//  Represents a dictionary, which is a book.
//********************************************************************

class Dictionary extends Book
{
   private int definitions = 52500;
   //-----------------------------------------------------------------
   //  Prints a message using both local and inherited values.
   //-----------------------------------------------------------------
   public void definitionMessage ()
   {
      System.out.println ("Number of definitions: " + definitions);
      System.out.println ("Definitions per page: " + definitions/pages);
   }
}
```

```
//********************************************************************
//  Words2.java       Author: Lewis and Loftus
//  Demonstrates the use of the super reference.
//********************************************************************

class Words2
{
   //-----------------------------------------------------------------
   //  Instantiates a derived class and invokes its inherited and
   //  local methods.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Dictionary2 webster = new Dictionary2 (1500, 52500);

      webster.pageMessage();
      webster.definitionMessage();
   }
}

//********************************************************************
//  Book2.java       Author: Lewis and Loftus
//  Represents a book.
//********************************************************************

class Book2
{
   protected int pages;

   //-----------------------------------------------------------------
   //  Sets up the book with the specified number of pages.
   //-----------------------------------------------------------------
   public Book2 (int pages)
   {
      this.pages = pages;
   }

   //-----------------------------------------------------------------
   //  Prints a message concerning the pages of this book.
   //-----------------------------------------------------------------
   public void pageMessage ()
   {
      System.out.println ("Number of pages: " + pages);
   }
}
```

```
//********************************************************************
//  Dictionary2.java       Author: Lewis and Loftus
//  Represents a dictionary, which is a book.
//********************************************************************

class Dictionary2 extends Book2
{
   private int definitions;

   //-----------------------------------------------------------------
   //  Sets up the dictionary with the specified number of pages
   //  (maintained by the Book parent class) and defintions.
   //-----------------------------------------------------------------
   public Dictionary2 (int pages, int definitions)
   {
      super (pages);

      this.definitions = definitions;
   }

   //-----------------------------------------------------------------
   //  Prints a message using both local and inherited values.
   //-----------------------------------------------------------------
   public void definitionMessage ()
   {
      System.out.println ("Number of definitions: " + definitions);

      System.out.println ("Definitions per page: " + definitions/pages);
   }
}

//********************************************************************
//  Messages.java       Author: Lewis and Loftus
//  Demonstrates the use of an overridden method.
//********************************************************************

class Messages
{
   //-----------------------------------------------------------------
   //  Instatiates two objects a invokes the message method in each.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Thought parked = new Thought();
      Advice dates = new Advice();

      parked.message ();

      dates.message ();  // overridden
   }
}
```

```
//********************************************************************
//  Thought.java       Author: Lewis and Loftus
//  Represents a stray thought.
//********************************************************************

class Thought
{
   //-----------------------------------------------------------------
   //  Prints a message.
   //-----------------------------------------------------------------
   public void message()
   {
      System.out.println ("I feel like I'm diagonally parked in a " +
                          "parallel universe.");

      System.out.println();
   }
}


//********************************************************************
//  Advice.java       Author: Lewis and Loftus
//  Represents a hopefully helpful piece of advice.
//********************************************************************

class Advice extends Thought
{
   //-----------------------------------------------------------------
   //  Prints a message. This method overrides the parent's version.
   //  It also invokes the parent's version explicitly using super.
   //-----------------------------------------------------------------
   public void message()
   {
      System.out.println ("Warning: Dates in calendar are closer " +
                          "than they appear.");

      System.out.println();

      super.message();
   }
}
```

```
//********************************************************************
//  Academia.java       Author: Lewis and Loftus
//  Demonstrates inheritance from the Object class.
//********************************************************************
class Academia
{
   //-----------------------------------------------------------------
   //  Creates objects of two student types, prints some information
   //  about them, then checks them for equality.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Student susan = new Student ("Susan", 5);
      GradStudent frank = new GradStudent ("Frank", 3, "GTA", 8.75);

      System.out.println (susan);

      System.out.println ();

      System.out.println (frank);

      System.out.println ();

      if (! susan.equals(frank))
         System.out.println ("These are two different students.");
   }
}

//********************************************************************
//  Student.java       Author: Lewis and Loftus
//  Represents a student.
//********************************************************************
class Student
{
   protected String name;
   protected int numCourses;

   //-----------------------------------------------------------------
   //  Sets up a student with the specified name and number of courses.
   //-----------------------------------------------------------------
   public Student (String name, int numCourses)
   {
      this.name = name;
      this.numCourses = numCourses;
   }

   //-----------------------------------------------------------------
   //  Returns information about this student as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result = "Student name: " + name + "\n";
      result += "Number of courses: " + numCourses;
      return result;
   }
}
```

```
//********************************************************************
//  GradStudent.java          Lewis and Loftus
//  Represents a graduate student, with financial support.
//********************************************************************

class GradStudent extends Student
{
   private String source;
   private double rate;

   //-----------------------------------------------------------------
   //  Sets up the gradate student using the specified information.
   //-----------------------------------------------------------------
   public GradStudent (String name, int numCourses, String source,
                       double rate)
   {
      super (name, numCourses);

      this.source = source;
      this.rate = rate;
   }

   //-----------------------------------------------------------------
   //  Returns a description of this graduate student as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result = super.toString();

      result += "\nSupport source: " + source + "\n";
      result += "Hourly pay rate: " + rate;

      return result;
   }
}

//********************************************************************
//  Firm.java       Author: Lewis and Loftus
//  Demonstrates polymorphic processing.
//********************************************************************

class Firm
{
   //-----------------------------------------------------------------
   //  Creates a staff of employees for a firm and pays them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Staff personnel = new Staff();

      personnel.payday();
   }
}
```

```
//********************************************************************
//  Staff.java       Author: Lewis and Loftus
//  Represents the personnel staff of a particular business.
//********************************************************************
class Staff
{
   StaffMember[] staffList;

   //-----------------------------------------------------------------
   //  Sets up the list of staff members.
   //-----------------------------------------------------------------
   public Staff ()
   {
      staffList = new StaffMember[6];

      staffList[0] = new Executive ("Sam", "123 Main Line",
         "555-0469", "123-45-6789", 1923.07);
      staffList[1] = new Employee ("Carla", "456 Off Line",
         "555-0101", "987-65-4321", 846.15);
      staffList[2] = new Employee ("Woody", "789 Off Rocker",
         "555-0000", "010-20-3040", 769.23);
      staffList[3] = new Hourly ("Diane", "678 Fifth Ave.",
         "555-0690", "958-47-3625", 8.55);
      staffList[4] = new Volunteer ("Norm", "987 Suds Blvd.",
         "555-8374");
      staffList[5] = new Volunteer ("Cliff", "321 Duds Lane",
         "555-7282");

      ((Executive)staffList[0]).awardBonus (5000.00);

      ((Hourly)staffList[3]).addHours (40);
   }

   //-----------------------------------------------------------------
   //  Pays all staff members.
   //-----------------------------------------------------------------
   public void payday ()
   {
      double amount;

      for (int count=0; count < staffList.length; count++)
      {
         System.out.println (staffList[count]);

         amount = staffList[count].pay();  // polymorphic

         if (amount == 0.0)
            System.out.println ("Thanks!");
         else
            System.out.println ("Paid: " + amount);

         System.out.println ("-----------------------------------");
      }
   }
}
```

```
//********************************************************************
//  StaffMember.java       Author: Lewis and Loftus
//  Represents a generic staff member.
//********************************************************************
abstract class StaffMember
{
   protected String name;
   protected String address;
   protected String phone;
   //-----------------------------------------------------------------
   //  Sets up a staff member using the specified information.
   //-----------------------------------------------------------------
   public StaffMember (String name, String address, String phone)
   {
      this.name = name;
      this.address = address;
      this.phone = phone;
   }
   //-----------------------------------------------------------------
   //  Returns a string including the basic employee information.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result = "Name: " + name + "\n";
      result += "Address: " + address + "\n";
      result += "Phone: " + phone;
      return result;
   }
   //-----------------------------------------------------------------
   //  Derived classes must define the pay method for each employee type.
   //-----------------------------------------------------------------
   public abstract double pay();
}

//********************************************************************
//  Volunteer.java       Author: Lewis and Loftus
//  Represents a staff member that works as a volunteer.
//********************************************************************
class Volunteer extends StaffMember
{
   //-----------------------------------------------------------------
   //  Sets up a volunteer using the specified information.
   //-----------------------------------------------------------------
   public Volunteer (String name, String address, String phone)
   {
      super (name, address, phone);
   }

   //-----------------------------------------------------------------
   //  Returns a zero pay value for this volunteer.
   //-----------------------------------------------------------------
   public double pay()
   {
      return 0.0;
   }
}
```

```
//********************************************************************
//  Employee.java       Author: Lewis and Loftus
//  Represents a general paid employee.
//********************************************************************
class Employee extends StaffMember
{
   protected String socialSecurityNumber;
   protected double payRate;
   //-----------------------------------------------------------------
   //  Sets up an employee with the specified information.
   //-----------------------------------------------------------------
   public Employee (String name, String address, String phone,
                    String socialSecurityNumber, double payRate)
   {
      super (name, address, phone);
      this.socialSecurityNumber = socialSecurityNumber;
      this.payRate = payRate;
   }
   //-----------------------------------------------------------------
   //  Returns information about an employee as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result = super.toString();
      result += "\nSocial Security Number: " + socialSecurityNumber;
      return result;
   }
   //-----------------------------------------------------------------
   //  Returns the pay rate for this employee.
   //-----------------------------------------------------------------
   public double pay ()
   {     return payRate;
   }
}

//********************************************************************
//  Executive.java       Author: Lewis and Loftus
//  Represents an executive staff member, who can earn a bonus.
//********************************************************************
class Executive extends Employee
{
   private double bonus;
   //-----------------------------------------------------------------
   //  Sets up an executive with the specified information.
   //-----------------------------------------------------------------
   public Executive (String name, String address, String phone,
                                String socialSecurityNumber, double payRate)
   {
      super (name, address, phone, socialSecurityNumber, payRate);
      bonus = 0;                        // bonus has yet to be awarded
   }
   //-----------------------------------------------------------------
   //  Awards the specified bonus to this executive.
   //-----------------------------------------------------------------
   public void awardBonus (double execBonus)
   {     bonus = execBonus;   }
```

```
   //-----------------------------------------------------------------
   //  Computes and returns the pay for an executive, which is the
   //  regular employee payment plus a one-time bonus.
   //-----------------------------------------------------------------
   public double pay ()
   {
      double payment = super.pay() + bonus;
      bonus = 0;
      return payment;
   }
}

//********************************************************************
//  Hourly.java       Author: Lewis and Loftus
//  Represents an employee that gets paid by the hour.
//********************************************************************
class Hourly extends Employee
{
   private int hoursWorked;

   //-----------------------------------------------------------------
   //  Sets up this hourly employee using the specified information.
   //-----------------------------------------------------------------
   public Hourly (String name, String address, String phone, String socialSecurityNumber, double payRate)
   {
      super (name, address, phone, socialSecurityNumber, payRate);
      hoursWorked = 0;
   }
   //-----------------------------------------------------------------
   //  Adds the specified number of hours to this employee's accumulated hours.
   //-----------------------------------------------------------------
   public void addHours (int moreHours)
   {
      hoursWorked += moreHours;
   }
   //-----------------------------------------------------------------
   //  Computes and returns the pay for this hourly employee.
   //-----------------------------------------------------------------
   public double pay ()
   {
      double payment = payRate * hoursWorked;
      hoursWorked = 0;
      return payment;
   }
   //-----------------------------------------------------------------
   //  Returns information about this hourly employee as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result = super.toString();
      result += "\nCurrent hours: " + hoursWorked;
      return result;
   }
}
```

```
//********************************************************************
//  FoodAnalysis.java       Author: Lewis and Loftus
//  Demonstrates indirect referencing through inheritance.
//********************************************************************
class FoodAnalysis
{
   //-----------------------------------------------------------------
   //  Instantiates a Pizza object and prints its calories per
   //  serving.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Pizza special = new Pizza (275);

      System.out.println ("Calories per serving: " +
                          special.caloriesPerServing());
   }
}

//********************************************************************
//  FoodItem.java       Author: Lewis and Loftus
//  Demonstrates indirect referencing through inheritance.
//********************************************************************
class FoodItem
{
   final private int CALORIES_PER_GRAM = 9;
   private int fatGrams;
   protected int servings;

   //-----------------------------------------------------------------
   //  Sets up this food item with the specified number of fat grams
   //  and number of servings.
   //-----------------------------------------------------------------
   public FoodItem (int fatGrams, int servings)
   {
      this.fatGrams = fatGrams;
      this.servings = servings;
   }

   //-----------------------------------------------------------------
   //  Computes and returns the number of calories in this food item
   //  due to fat.
   //-----------------------------------------------------------------
   private int calories ()
   {
      return fatGrams * CALORIES_PER_GRAM;
   }

   //-----------------------------------------------------------------
   //  Computes and returns the number of fat calories per serving.
   //-----------------------------------------------------------------
   public int caloriesPerServing ()
   {
      return (calories() / servings);
   }
}
```

```
//********************************************************************
//  Pizza.java       Author: Lewis and Loftus
//  Demonstrates indirect referencing through inheritance.
//********************************************************************

class Pizza extends FoodItem
{
   //-----------------------------------------------------------------
   //  Sets up a pizza with the specified amount of fat; assumes
   //  eight servings.
   //-----------------------------------------------------------------
   public Pizza (int fatGrams)
   {
      super (fatGrams, 8);
   }
}

//********************************************************************
//  OffCenter.java       Author: Lewis and Loftus
//  Demonstrates the extension of an event adatpter class.
//********************************************************************

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.text.DecimalFormat;

public class OffCenter extends Applet
{
   private final int APPLET_WIDTH = 200;
   private final int APPLET_HEIGHT = 200;

   private DecimalFormat fmt;
   private Point current;
   private int centerX, centerY;
   private double length;

   //-----------------------------------------------------------------
   //  Sets up the applet, including creating a listener from the
   //  inner class and adding it to the applet.
   //-----------------------------------------------------------------
   public void init()
   {
      addMouseListener (new OffCenterListener());

      centerX = APPLET_WIDTH / 2;
      centerY = APPLET_HEIGHT / 2;

      fmt = new DecimalFormat ("0.##");

      setBackground (Color.yellow);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }
```

```
   //-----------------------------------------------------------------
   //  Draws a line from the mouse pointer to the center point of
   //  the applet and displays the distance.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.setColor (Color.black);
      if (current != null)
      {
         page.drawLine (current.x, current.y, centerX, centerY);
         page.drawString ("Distance: " + fmt.format(length), 50, 15);
      }
   }

   class OffCenterListener extends MouseAdapter
   {
      //-----------------------------------------------------------
      //  Computes the distance from the mouse pointer to the center
      //  point of the applet.
      //-----------------------------------------------------------
      public void mouseClicked (MouseEvent event)
      {
         current = event.getPoint();
         length = Math.sqrt(Math.pow((current.x-centerX), 2) +
                            Math.pow((current.y-centerY), 2));
         repaint();
      }
   }
}

<! OffCenter.html>
<HTML>

<HEAD>
<TITLE>The OffCenter Applet</TITLE>
</HEAD>
<BODY>
<center>
<H3>The OffCenter Applet:</H3>
<APPLET CODE="OffCenter.class" WIDTH=200 HEIGHT=200></APPLET>
<HR>
</center>
</BODY>
</HTML>
```

```
//********************************************************************
//  Fahrenheit.java       Author: Lewis and Loftus
//  Demonstrates the use of GUI components.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Fahrenheit extends Applet implements ActionListener
{
   private int APPLET_WIDTH = 200;
   private int APPLET_HEIGHT = 100;
   private Label inputLabel, outputLabel, resultLabel;
   private TextField fahrenheit;
   //-----------------------------------------------------------------
   //  Sets up the applet with its various components.
   //-----------------------------------------------------------------
   public void init()
   {
      inputLabel = new Label ("Enter Fahrenheit:");
      outputLabel = new Label ("Temperature in Celcius:");
      resultLabel = new Label ("N/A");
      fahrenheit = new TextField (5);
      fahrenheit.addActionListener (this);

      add (inputLabel);
      add (fahrenheit);
      add (outputLabel);
      add (resultLabel);
      setBackground (Color.white);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }
   //-----------------------------------------------------------------
   //  Performs the conversion when the enter key is pressed in the text field.
   //-----------------------------------------------------------------
   public void actionPerformed (ActionEvent event)
   {
      int fahrenheitTemp, celciusTemp;
      String text = fahrenheit.getText();
      fahrenheitTemp = Integer.parseInt (text);
      celciusTemp = (fahrenheitTemp-32) * 5/9;
      resultLabel.setText (Integer.toString (celciusTemp));
   }
}

<! Fahrenheit.html>
<HTML>
   <HEAD>
      <TITLE>The Fahrenheit Applet</TITLE>
   </HEAD>
   <BODY>
      <H3>The Fahrenheit Applet:</H3>
      <APPLET CODE="Fahrenheit.class" WIDTH=200 HEIGHT=100></APPLET>
      <HR>
   </BODY>
</HTML>
```

```
//********************************************************************
//  Doodle.java       Author: Lewis and Loftus
//  Demonstrates the use of GUI components.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class Doodle extends Applet implements ActionListener
{
   private int APPLET_WIDTH = 300;
   private int APPLET_HEIGHT = 250;

   private Label titleLabel;
   private DoodleCanvas canvas;
   private Button clearButton;
   //-----------------------------------------------------------------
   //  Creates the GUI components and adds them to the applet. The
   //  applet serves as the listener for the button.
   //-----------------------------------------------------------------
   public void init ()
   {
      titleLabel = new Label ("Doodle using the mouse.");
      titleLabel.setBackground (Color.cyan);
      add (titleLabel);

      canvas = new DoodleCanvas();
      add (canvas);

      clearButton = new Button("Clear");
      clearButton.addActionListener (this);
      add (clearButton);

      setBackground (Color.cyan);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }
   //-----------------------------------------------------------------
   //  Clears the canvas when the clear button is pushed.
   //-----------------------------------------------------------------
   public void actionPerformed (ActionEvent event)
   {
      canvas.clear();
   }
}

<! Doodle.html>
<HTML>
   <HEAD>
      <TITLE>The Doodle Applet</TITLE>
   </HEAD>
   <BODY>
      <H3>The Doodle Applet:</H3>
      <APPLET CODE="Doodle.class" WIDTH=300 HEIGHT=300></APPLET>
      <HR>
   </BODY>
</HTML>
```

```
//********************************************************************
//  DoodleCanvas.java       Author: Lewis and Loftus
//  Represents a drawing surface for creating simple doodles.
//********************************************************************
import java.awt.*;
import java.awt.event.*;

class DoodleCanvas extends Canvas implements MouseListener, MouseMotionListener
{
   private final int CANVAS_WIDTH = 200;
   private final int CANVAS_HEIGHT = 200;

   private int lastX, lastY;

   //-----------------------------------------------------------------
   //  Creates an initially empty canvas.
   //-----------------------------------------------------------------
   public DoodleCanvas ()
   {
      addMouseListener (this);
      addMouseMotionListener (this);

      setBackground (Color.white);
      setSize (CANVAS_WIDTH, CANVAS_HEIGHT);
   }
   //-----------------------------------------------------------------
   //  Determines up the initial point for a new doodle line.
   //-----------------------------------------------------------------
   public void mousePressed (MouseEvent event)
   {
      Point first = event.getPoint();
      lastX = first.x;
      lastY = first.y;
   }
   //-----------------------------------------------------------------
   //  Draws a line from the last point to the current point.
   //-----------------------------------------------------------------
   public void mouseDragged (MouseEvent event)
   {
      Point current = event.getPoint();

      Graphics page = getGraphics();
      page.drawLine (lastX, lastY, current.x, current.y);

      lastX = current.x;
      lastY = current.y;
   }
   //-----------------------------------------------------------------
   //  Clears the canvas.
   //-----------------------------------------------------------------
   public void clear ()
   {
      Graphics page = getGraphics();
      page.drawRect (0, 0, CANVAS_WIDTH, CANVAS_HEIGHT);
      repaint();
   }
```

```
    //---------------------------------------------------------------
    //  Provide empty definitions for unused event methods.
    //---------------------------------------------------------------
    public void mouseReleased (MouseEvent event) {}
    public void mouseClicked (MouseEvent event) {}
    public void mouseEntered (MouseEvent event) {}
    public void mouseExited (MouseEvent event) {}
    public void mouseMoved (MouseEvent event) {}
}
```