# Chapter 5 Sample Programs

```
//********************************************************************
//  ParameterPassing.java       Author: Lewis and Loftus
//  Demonstrates the effects of passing various types of parameters.
//********************************************************************
class ParameterPassing
{
   //-----------------------------------------------------------------
   //  Sets up three variables (one primitive and two objects) to
   //  serve as actual parameters to the changeValues method. Prints
   //  their values before and after calling the method.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      ParameterTester tester = new ParameterTester();
      int a1 = 111;
      Num a2 = new Num (222);
      Num a3 = new Num (333);

      System.out.println ("Before calling changeValues:");
      System.out.println ("a1\ta2\ta3");
      System.out.println (a1 + "\t" + a2 + "\t" + a3 + "\n");
      tester.changeValues (a1, a2, a3);
      System.out.println ("After calling changeValues:");
      System.out.println ("a1\ta2\ta3");
      System.out.println (a1 + "\t" + a2 + "\t" + a3 + "\n");
   }
}

//********************************************************************
//  ParameterTester.java       Author: Lewis and Loftus
//  Demonstrates the effects of passing various types of parameters.
//********************************************************************
class ParameterTester
{
   //-----------------------------------------------------------------
   //  Modifies the parameters, printing their values before and
   //  after making the changes.
   //-----------------------------------------------------------------
   public void changeValues (int f1, Num f2, Num f3)
   {
      System.out.println ("Before changing the values:");
      System.out.println ("f1\tf2\tf3");
      System.out.println (f1 + "\t" + f2 + "\t" + f3 + "\n");

      f1 = 999;
      f2.setValue(888);
      f3 = new Num (777);

      System.out.println ("After changing the values:");
      System.out.println ("f1\tf2\tf3");
      System.out.println (f1 + "\t" + f2 + "\t" + f3 + "\n");
   }
}
```

```
//********************************************************************
//  Num.java       Author: Lewis and Loftus
//  Represents a single integer as an object.
//********************************************************************

class Num
{
   private int value;

   //-----------------------------------------------------------------
   //  Sets up the new Num object, storing an initial value.
   //-----------------------------------------------------------------
   public Num (int update)
   {
      value = update;
   }

   //-----------------------------------------------------------------
   //  Sets the stored value to the newly specified value.
   //-----------------------------------------------------------------
   public void setValue (int update)
   {
      value = update;
   }

   //-----------------------------------------------------------------
   //  Returns the stored integer value as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      return value + "";
   }
}

//********************************************************************
//  CountInstances.java       Author: Lewis and Loftus
//  Demonstrates the use of the static modifier.
//********************************************************************

class CountInstances
{
   //-----------------------------------------------------------------
   //  Creates several MyClass objects and prints the number of
   //  objects that were created.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      MyClass obj;

      for (int scan=1; scan <= 10; scan++)
         obj = new MyClass();

      System.out.println ("Objects created: " + MyClass.getCount());
   }
}
```

```
//********************************************************************
//  MyClass.java       Author: Lewis and Loftus
//  Demonstrates the use of the static modifier.
//********************************************************************
class MyClass
{
   private static int count = 0;

   //-----------------------------------------------------------------
   //  Counts the number of instances created.
   //-----------------------------------------------------------------
   public MyClass ()
   {
      count++;
   }

   //-----------------------------------------------------------------
   //  Returns the number of instances of this class that have been
   //  created.
   //-----------------------------------------------------------------
   public static int getCount ()
   {
      return count;
   }
}

//********************************************************************
//  Speaker.java       Author: Lewis and Loftus
//  Demonstrates the declaration of an interface.
//********************************************************************
interface Speaker
{
   public void speak ();
   public void announce (String str);
}

//********************************************************************
//  Philosopher.java       Author: Lewis and Loftus
//  Demonstrates the implementation of an interface.
//********************************************************************
class Philosopher implements Speaker
{
   private String philosophy;

   //-----------------------------------------------------------------
   //  Establishes this philosopher's philosophy.
   //-----------------------------------------------------------------
   public Philosopher (String philosophy)
   {
      this.philosophy = philosophy;
   }
```

```
   //-----------------------------------------------------------------
   //  Prints this philosophers's philosophy.
   //-----------------------------------------------------------------
   public void speak ()
   {
      System.out.println (philosophy);
   }

   //-----------------------------------------------------------------
   //  Prints the specified announcement.
   //-----------------------------------------------------------------
   public void announce (String announcement)
   {
      System.out.println (announcement);
   }

   //-----------------------------------------------------------------
   //  Prints this philosophers's philosophy multiple times.
   //-----------------------------------------------------------------
   public void pontificate ()
   {
      for (int count=1; count <= 5; count++)
         System.out.println (philosophy);
   }
}
//********************************************************************
//  Dog.java       Author: Lewis and Loftus
//
//  Demonstrates the implementation of an interface.
//********************************************************************
class Dog implements Speaker
{
   //-----------------------------------------------------------------
   //  Prints this dog's philosophy.
   //-----------------------------------------------------------------
   public void speak ()
   {
      System.out.println ("woof");
   }

   //-----------------------------------------------------------------
   //  Prints this dog's philosophy and the specified announcement.
   //-----------------------------------------------------------------
   public void announce (String announcement)
   {
      System.out.println ("woof: " + announcement);
   }
}
```

```
//********************************************************************
//  Talking.java       Author: Lewis and Loftus
//  Demonstrates the use of an interface for polymorphic references.
//********************************************************************

class Talking
{
   //-----------------------------------------------------------------
   //  Instantiates two objects using an interface reference and
   //  invokes one of the common methods. Then casts the interface
   //  reference into a class reference to invoke its unique method.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Speaker current;

      current = new Dog();
      current.speak();

      current = new Philosopher ("I think, therefore I am.");
      current.speak();

      ((Philosopher) current).pontificate();
   }
}

//********************************************************************
//  Dots.java       Author: Lewis and Loftus
//  Demonstrates events and listeners.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
// import java.awt.event.*;

public class Dots extends Applet
{
   private final int APPLET_WIDTH = 200;
   private final int APPLET_HEIGHT = 100;
   private final int RADIUS = 6;
   private Point clickPoint = null;

   //-----------------------------------------------------------------
   //  Creates a listener for mouse events for this applet.
   //-----------------------------------------------------------------
   public void init()
   {
      DotsMouseListener listener = new DotsMouseListener(this);
      addMouseListener(listener);

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

```
   //-----------------------------------------------------------------
   //  Draws the dot at the appropriate location.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.setColor (Color.green);
      if (clickPoint != null)
         page.fillOval (clickPoint.x - RADIUS, clickPoint.y - RADIUS,
                        RADIUS * 2, RADIUS * 2);
   }

   //-----------------------------------------------------------------
   //  Sets the point at which to draw the next dot.
   //-----------------------------------------------------------------
   public void setPoint (Point point)
   {
      clickPoint = point;
   }
}

<! Dots.html>
<HTML>
<HEAD>
<TITLE>The Dots Applet</TITLE>
</HEAD>

<BODY>
<center>
<H3>The Dots Applet:</H3>
<APPLET CODE="Dots.class" WIDTH=200 HEIGHT=100>
</APPLET>
<HR>
</center>
</BODY>
</HTML>


//********************************************************************
//  DotsMouseListener.java       Author: Lewis and Loftus
//  Represents a listener object for mouse events.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

class DotsMouseListener implements MouseListener
{
   private Dots applet;

   //-----------------------------------------------------------------
   //  Stores a reference to the applet.
   //-----------------------------------------------------------------
   public DotsMouseListener (Dots applet)
   {
      this.applet = applet;
   }
```

```
   //-----------------------------------------------------------------
   //  Determines the point at which the mouse is clicked, sets the
   //  point in the applet, then forces the applet to repaint.
   //-----------------------------------------------------------------
   public void mouseClicked (MouseEvent event)
   {
      Point clickPoint = event.getPoint();
      applet.setPoint (clickPoint);
      applet.repaint();
   }

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

//********************************************************************
//  RubberLines.java       Author: Lewis and Loftus
//  Demonstrates events, listeners and rubberbanding.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class RubberLines extends Applet implements MouseListener,
                                 MouseMotionListener
{
   private final int APPLET_WIDTH = 200;
   private final int APPLET_HEIGHT = 200;
   private Point point1 = null;
   private Point point2 = null;

   //-----------------------------------------------------------------
   //  Adds this class as a listener for all mouse related events.
   //-----------------------------------------------------------------
   public void init()
   {
      addMouseListener (this);
      addMouseMotionListener (this);
      setBackground (Color.black);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }

   //-----------------------------------------------------------------
   //  Draws the current line from the intial mouse down point to
   //  the current position of the mouse.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.setColor (Color.green);
      if (point1 != null && point2 != null)
         page.drawLine (point1.x, point1.y, point2.x, point2.y);
   }
```

```
   //-----------------------------------------------------------------
   //  Captures the position at which the mouse is initially pushed.
   //-----------------------------------------------------------------
   public void mousePressed (MouseEvent event)
   {
      point1 = event.getPoint();
   }

   //-----------------------------------------------------------------
   //  Gets the current position of the mouse as it is dragged and
   //  draws the line to create the rubberband effect.
   //-----------------------------------------------------------------
   public void mouseDragged (MouseEvent event)
   {
      point2 = event.getPoint();
      repaint();
   }

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

<! RubberLines.html>
<HTML>
<HEAD>
<TITLE>The RubberLines Applet</TITLE>
</HEAD>

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


//********************************************************************
//  Direction.java       Author: Lewis and Loftus
//  Demonstrates key events and the use of inner classes for event
//  listeners.
//********************************************************************

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class Direction extends Applet
{
   private final int APPLET_WIDTH = 200;
   private final int APPLET_HEIGHT = 200;
   private final int JUMP = 5;  // increment for image movement

   private final int IMAGE_SIZE = 31;

   private Image up, down, right, left, currentImage;
   private AudioClip bonk;
   private int x, y;

   //-----------------------------------------------------------------
   //  Sets up the applet by creating listeners, loading images, etc.
   //-----------------------------------------------------------------
   public void init()
   {
      requestFocus();  // make sure the applet has the keyboard focus

      addKeyListener (new DirectionKeyListener());

      x = y = 0;

      up = getImage (getCodeBase(), "cyanUp.gif");
      down = getImage (getCodeBase(), "cyanDown.gif");
      left = getImage (getCodeBase(), "cyanLeft.gif");
      right = getImage (getCodeBase(), "cyanRight.gif");

      currentImage = right;

      bonk = getAudioClip (getCodeBase(), "bonk.au");

      setBackground (Color.black);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }

   //-----------------------------------------------------------------
   //  Paints the current image in the current location.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.drawImage (currentImage, x, y, this);
   }
```

```
   //*****************************************************************
   //  Represents a listener for keyboard activity.
   //*****************************************************************
   private class DirectionKeyListener implements KeyListener
   {
      //--------------------------------------------------------------
      //  Responds to the user pressing arrow keys by adjusting the
      //  image location accordingly.
      //--------------------------------------------------------------
      public void keyPressed (KeyEvent event)
      {
         switch (event.getKeyCode())
         {
            case KeyEvent.VK_UP:   currentImage = up;
                                                   if (y > 0)
                                                   y -= JUMP;
                                                   break;
            case KeyEvent.VK_DOWN:  currentImage = down;
                                                   if (y < APPLET_HEIGHT-IMAGE_SIZE)
                                                   y += JUMP;
                                                   break;
            case KeyEvent.VK_LEFT:  currentImage = left;
                                                   if (x > 0)
                                                   x -= JUMP;
                                                   break;
            case KeyEvent.VK_RIGHT: currentImage = right;
                                                   if (x < APPLET_WIDTH-IMAGE_SIZE)
                                                   x += JUMP;
                                                   break;
            default:               bonk.play();
         }
         repaint();
      }

      //--------------------------------------------------------------
      //  Provide empty definitions for unused event methods.
      //--------------------------------------------------------------
      public void keyTyped (KeyEvent event) {}
      public void keyReleased (KeyEvent event) {}
   }
}

<! Direction.html>
<HTML>
<HEAD>
<TITLE>The Direction Applet</TITLE>
</HEAD>
<BODY>
<center>

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

```
//********************************************************************
//  Rebound.java       Author: Lewis and Loftus
//  Demonstrates an animation and the use of the Timer class.
//********************************************************************
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import javax.swing.Timer;
public class Rebound extends Applet
{
   private final int APPLET_WIDTH = 200;
   private final int APPLET_HEIGHT = 100;
   private final int IMAGE_SIZE = 35;
   private final int DELAY = 20;
   private Timer timer;
   private Image image;
   private int x, y, moveX, moveY;

   //-----------------------------------------------------------------
   //  Sets up the applet, including the timer for the animation.
   //-----------------------------------------------------------------
   public void init()
   {
      addMouseListener (new ReboundMouseListener());
      timer = new Timer (DELAY, new ReboundActionListener());
      timer.start();

      x = 0;  y = 40;
      moveX = moveY = 3;
      image = getImage (getCodeBase(), "happyFace.gif");
      setBackground (Color.black);
      setSize (APPLET_WIDTH, APPLET_HEIGHT);
   }

   //-----------------------------------------------------------------
   //  Draws the image in the current location.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      page.drawImage (image, x, y, this);
   }

   //*****************************************************************
   //  Represents the mouse listner for the applet.
   //*****************************************************************
   private class ReboundMouseListener implements MouseListener
   {
      //--------------------------------------------------------------
      //  Stops or starts the timer (and therefore the animation)
      //  when the mouse button is clicked.
      //--------------------------------------------------------------
      public void mouseClicked (MouseEvent event)
      {
         if (timer.isRunning())  timer.stop();
         else                    timer.start();
      }
```

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

   //*****************************************************************
   //  Represents the action listener for the timer.
   //*****************************************************************
   private class ReboundActionListener implements ActionListener
   {
      //-----------------------------------------------------------
      //  Updates the position of the image and possibly the direction
      //  of movement whenever the timer fires an action event.
      //-----------------------------------------------------------
      public void actionPerformed (ActionEvent event)
      {
         x += moveX;
         y += moveY;

         if (x <= 0 || x >= APPLET_WIDTH-IMAGE_SIZE)
            moveX = moveX * -1;
         if (y <= 0 || y >= APPLET_HEIGHT-IMAGE_SIZE)
            moveY = moveY * -1;

         repaint();
      }
   }
}

<! Rebound.html>
<HTML>
<HEAD>
<TITLE>The Rebound Applet</TITLE>
</HEAD>

<BODY>
<center>
<H3>The Rebound Applet:</H3>
<APPLET CODE="Rebound.class" WIDTH=200 HEIGHT=100></APPLET>
<HR>
</center>
</BODY>
</HTML>
```