Chapter 4 Sample Programs

```
//********************************************************************
//  CountFlips.java       Author: Lewis and Loftus
//  Demonstrates the use of a programmer-defined class.
//********************************************************************

import Coin;

public class CountFlips
{
   //-----------------------------------------------------------------
   //  Flips a coin multiple times and counts the number of heads
   //  and tails that result.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int NUM_FLIPS = 1000;
      int heads = 0, tails = 0;

      Coin myCoin = new Coin();  // instantiate the Coin object

      for (int count=1; count <= NUM_FLIPS; count++)
      {
         myCoin.flip();

         if (myCoin.getFace() == myCoin.HEADS)
            heads++;
         else
            tails++;
      }

      System.out.println ("The number flips: " + NUM_FLIPS);
      System.out.println ("The number of heads: " + heads);
      System.out.println ("The number of tails: " + tails);
   }
}
```

```
//********************************************************************
//  Coin.java       Author: Lewis and Loftus
//  Represents a coin with two sides that can be flipped.
//********************************************************************

public class Coin
{
   public final int HEADS = 0;
   public final int TAILS = 1;

   private int face;


   //-----------------------------------------------------------------
   //  Sets up the coin by flipping it initially.
   //-----------------------------------------------------------------
   public Coin ()
   {
      flip();
   }

   //-----------------------------------------------------------------
   //  Flips the coin by randomly choosing a face.
   //-----------------------------------------------------------------
   public void flip ()
   {
      face = (int) (Math.random() * 2);
   }

   //-----------------------------------------------------------------
   //  Returns the current face of the coin as an integer.
   //-----------------------------------------------------------------
   public int getFace ()
   {
      return face;
   }

   //-----------------------------------------------------------------
   //  Returns the current face of the coin as a string.
   //-----------------------------------------------------------------
   public String toString()
   {
      String faceName;

      if (face == HEADS)
         faceName = "Heads";
      else
         faceName = "Tails";

      return faceName;
   }
}
```

```
//********************************************************************
//  FlipRace.java       Author: Lewis and Loftus
//  Demonstrates the existence of separate data space in multiple
//  instantiations of a programmer-defined class.
//********************************************************************

import Coin;

public class FlipRace
{
   //-----------------------------------------------------------------
   //  Flips two coins until one of them comes up heads a set number
   //  of times in a row.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int GOAL = 3;
      int count1 = 0, count2 = 0;

      // Create two separate coin objects
      Coin coin1 = new Coin();
      Coin coin2 = new Coin();

      while (count1 < GOAL && count2 < GOAL)
      {
         coin1.flip();
         coin2.flip();

         // Print the flip results (uses Coin's toString method)
         System.out.print ("Coin 1: " + coin1);
         System.out.println ("   Coin 2: " + coin2);

         // Increment or reset the counters
         count1 = (coin1.getFace() == coin1.HEADS) ? count1+1 : 0;
         count2 = (coin2.getFace() == coin2.HEADS) ? count2+1 : 0;
      }

      // Determine the winner
      if (count1 < GOAL)
         System.out.println ("Coin 2 Wins!");
      else
         if (count2 < GOAL)
            System.out.println ("Coin 1 Wins!");
         else
            System.out.println ("It's a TIE!");
   }
}
```

```
//********************************************************************
//  BankAccounts.java       Author: Lewis and Loftus
//  Driver to exercise the use of multiple Account objects.
//********************************************************************

import Account;

public class BankAccounts
{
   //-----------------------------------------------------------------
   //  Creates some bank accounts and requests various services.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Account acct1 = new Account ("Ted Murphy", 72354, 102.56);
      Account acct2 = new Account ("Jane Smith", 69713, 40.00);
      Account acct3 = new Account ("Edward Demsey", 93757, 759.32);

      acct1.deposit (25.85);

      double smithBalance = acct2.deposit (500.00);
      System.out.println ("Smith balance after deposit: " +
                          smithBalance);

      System.out.println ("Smith balance after withdrawal: " +
                          acct2.withdraw (430.75, 1.50));

      acct3.withdraw (800.00, 0.0);  // exceeds balance

      acct1.addInterest();
      acct2.addInterest();
      acct3.addInterest();

      System.out.println ();
      System.out.println (acct1);
      System.out.println (acct2);
      System.out.println (acct3);
   }
}
```

```
//********************************************************************
//  Account.java       Author: Lewis and Loftus
//  Represents a bank account with basic services such as deposit
//  and withdraw.
//********************************************************************

import java.text.NumberFormat;

public class Account
{
   private NumberFormat fmt = NumberFormat.getCurrencyInstance();

   private final double RATE = 0.045;  // interest rate of 4.5%

   private long acctNumber;
   private double balance;
   private String name;

   //-----------------------------------------------------------------
   //  Sets up the account by defining its owner, account number,
   //  and initial balance.
   //-----------------------------------------------------------------
   public Account (String owner, long account, double initial)
   {
      name = owner;
      acctNumber = account;
      balance = initial;
   }

   //-----------------------------------------------------------------
   //  Validates the transaction, then deposits the specified amount
   //  into the account. Returns the new balance.
   //-----------------------------------------------------------------
   public double deposit (double amount)
   {
      if (amount < 0)  // deposit value is negative
      {
         System.out.println ();
         System.out.println ("Error: Deposit amount is invalid.");
         System.out.println (acctNumber + "  " + fmt.format(amount));
      }
      else
         balance = balance + amount;

      return balance;
   }

   //-----------------------------------------------------------------
   //  Validates the transaction, then withdraws the specified amount
   //  from the account. Returns the new balance.
   //-----------------------------------------------------------------

   public double withdraw (double amount, double fee)
   {
      amount += fee;
```

```
      if (amount < 0)  // withdraw value is negative
      {
        System.out.println ();
        System.out.println ("Error: Withdraw amount is invalid.");
        System.out.println ("Account: " + acctNumber);
        System.out.println ("Requested: " + fmt.format(amount));
      }
      else
        if (amount > balance)  // withdraw value exceeds balance
        {
          System.out.println ();
          System.out.println ("Error: Insufficient funds.");
          System.out.println ("Account: " + acctNumber);
          System.out.println ("Requested: " + fmt.format(amount));
          System.out.println ("Available: " + fmt.format(balance));
        }
        else
          balance = balance - amount;
      return balance;
    }

    //-----------------------------------------------------------------
    //  Adds interest to the account and returns the new balance.
    //-----------------------------------------------------------------
    public double addInterest ()
    {
      balance += (balance * RATE);
      return balance;
    }

    //-----------------------------------------------------------------
    //  Returns the current balance of the account.
    //-----------------------------------------------------------------
    public double getBalance ()
    {
      return balance;
    }

    //-----------------------------------------------------------------
    //  Returns the account number.
    //-----------------------------------------------------------------
    public long getAccountNumber ()
    {
      return acctNumber;
    }

    //-----------------------------------------------------------------
    //  Returns a one-line description of the account as a string.
    //-----------------------------------------------------------------
    public String toString ()
    {
      return (acctNumber + "\t" + name + "\t" + fmt.format(balance));
    }
  }
```

```
//********************************************************************
//  RationalNumbers.java       Author: Lewis and Loftus
//
//  Driver to exercise the use of multiple Rational objects.
//********************************************************************

import Rational;

public class RationalNumbers
{
   //-----------------------------------------------------------------
   //  Creates some rational number objects and performs various
   //  operations on them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Rational r1 = new Rational (6, 8);
      Rational r2 = new Rational (1, 3);

      System.out.println ("First rational number: " + r1);
      System.out.println ("Second rational number: " + r2);

      if (r1.equals(r2))
         System.out.println ("r1 and r2 are equal.");
      else
         System.out.println ("r1 and r2 are NOT equal.");

      Rational r3 = r1.add(r2);
      Rational r4 = r1.subtract(r2);
      Rational r5 = r1.multiply(r2);
      Rational r6 = r1.divide(r2);

      System.out.println ("r1 + r2: " + r3);
      System.out.println ("r1 - r2: " + r4);
      System.out.println ("r1 * r2: " + r5);
      System.out.println ("r1 / r2: " + r6);
   }
}
```

```
//********************************************************************
//  Rational.java       Author: Lewis and Loftus
//
//  Represents one rational number with a numerator and denominator.
//********************************************************************

public class Rational
{
  private int numerator, denominator;

  //-----------------------------------------------------------------
  //  Sets up the rational number by ensuring a nonzero denominator
  //  and making only the numerator signed.
  //-----------------------------------------------------------------
  public Rational (int numer, int denom)
  {
    if (denom == 0)
      denom = 1;

    // Make the numerator "store" the sign
    if (denom < 0)
    {
      numer = numer * -1;
      denom = denom * -1;
    }

    numerator = numer;
    denominator = denom;

    reduce();
  }

  //-----------------------------------------------------------------
  //  Adds this rational number to the one passed as a parameter.
  //  A common denominator is found by multiplying the individual
  //  denominators.
  //-----------------------------------------------------------------
  public Rational add (Rational op2)
  {
    int commonDenominator = denominator * op2.getDenominator();
    int numerator1 = numerator * op2.getDenominator();
    int numerator2 = op2.getNumerator() * denominator;
    int sum = numerator1 + numerator2;

    return new Rational (sum, commonDenominator);
  }
```

```
//-----------------------------------------------------------------
//  Subtracts the rational number passed as a parameter from this
//  rational number.
//-----------------------------------------------------------------
public Rational subtract (Rational op2)
{
   int commonDenominator = denominator * op2.getDenominator();
   int numerator1 = numerator * op2.getDenominator();
   int numerator2 = op2.getNumerator() * denominator;
   int difference = numerator1 - numerator2;
   return new Rational (difference, commonDenominator);
}

//-----------------------------------------------------------------
//  Multiplies this rational number by the one passed as a
//  parameter.
//-----------------------------------------------------------------
public Rational multiply (Rational op2)
{
   int numer = numerator * op2.getNumerator();
   int denom = denominator * op2.getDenominator();
   return new Rational (numer, denom);
}

//-----------------------------------------------------------------
//  Divides this rational number by the one passed as a parameter
//  by multiplying by the reciprocal of the second rational.
//-----------------------------------------------------------------
public Rational divide (Rational op2)
{
   return multiply (op2.reciprocal());
}

//-----------------------------------------------------------------
//  Returns the reciprocal of this rational number.
//-----------------------------------------------------------------
public Rational reciprocal ()
{
   return new Rational (denominator, numerator);
}

//-----------------------------------------------------------------
//  Returns the numerator of this rational number.
//-----------------------------------------------------------------
public int getNumerator ()
{
   return numerator;
}

//-----------------------------------------------------------------
//  Returns the denominator of this rational number.
//-----------------------------------------------------------------
public int getDenominator ()
{
   return denominator;
}
```

```
  //-----------------------------------------------------------------
  //  Determines if this rational number is equal to the one passed
  //  as a parameter.  Assumes they are both reduced.
  //-----------------------------------------------------------------
  public boolean equals (Rational op2)
  {
    return ( numerator == op2.getNumerator() &&
             denominator == op2.getDenominator() );
  }

  //-----------------------------------------------------------------
  //  Returns this rational number as a string.
  //-----------------------------------------------------------------
  public String toString ()
  {
    String result;
    if (numerator == 0)
      result = "0";
    else
      if (denominator == 1)
        result = numerator + "";
      else
        result = numerator + "/" + denominator;
    return result;
  }

  //-----------------------------------------------------------------
  //  Reduces this rational number by dividing both the numerator
  //  and the denominator by their greatest common divisor.
  //-----------------------------------------------------------------
  private void reduce ()
  {
    if (numerator != 0)
    {
      int common = gcd (Math.abs(numerator), denominator);

      numerator = numerator / common;
      denominator = denominator / common;
    }
  }

  //-----------------------------------------------------------------
  //  Computes and returns the greatest common divisor of the two
  //  positive parameters. Uses Euclid's algorithm.
  //-----------------------------------------------------------------
  private int gcd (int num1, int num2)
  {
    while (num1 != num2)
      if (num1 > num2)
        num1 = num1 - num2;
      else
        num2 = num2 - num1;

    return num1;
  }
}
```

```
//********************************************************************
//  SnakeEyes.java       Author: Lewis and Loftus
//
//  Demonstrates the use of a class with overloaded constructors.
//********************************************************************

import Die;

public class SnakeEyes
{
   //-----------------------------------------------------------------
   //  Creates two die objects, then rolls both dice a set number of
   //  times, counting the number of snake eyes that occur.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final int ROLLS = 500;
      int snakeEyes = 0, num1, num2;

      Die die1 = new Die();    // creates a six-sided die
      Die die2 = new Die(20);  // creates a twenty-sided die

      for (int roll = 1; roll <= ROLLS; roll++)
      {
         num1 = die1.roll();
         num2 = die2.roll();

         if (num1 == 1 && num2 == 1)  // check for snake eyes
            snakeEyes++;
      }

      System.out.println ("Number of rolls: " + ROLLS);
      System.out.println ("Number of snake eyes: " + snakeEyes);
      System.out.println ("Ratio: " + (float)snakeEyes/ROLLS);
   }
}
```

```
//********************************************************************
//  PigLatin.java       Author: Lewis and Loftus
//  Driver to exercise the PigLatinTranslator class.
//********************************************************************
import PigLatinTranslator;
import cs1.Keyboard;
public class PigLatin
{
   //-----------------------------------------------------------------
   //  Reads sentences and translates them into Pig Latin.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String sentence, result, another;
      PigLatinTranslator translator = new PigLatinTranslator();
      do {
         System.out.println ();
         System.out.println ("Enter a sentence (no punctuation):");
         sentence = Keyboard.readString();

         System.out.println ();
         result = translator.translate (sentence);
         System.out.println ("That sentence in Pig Latin is:");
         System.out.println (result);

         System.out.println ();
         System.out.print ("Translate another sentence (y/n)? ");
         another = Keyboard.readString();
      } while (another.equalsIgnoreCase("y"));
   }
}

//********************************************************************
//  PigLatinTranslator.java       Author: Lewis and Loftus
//  Represents a translation system from English to Pig Latin.
//  Demonstrates method decomposition and the use of StringTokenizer.
//********************************************************************
import java.util.StringTokenizer;
public class PigLatinTranslator
{
   //-----------------------------------------------------------------
   //  Translates a sentence of words into Pig Latin.
   //-----------------------------------------------------------------
   public String translate (String sentence)
   {
      String result = "";
      sentence = sentence.toLowerCase();
      StringTokenizer tokenizer = new StringTokenizer (sentence);

      while (tokenizer.hasMoreTokens())
      {
         result += translateWord (tokenizer.nextToken());
         result += " ";
      }
      return result;
   }
```

```
  //-----------------------------------------------------------------
  //  Translates one word into Pig Latin. If the word begins with a
  //  vowel, the suffix "yay" is appended to the word.  Otherwise,
  //  the first letter or two are moved to the end of the word,
  //  and "ay" is appended.
  //-----------------------------------------------------------------
  private String translateWord (String word)
  {
    String result = "";

    if (beginsWithVowel(word))
      result = word + "yay";
    else
      if (beginsWithPrefix(word))
        result = word.substring(2) + word.substring(0,2) + "ay";
      else
        result = word.substring(1) + word.charAt(0) + "ay";

    return result;
  }

  //-----------------------------------------------------------------
  //  Determines if the specified word begins with a vowel.
  //-----------------------------------------------------------------
  private boolean beginsWithVowel (String word)
  {
    String vowels = "aeiouAEIOU";

    char letter = word.charAt(0);

    return (vowels.indexOf(letter) != -1);
  }

  //-----------------------------------------------------------------
  //  Determines if the specified word begins with a particular
  //  two-character prefix.
  //-----------------------------------------------------------------
  private boolean beginsWithPrefix (String str)
  {
    return ( str.startsWith ("bl") || str.startsWith ("pl") ||
             str.startsWith ("br") || str.startsWith ("pr") ||
             str.startsWith ("ch") || str.startsWith ("sh") ||
             str.startsWith ("cl") || str.startsWith ("sl") ||
             str.startsWith ("cr") || str.startsWith ("sp") ||
             str.startsWith ("dr") || str.startsWith ("sr") ||
             str.startsWith ("fl") || str.startsWith ("st") ||
             str.startsWith ("fr") || str.startsWith ("th") ||
             str.startsWith ("gl") || str.startsWith ("tr") ||
             str.startsWith ("gr") || str.startsWith ("wh") ||
             str.startsWith ("kl") || str.startsWith ("wr") ||
             str.startsWith ("ph") );
  }
}
```

```
//********************************************************************
//  LineUp.java       Author: Lewis and Loftus
//  Demonstrates the use of a graphical object.
//********************************************************************

import StickFigure;
import java.applet.Applet;
import java.awt.*;

public class LineUp extends Applet
{
   private final int APPLET_WIDTH = 400;
   private final int APPLET_HEIGHT = 150;
   private final int HEIGHT_MIN = 100;
   private final int VARIANCE = 30;

   private StickFigure figure1, figure2, figure3, figure4;

   //-----------------------------------------------------------------
   //  Creates several stick figures with varying characteristics.
   //-----------------------------------------------------------------
   public void init ()
   {
      int h1, h2, h3, h4;  // heights of stick figures


      h1 = (int) (Math.random() * VARIANCE) + HEIGHT_MIN;
      h2 = (int) (Math.random() * VARIANCE) + HEIGHT_MIN;
      h3 = (int) (Math.random() * VARIANCE) + HEIGHT_MIN;
      h4 = (int) (Math.random() * VARIANCE) + HEIGHT_MIN;

      figure1 = new StickFigure (100, 150, Color.red, h1);
      figure2 = new StickFigure (150, 150, Color.cyan, h2);
      figure3 = new StickFigure (200, 150, Color.green, h3);
      figure4 = new StickFigure (250, 150, Color.yellow, h4);

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

   //-----------------------------------------------------------------
   //  Paints the stick figures on the applet.
   //-----------------------------------------------------------------
   public void paint (Graphics page)
   {
      figure1.draw (page);
      figure2.draw (page);
      figure3.draw (page);
      figure4.draw (page);
   }
}
```

```
<! LineUp.html>
<HTML>
<HEAD>
<TITLE>The Line Up Applet</TITLE>
</HEAD>
<BODY>
<center>
<H3>The Line Up Applet:</H3>
<APPLET CODE="LineUp.class" WIDTH=400 HEIGHT=150>
</APPLET>
<HR>
</center>
</BODY>
</HTML>

//********************************************************************
//  StickFigure.java       Author: Lewis and Loftus
//
//  Represents a graphical stick figure.
//********************************************************************
import java.awt.*;

public class StickFigure
{
   private int baseX;     // center of figure
   private int baseY;     // floor (bottom of feet)
   private Color color;   // color of stick figure
   private int height;    // height of stick figure

   //-----------------------------------------------------------------
   //  Sets up the stick figure's primary attributes.
   //-----------------------------------------------------------------
   public StickFigure (int center, int bottom, Color shade, int size)
   {
      baseX = center;
      baseY = bottom;
      color = shade;
      height = size;
   }

   //-----------------------------------------------------------------
   //  Draws this figure relative to baseX, baseY, and height.
   //-----------------------------------------------------------------
   public void draw (Graphics page)
   {
      int top = baseY - height;  // top of head

      page.setColor (color);
      page.drawOval (baseX-10, top, 20, 20);                          // head
            page.drawLine (baseX, top+20, baseX, baseY-30);           // trunk
      page.drawLine (baseX, baseY-30, baseX-15, baseY);               // legs
      page.drawLine (baseX, baseY-30, baseX+15, baseY);
      page.drawLine (baseX, baseY-70, baseX-25, baseY-70);  // arms
      page.drawLine (baseX, baseY-70, baseX+20, baseY-85);
   }
}
```