Chapter 5: Conditionals and Loops Lab Exercises

Topics	**Lab Exercises**
Boolean expressions	PreLab Exercises
The **if** statement	Computing a Raise
The **while** statement	PreLab Exercises
	Counting and Looping
	Powers of 2 Factorials
	A Guessing Game
Iterators & Reading Text Files	Baseball Statistics
ArrayList Class	A Shopping Cart Using the ArrayList Class
Determining Event Sources	Vote Counter, Revisited
Checkboxes & Radio Buttons	Adding Buttons to StyleOptions.java

Prelab Exercises
Sections 5.1-5.3

1. Rewrite each condition below in valid Java syntax (give a boolean expression):
   a. x > y > z
   b. x and y are both less than 0
   c. neither x nor y is less than 0
   d. x is equal to y but not equal to z

2. Suppose *gpa* is a variable containing the grade point average of a student. Suppose the goal of a program is to let a student know if he/she made the Dean's list (the gpa must be 3.5 or above). Write an *if... else...* statement that prints out the appropriate message (either "Congratulations—you made the Dean's List" or "Sorry you didn't make the Dean's List").

3. Complete the following program to determine the raise and new salary for an employee by adding if ... else statements to compute the raise. The input to the program includes the current annual salary for the employee and a number indicating the performance rating (1=excellent, 2=good, and 3=poor). An employee with a rating of 1 will receive a 6% raise, an employee with a rating of 2 will receive a 4% raise, and one with a rating of 3 will receive a 1.5% raise.

```
// ***************************************************************
//   Salary.java
//   Computes the raise and new salary for an employee
// ***************************************************************
import java.util.Scanner;

public class Salary
{
   public static void main (String[] args)
   {
      double currentSalary;  // current annual salary
      double rating;         // performance rating
      double raise;         // dollar amount of the raise

      Scanner scan = new Scanner(System.in);

      // Get the current salary and performance rating
      System.out.print ("Enter the current salary: ");
      currentSalary = scan.nextDouble();
      System.out.print ("Enter the performance rating: ");
      rating = scan.nextDouble();

      // Compute the raise -- Use if ... else ...

      // Print the results
      System.out.println ("Amount of your raise: $" + raise);
      System.out.println ("Your new salary: $" + currentSalary + raise);

   }
}
```

Computing A Raise

File *Salary.java* contains most of a program that takes as input an employee's salary and a rating of the employee's performance and computes the raise for the employee. This is similar to question #3 in the pre-lab, except that the performance rating here is being entered as a String—the three possible ratings are "Excellent", "Good", and "Poor". As in the pre-lab, an employee who is rated excellent will receive a 6% raise, one rated good will receive a 4% raise, and one rated poor will receive a 1.5% raise.

Add the *if... else...* statements to program Salary to make it run as described above. Note that you will have to use the *equals* method of the String class (not the relational operator ==) to compare two strings (see Section 5.3, Comparing Data).

```
// ***************************************************************
//   Salary.java
//
//   Computes the amount of a raise and the new
//   salary for an employee.  The current salary
//   and a performance rating (a String: "Excellent",
//   "Good" or "Poor") are input.
// ***************************************************************

import java.util.Scanner;
import java.text.NumberFormat;

public class Salary
{
   public static void main (String[] args)
   {
      double currentSalary;   // employee's current  salary
      double raise;           // amount of the raise
      double newSalary;       // new salary for the employee
      String rating;          // performance rating

      Scanner scan = new Scanner(System.in);

      System.out.print ("Enter the current salary: ");
      currentSalary = scan.nextDouble();
      System.out.print ("Enter the performance rating (Excellent, Good, or Poor): ");
      rating = scan.next();

      // Compute the raise using if ...

      newSalary = currentSalary + raise;

      // Print the results
      NumberFormat money = NumberFormat.getCurrencyInstance();
      System.out.println();
      System.out.println("Current Salary:         " + money.format(currentSalary));
      System.out.println("Amount of your raise: " + money.format(raise));
      System.out.println( "Your new salary:       " + money. format (newSalary) );
      System.out.println();
   }
}
```

Prelab Exercises
Section 5.4

In a while loop, execution of a set of statements (the *body* of the loop) continues until the boolean expression controlling the loop (the *condition*) becomes false. As for an if statement, the condition must be enclosed in parentheses. For example, the loop below prints the numbers from 1 to to LIMIT:

```
final int LIMIT = 100;         // setup
int count = 1;

while (count <= LIMIT)         // condition
{                              // body
  System.out.println(count);   //  -- perform task
  count = count + 1;           //  -- update condition
}
```

There are three parts to a loop:

- The *setup*, or *initialization*. This comes before the actual loop, and is where variables are initialized in preparation for the first time through the loop.
- The *condition*, which is the boolean expression that controls the loop. This expression is evaluated each time through the loop. If it evaluates to true, the body of the loop is executed, and then the condition is evaluated again; if it evaluates to false, the loop terminates.
- The *body* of the loop. The body typically needs to do two things:
  - Do some work toward the task that the loop is trying to accomplish. This might involve printing, calculation, input and output, method calls—this code can be arbitrarily complex.
  - Update the condition. Something has to happen inside the loop so that the condition will eventually be false—otherwise the loop will go on forever (an *infinite* loop). This code can also be complex, but often it simply involves incrementing a counter or reading in a new value.

  Sometimes doing the work and updating the condition are related. For example, in the loop above, the print statement is doing work, while the statement that increments count is both doing work (since the loop's task is to print the values of count) and updating the condition (since the loop stops when count hits a certain value).

The loop above is an example of a *count-controlled* loop, that is, a loop that contains a counter (a variable that increases or decreases by a fixed value—usually 1—each time through the loop) and that stops when the counter reaches a certain value. Not all loops with counters are count-controlled; consider the example below, which determines how many even numbers must be added together, starting at 2, to reach or exceed a given limit.

```
final int LIMIT = 16; TRACE
int count = 1;                 sum     nextVal     count
int sum = 0;                   ---     -------     -----
int nextVal = 2;

while (sum < LIMIT)
{
  sum = sum + nextVal;
  nextVal = nextVal + 2;
  count = count + 1;
}

System.out.println("Had to add together " + (count-1) + " even numbers " +
                   "to reach value " + LIMIT + ".  Sum is " + sum);
```

Note that although this loop counts how many times the body is executed, the condition does not depend on the value of count.

Not all loops have counters. For example, if the task in the loop above were simply to add together even numbers until the sum reached a certain limit and then print the sum (as opposed to printing the number of things added together), there would be no need for the counter. Similarly, the loop below sums integers input by the user and prints the sum; it contains no counter.

```
int sum = 0;                                        //setup
String keepGoing = "y";
int nextVal;

while (keepGoing.equals("y") || keepGoing.equals("Y"))
{
  System.out.print("Enter the next integer: ");       //do work
  nextVal = scan.nextInt();
  sum = sum + nextVal;

  System.out.println("Type y or Y to keep going");    //update condition
  keepGoing = scan.next();
}
System.out.println("The sum of your integers is " + sum);
```

Exercises

1. In the first loop above, the println statement comes before the value of count is incremented. What would happen if you reversed the order of these statements so that count was incremented before its value was printed? Would the loop still print the same values? Explain.

2. Consider the second loop above.
   a. Trace this loop, that is, in the table next to the code show values for variables nextVal, sum and count at each iteration. Then show what the code prints.
   b. Note that when the loop terminates, the number of even numbers added together before reaching the limit is count-1, not count. How could you modify the code so that when the loop terminates, the number of things added together is simply count?

3. Write a while loop that will print “I love computer science!!” 100 times. Is this loop count-controlled?

4. Add a counter to the third example loop above (the one that reads and sums integers input by the user). After the loop, print the number of integers read as well as the sum. Just note your changes on the example code. Is your loop now count-controlled?

5. The code below is supposed to print the integers from 10 to 1 backwards. What is wrong with it? (Hint: there are two problems!) Correct the code so it does the right thing.

```
count = 10;
while (count >= 0)
{
 System.out.println(count);
 count = count + 1;
}
```

Counting and Looping

The program in *LoveCS.java* prints "I love Computer Science!!" 10 times. Copy it to your directory and compile and run it to see how it works. Then modify it as follows:

```
// ****************************************************************
//   LoveCS.java
//
//   Use a while loop to print many messages declaring your
//   passion for computer science
// ****************************************************************

public class LoveCS
{
    public static void main(String[] args)
    {
      final int LIMIT = 10;

      int count = 1;

      while (count <= LIMIT){
          System.out.println("I love Computer Science!!");
          count++;

      }
    }
}
```

1. Instead of using constant LIMIT, ask the user how many times the message should be printed. You will need to declare a variable to store the user's response and use that variable to control the loop. (Remember that all caps is used only for constants!)
2. Number each line in the output, and add a message at the end of the loop that says how many times the message was printed. So if the user enters 3, your program should print this:

```
1 I love Computer Science!!
2 I love Computer Science!!
3 I love Computer Science!!
Printed this message 3 times.
```

3. If the message is printed N times, compute and print the sum of the numbers from 1 to N. So for the example above, the last line would now read:

```
Printed this message 3 times. The sum of the numbers from  1 to 3 is 6.
```

   Note that you will need to add a variable to hold the sum.

Powers of 2

File *PowersOf2.java* contains a skeleton of a program to read in an integer from the user and print out that many powers of 2, starting with 20.

1. Using the comments as a guide, complete the program so that it prints out the number of powers of 2 that the user requests. **Do not use Math.pow to compute the powers of 2!** Instead, compute each power from the previous one (how do you get 2^n from 2^{n-1}?). For example, if the user enters 4, your program should print this:

```
Here are the first 4 powers of 2:
1
2
4
8
```

2. Modify the program so that instead of just printing the powers, you print which power each is, e.g.:

```
Here are the first 4 powers of 2:

2^0 = 1
2^1 = 2
2^2 = 4
2^3 = 8
```

```
// ***************************************************************
//   PowersOf2.java
//
//   Print out as many powers of 2 as the user requests
//
// ***************************************************************
import java.util.Scanner;

public class PowersOf2
{
    public static void main(String[] args)
    {
        int numPowersOf2;        //How many powers of 2 to compute
        int nextPowerOf2 = 1;    //Current power of  2
        int exponent;            //Exponent for current power of 2 -- this
                                 //also serves as a counter for the loop Scanner
        scan = new Scanner(System.in);

        System.out.println("How many powers of 2 would you like printed?");
        numPowersOf2 = scan.nextInt();

        //print a message saying how many powers of 2 will be printed
        //initialize exponent -- the first thing printed is 2 to the what?

        while (  )
        {
            //print out current power of 2

            //find next power of 2 -- how do you get this from the last one?

            //increment exponent
        }
    }
}
```

Factorials

The *factorial* of n (written n!) is the product of the integers between 1 and n. Thus 4! = 1*2*3*4 = 24. By definition, 0! = 1. Factorial is not defined for negative numbers.

1. Write a program that asks the user for a non-negative integer and computes and prints the factorial of that integer. You'll need a while loop to do most of the work—this is a lot like computing a sum, but it's a product instead. And you'll need to think about what should happen if the user enters 0.

2. Now modify your program so that it checks to see if the user entered a negative number. If so, the program should print a message saying that a nonnegative number is required and ask the user the enter another number. The program should keep doing this until the user enters a nonnegative number, after which it should compute the factorial of that number. **Hint:** you will need another while loop **before** the loop that computes the factorial. You should not need to change any of the code that computes the factorial!

A Guessing Game

File *Guess.java* contains a skeleton for a program to play a guessing game with the user. The program should randomly generate an integer between 1 and 10, then ask the user to try to guess the number. If the user guesses incorrectly, the program should ask them to try again until the guess is correct; when the guess is correct, the program should print a congratulatory message.

1. Using the comments as a guide, complete the program so that it plays the game as described above.
2. Modify the program so that if the guess is wrong, the program says whether it is too high or too low. You will need an if statement (inside your loop) to do this.
3. Now add code to count how many guesses it takes the user to get the number, and print this number at the end with the congratulatory message.
4. Finally, count how many of the guesses are too high and how many are too low. Print these values, along with the total number of guesses, when the user finally guesses correctly.

```
// ************************************************************
//   Guess.java
//
//   Play a game where the user guesses a number from 1 to 10
//
// ************************************************************
import java.util.Scanner;
import java.util.Random;

public class Guess
{
    public static void main(String[] args)
    {
      int numToGuess;       //Number the user tries to guess
      int guess;            //The user's guess

      Scanner scan = new Scanner(System.in);
      Random generator = new Random();

      //randomly generate the  number to guess

      //print message asking user to enter a guess

      //read in guess

      while (  )  //keep going as long as the guess is wrong
        {
          //print message saying guess is wrong
          //get another guess from the user
      }

      //print message saying guess is right

    }

}
```

Baseball Statistics

The local Kids' League coach keeps some of the baseball team statistics in a text file organized as follows: each line of the file contains the name of the player followed by a list of symbols indicating what happened on each at bat for the player. The letter h indicates a hit, o an out, w a walk, and s a sacrifice fly. Each item on the line is separated by a comma. There are no blank spaces except in the player name. So, for example the file could look as follows:

```
Sam Slugger,h,h,o,s,w,w,h,w,o,o,o,h,s
Jill Jenks,o,o,s,h,h,o,o
Will Jones,o,o,w,h,o,o,o,o,w,o,o
```

The file *BaseballStats.java* contains the skeleton of a program thats reads and processes a file in this format. Study the program and note that three Scanner objects are declared.

- One scanner (*scan*) is used to read in a file name from standard input.
- The file name is then used to create a scanner (*fileScan*) to operate on that file.
- A third scanner (*lineScan*) will be used to parse each line in the file.

Also note that the main method throws an IOException. This is needed in case there is a problem opening the file.

Complete the program as follows:

1. First add a while loop that reads each line in the file and prints out each part (name, then each at bat, without the commas) in a way similar to the URLDissector program in Listing 5.10 of the text. In particular inside the loop you need to
   a. read the next line from the file
   b. create a comma delimited scanner (lineScan) to parse the line
   c. read and print the name of the player, and finally,
   d. have a loop that prints each at bat code.

2. Compile and run the program to be sure it works.

3. Now modify the inner loop that parses a line in the file so that instead of printing each part it counts (separately) the number of hits, outs, walks, and sacrifices. Each of these summary statistics, as well as the batting average, should be printed for each player. Recall that the batting average is the number of hits divided by the total number of hits and outs.

4. Test the program on the file *stats.dat* and *stats2.dat* .

```
// ****************************************************************
//  BaseballStats.java
//
//  Reads baseball data in from a comma delimited file. Each line
//  of the file contains a name followed by a list of symbols
//  indicating the result of each at bat: h for hit, o for out,
//  w for walk, s for sacrifice.  Statistics are computed and
//  printed for each player.
// ****************************************************************

import java.util.Scanner;
import java.io.*;

public class BaseballStats
{

  //-------------------------------------------------
  //  Reads baseball stats from a file and counts
  //  total hits, outs, walks, and sacrifice flies
  //  for each player.
  //-------------------------------------------------
  public static void main (String[] args) throws IOException
```

```
    {
        Scanner fileScan, lineScan;
        String fileName;

        Scanner scan = new Scanner(System.in);

        System.out.print ("Enter the name of the input file: ");
        fileName = scan.nextLine();
        fileScan = new Scanner(new File(fileName));

        // Read and process each line of the file



    }
}
```

stats.dat

```
Willy Wonk,o,o,h,o,o,o,o,h,w,o,o,o,o,s,h,o,h
Shari Jones,h,o,o,s,s,h,o,o,o,h,o,o,o,o
Barry Bands,h,h,w,o,o,o,w,h,o,o,h,h,o,o,w,w,w,h,o,o
Sally Slugger,o,h,h,o,o,h,h,w
Missy Lots,o,o,s,o,o,w,o,o,o
Joe Jones,o,h,o,o,o,o,h,h,o,o,o,o,w,o,o,o,h,o,h,h
Larry Loop,w,s,o,o,o,h,o,o,h,s,o,o,o,h,h
Sarah Swift,o,o,o,o,h,h,w,o,o,o
Bill Bird,h,o,h,o,h,w,o,o,o,h,s,s,h,o,o,o,o,o,o
Don Daring,o,o,h,h,o,o,h,o,h,o,o,o,o,o,o,h
Jill Jet,o,s,s,h,o,o,h,h,o,o,o,h,o,h,w,o,o,h,h,o
```

stats2.dat

```
Barry Bands,h,h,w,o,o,o,w,h,o,o,h,h,o,o,w,w,w,h,o,o
```

A Shopping Cart Using the ArrayList Class

In this exercise you will implement a shopping cart using the ArrayList class. The file *Item.java* contains the definition of a class named *Item* that models an item one would purchase (this class was used in an earlier lab). An item has a name, price, and quantity (the quantity purchased). The file *Shop.java* is an incomplete program that models shopping.

1. Complete Shop.java as follows:
   a. Declare and instantiate a variable *cart* to be an empty ArrayList.
   b. Fill in the statements in the loop to add an item to the cart and to print the cart contents (using the default *toString* in the ArrayList class). Comments in the code indicate where these statements go.
   c. Compile your program and run it.

2. You should have observed two problems with using the default printing for the cart object: the output doesn't look very good and the total price of the goods in the cart is not computed or printed. Modify the program to correct these problems by replacing the print statement with a loop that does the following:
   a. gets each item from the cart and prints the item
   b. computes the total price of the items in the cart (you need to use the *getPrice* and *getQuantity* methods of the Item class). The total price should be printed after the loop.

3. Compile and run your program.

```
// ***************************************************************
//   Shop.java
//
//   Uses the Item class to create items and add them to a shopping
//   cart stored in an ArrayList.
// ***************************************************************

import java.util.ArrayList;
import java.util.Scanner;

public class Shop
{
   public static void main (String[] args)
   {

      Item item;
      String itemName;
      double itemPrice;
      int quantity;

      Scanner scan = new Scanner(System.in);

      String keepShopping = "y";

      do
         {
            System.out.print ("Enter the name of the item: ");
            itemName = scan.nextLine();

            System.out.print ("Enter the unit price: ");
            itemPrice = scan.nextDouble();

            System.out.print ("Enter the quantity: ");
            quantity = scan.nextInt();

            // *** create a new item and add it to the cart
            // *** print the contents of the cart object using println

            System.out.print ("Continue shopping (y/n)? ");
            keepShopping = scan.nextLine();
         }
      while (keepShopping.equals("y"));

  }
}
```

Vote Counter, Revisited

Chapter 4 had a lab exercise that created a GUI with two buttons representing two candidates (Joe and Sam) in an election or popularity contest. The program computed the number of times each button was pressed. In that exercise two different listeners were used, one for each button. This exercise is a slight modification. Only one listener will be used and its ActionPerformed method will determine which button was pressed.

The files *VoteCounter.java* and *VoteCounterPanel.java* contain slight revisions to the skeleton programs used in the Chapter 4 exercise. Save them to your directory and do the following:

1. Add variables for Sam - a vote counter, a button, and a label.

2. Add the button and label for Sam to the panel; add the listener to the button.

3. Modify the ActionPerformed method of the VoteButtonListener class to determine which button was pressed and update the correct counter. (See the LeftRight example or the Quote example for how to determine the source of an event.)

4. Test your program.

5. Now modify the program to add a message indicating who is winning. To do this you need to instantiate a new label, add it to the panel, and add an if statement in actionPerformed that determines who is winning (also test for ties) and sets the text of the label with the appropriate message.

```
//*********************************************************
// VoteCounter.java
//
// Demonstrates a graphical user interface and event
// listeners to tally votes for two candidates, Joe and Sam.
//*********************************************************

import javax.swing.JFrame;

public class VoteCounter
{
   //----------------------------------------
   // Creates the main program frame.
   //----------------------------------------
   public static void main(String[] args)
   {
       JFrame frame = new JFrame("Vote Counter");
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       frame.getContentPane().add(new VoteCounterPanel());

       frame.pack();
       frame.setVisible(true);
   }
}
```

```
//*********************************************************
// VoteCounterPanel.java
//
// Panel for the GUI that tallies votes for two candidates,
// Joe and Sam.
//*********************************************************

import java.awt.*;
import java.awt.event.*;
import javax.swing. *;

public class VoteCounterPanel extends JPanel
{
    private int votesForJoe;
    private JButton joe;
    private JLabel labelJoe;

    //----------------------------------------------------
    // Constructor: Sets up the GUI.
    //----------------------------------------------------
    public VoteCounterPanel()
    {
       votesForJoe = 0;

       joe = new JButton("Vote for Joe");
       joe.addActionListener(new VoteButtonListener());

       labelJoe = new JLabel("Votes for Joe: " + votesForJoe);

       add(joe);
       add(labelJoe);

       setPreferredSize(new Dimension(300, 40));
       setBackground(Color.cyan);
    }

    //***************************************************
    // Represents a listener for button push (action) events
    //***************************************************
    private class VoteButtonListener implements ActionListener
    {

       //-----------------------------------------------
       // Updates the appropriate vote counter when a
       // button is pushed for one of the candidates.
       //-----------------------------------------------
       public void actionPerformed(ActionEvent event)
       {
          votesForJoe++;
          labelJoe.setText("Votes for Joe: " + votesForJoe);
       }
    }
}
```

Adding Buttons to *StyleOptions.java*

The files *StyleOptions.java* and *StyleOptionsPanel.java* are from Listings "5.14 and 5.15" of the text (with a couple of slight changes—an instance variable *fontSize* is used rather than the literal 36 for font size and the variable *style* is an instance variable rather than local to the itemStateChanged method). The program demonstrates checkboxes and ItemListeners. In this exercise you will add a set of three radio buttons to let the user choose among three font sizes. The method of adding the radio buttons will be very similar to that in the QuoteOptionsPanel class (Listing 5.17 of the text). Before modifying the program compile and run the current version to see how it works and study the QuoteOptionsPanel example.

Do the following to add the radio buttons to the panel:

1. Declare three objects *small*, *medium*, and *large* of type JRadioButton.

2. Instantiate the button objects labeling them "Small Font," "Medium Font," "Large Font." Initialize the large font button to true. Set the background color of the buttons to cyan.

3. Instantiate a button group object and add the buttons to it.

4. Radio buttons produce action events so you need an ActionListener to listen for radio button clicks. The code you need to add to *actionPerformed* will be similar to that in the QuoteListener in Listing 5.17. In this case you need to set the *fontSize* variable (use 12 for small, 24 for medium, and 36 for large) in the if statement, then call the *setFont* method to set the font for the *saying* object. (Note: Instead of adding an ActionListener you could use the current ItemListener and add code to check to see if the source of the event was a radio button.)

5. In *StyleOptionsPanel()* add the ItemListener object to each button and add each button to the panel.

6. Compile and run the program. Note that as the font size changes the checkboxes and buttons re-arrange themselves in the panel. You will learn how to control layout later in the course.

```
//********************************************************************
//  StyleOptions.java  Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//********************************************************************

import javax.swing.JFrame;

public class StyleOptions
{
   //-----------------------------------------------------------------
   //  Creates and presents the program frame.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
     JFrame frame = new JFrame ("Style Options");
     frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

     StyleOptionsPanel panel = new StyleOptionsPanel();
     frame.getContentPane().add (panel);

     frame.pack();
     frame.setVisible(true);
  }
}
```

```
//**********************************************************
//  StyleOptionsPanel.java      Author: Lewis/Loftus
//
//  Demonstrates the use of check boxes.
//**********************************************************

import javax. swing. *;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel
{
   private int fontSize = 36;
   private int style = Font.PLAIN;
   private JLabel saying;
   private JCheckBox bold, italic;

    //-------------------------------------------------------------
    //  Sets up a panel with a label and some check boxes that
    //  control the style of the label's font.
    //-------------------------------------------------------------
    public StyleOptionsPanel()
    {
      saying = new JLabel ("Say it with style!");
      saying.setFont (new Font ("Helvetica", style, fontSize));

      bold = new JCheckBox ("Bold");
      bold.setBackground (Color.cyan);
      italic = new JCheckBox ("Italic");
      italic.setBackground (Color.cyan);

      StyleListener listener = new StyleListener ();
      bold.addItemListener (listener);
      italic.addItemListener (listener) ;

      add (saying);
      add (bold);
      add (italic);

      setBackground (Color.cyan);
      setPreferredSize (new Dimension(300, 100));
    }

   //**********************************************************
   //  Represents the listener for both check boxes.
  //**********************************************************
  private class StyleListener implements ItemListener
  {
      //-------------------------------------------------------
      //  Updates the style of the label font style.
      //-------------------------------------------------------
      public void itemStateChanged (ItemEvent event)
      {
        style = Font.PLAIN;

        if (bold.isSelected())
            style = Font.BOLD;

        if (italic.isSelected())
            style += Font.ITALIC;
        saying.setFont (new Font ("Helvetica", style, fontSize));
      }
    }
}
```