THỰC HÀNH CÔNG CỤ VÀ MÔI TRƯỜNG VÀ LẬP TRÌNH 2

TS. Võ Phương Bình – Email: binhvp@dlu.edu.vn
Information Technology Faculty - Dalat University
Website: http://it.dlu.edu.vn/ivp-lab

LAB 4.2 (4 tiết): Thực hành ứng dụng GUI(tiếp)

A. Mục tiêu:

- Hiểu biết thiết kế giao diện với môi trường đồ họa nâng cao.
- Xử lý các sự kiện trên môi trường GUI nâng cao

B. Kết quả sau khi hoàn thành:

- Sử dụng được các thành phần thiết kế để có thể tạo ra các giao diện.
- Xây dựng thành công các ứng dụng.
- Xử lý được các sự kiện trong Java.

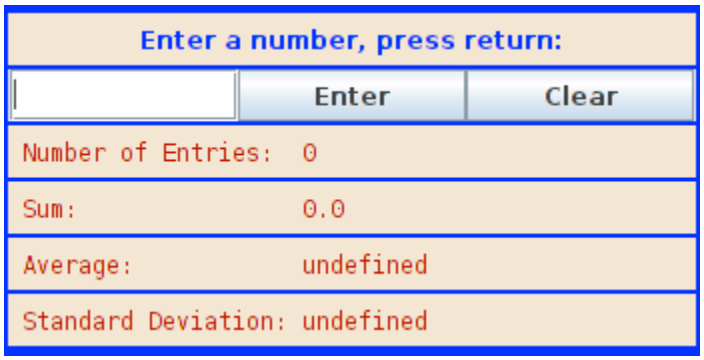
C. Luyện tập:

- Tìm hiểu các kiến thức liên quan đến Frame, thêm Component vào Frame và xử lý các sự kiện của các Component.
- Xem lại slide bài học các kiến thức liên quan đến thiết kế GUI.

D. Bài tập.

Exercise 4.1:

Write a GUI program that uses the *StatCalc* class to compute and display statistics of numbers entered by the user. The panel will have an instance variable of type *StatCalc* that does the computations. The panel should include a *JTextField* where the user enters a number. It should have four labels that display four statistics for the numbers that have been entered: the number of numbers, the sum, the mean, and the standard deviation. Every time the user enters a new number, the statistics displayed on the labels should change. The user enters a number by typing it into the *JTextField* and pressing return. There should be a "Clear" button that clears out all the data. This means creating a new *StatCalc* object and resetting the displays on the labels. My panel also has an "Enter" button that does the same thing as pressing the return key in the *JTextField*. (Recall that a *JTextField* generates an *ActionEvent* when the user presses return, so your panel should register itself to listen for *ActionEvents* from the *JTextField* as well as the buttons.) Here is a picture of my solution to this problem:



Discussion

In my solution, I used four labels to display results and another label at the top of the panel to display a message to the user. Aside from these labels, one row of the panel holds three other components: a *JTextField* and two *JButtons*. The panel uses a *GridLayout* with six rows. Five of the rows hold *JLabels*. The other row contains a *JPanel* that holds the *JTextField* and *JButtons*. This *JPanel* uses a *GridLayout* with three columns and just one row.

The constructor creates and lays out the components. Since I want the program to look nice, I set a background color and a foreground color for most of the components. I set the labels to be opaque, to make sure that the background of each label will actually be filled in with the label's background color. After looking at my first attempt, I decided to use a Monospaced font for the display labels. In a Monospaced font, all the characters are the same size. This makes it possible to line up the output values vertically by putting the same number of characters in each label. To make it easy to play with the colors and fonts, I declared three named constants

```
final static Color labelBG = new Color(240,225,200);  // beige
final static Color labelFG = new Color(180,0,0);  // dark red
final static Font labelFont = new Font("Monospaced", Font.PLAIN,
12);
```

I could then make one of the labels, such as `countLabel`, with the commands:

```
countLabel = new JLabel("Number of Entries:   0");
countLabel.setBackground(labelBG);
countLabel.setForeground(labelFG);
countLabel.setOpaque(true);
countLabel.setFont(labelFont);
```

However, since there are four labels to create, I wrote a subroutine to create a display label to show a given string:

```
/**
 * A utility routine for creating the labels that are used
 * for display.  This routine is called by the constructor.
 * @param text The text to show on the label.
 */
private JLabel makeLabel(String text) {
   JLabel label = new JLabel(text);
   label.setBackground(labelBG);
   label.setForeground(labelFG);
   label.setOpaque(true);
   label.setFont(labelFont);
   return label;
}
```

Then in the constructor, the labels can be created with four lines, instead of 16:

```
countLabel =   makeLabel(" Number of Entries:  0");
sumLabel =     makeLabel(" Sum:                0.0");
meanLabel =    makeLabel(" Average:            undefined");
standevLabel = makeLabel(" Standard Deviation: undefined");
```

Utility routines like `makeLabel()` are very commonly used when there are a lot of similar components to create. Note that when the labels are first created, the text on the labels is appropriate for a dataset that contains zero elements. In particular, if there are no data, the average and standard deviation are undefined.

The panel registers itself to listen for action events from the *JTextField* and from the *JButtons*. In the `actionPerformed()` method, the function `evt.getSource()` is called to find the *Object* that generated the event. This will be either the `numberInput` box, the `enterButton`, or the `clearButton`. The source of the event is checked to decide how to respond. (This is an alternative to checking the event's action command.)

If the user clicked the "Clear" button, the response is to create a new *StatCalc* object and to reset the display labels to reflect the fact that there is no data in the dataset. It's important to understand the effect of the command `"stats = new StatCalc();"`. The panel will continue to use the same *StatCalc* **variable**, `stats`. However, now the variable refers to a new *StatCalc* **object**. The new object does not yet have any data in its dataset. The next time the user enters a number, the dataset will get its first value. Always keep in mind the difference between variables and objects. Also, keep in mind that you have to think in terms of changing the state of the panel in response to events. I change the panel's state by starting to use a new *StatCalc* object, and the display labels are changed to keep them consistent with the new state.

When the user clicks the "Enter" button or presses return in the *JTextField*, we have to get the user's input and add it to the *StatCalc* object. This will cause the values of the four statistics to change. We have to change the display labels to show the new values. The code for getting the user's number from the input box includes a check to make sure that the user's input is a legal number. If the input is not legal, then I show an error message in the *JLabel* named `message` and return from the `actionPerformed()` method without entering any new data:

```
double num;  // The user's number.
try {
   num = Double.parseDouble(numberInput.getText());
}
catch (NumberFormatException e) {
       // The user's entry is not a legal number.
       // Put an error message in the message label
       // and return without entering a number.
   message.setText("\"" + numberInput.getText() +
                       "\" is not a legal number.");
   numberInput.selectAll();
   numberInput.requestFocus();
   return;
}
```

The commands "`numberInput.selectAll();`" and "`numberInput.requestFocus();`" are there as a convenience for the user. The first command, which was not covered in this chapter, selects all the text in the number input box. The second command gives the input focus to the input box. That way, the user can just start typing the next number, without having to click on the input box or erase the content of the box. (Since the contents of the box are selected, they will disappear automatically when the user starts typing, to be replaced with the new input. A surprising number of people have never learned that text selections work this way.)

Once we have the user's number, the command "`stats.enter(num);`" adds the number `num` into the dataset. The statistics about the data set can be obtained by calling the functions `stats.getCount()`, `stats.getSum()`, `stats.getMean()`, and `stats.getStandardDeviation()`. This information can be found by reading the *source code* for the *StatCalc* class. These values are used on the labels that display the statistics. For example,

```
countLabel.setText(" Number of Entries:  " + stats.getCount());
```

Returning to the topic of variables versus objects, a common novice mistake would be to try to change the text that is displayed on the label by saying

```
countLabel = new JLabel(" Number of Entries:  " +
stats.getCount());  // WRONG
```

since that type of command is what was used to set the text in the first place. But this statement has no effect on what is displayed on the screen. The reason why this doesn't work is instructive: The assignment command creates a new JLabel and sets `countLabel` to refer to that new object. The new label does indeed have the desired text. But the new label has nothing to do with what's on the screen! The original label was created in the constructor and added to the panel. The panel has a reference to the original label, and that original label continues to appear on the screen even if `countLabel` now refers to a new label. The correct command, using `countLabel.setText()`, **modifies the original label**, which is shown on the screen.

As a point of interest, I will mention another technique that has not been covered elsewhere in the book. You might want to limit the number of decimal places that are displayed in the statistics. You know how to do this in output to the command line, using `System.out.printf` or `TextIO.putf`, but we are not outputting the data here, just putting it into a String. The *String* class has a `static` method `String.format()` that solves this problem. `String.format(fmtString,val,val,val,...)` works similarly to `System.out.prinf(fmtString,val,val,val,...)`, but instead of producing output, `String.format` just creates a string and returns it. To set the text of the label that displays the average, for example, you Might say:

```
meanLabel.setText( String.format(" Average:            %1.9g",
stats.getMean()) );
```

I don't do this in my solution, but `String.format` can be very useful, especially in GUI programs.

The Solution

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

/**
 * In this panel, the user enters numbers in a text field box.
 * After entering each number, the user presses return (or clicks
 * on a button).  Some statistics are displayed about all the
 * numbers that the user has entered.
 */
public class StatCalcGUI extends JPanel implements ActionListener {

    /**
     * A main routine allows this class to be run as an application.
     */
    public static void main(String[] args) {
```

```
        JFrame window = new JFrame("Stat Calc");
        StatCalcGUI content = new StatCalcGUI();
        window.setContentPane(content);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setLocation(120,70);
        window.setSize(350,200);
        window.setVisible(true);
    }

    //-----------------------------------------------------------------
--

    final static Color labelBG = new Color(240,225,200);  // For creating
labels
    final static Color labelFG = new Color(180,0,0);
    final static Font labelFont = new Font("Monospaced", Font.PLAIN, 12);

    private JLabel countLabel;    // A label for displaying the number of
numbers.
    private JLabel sumLabel;      // A label for displaying the sum of
the numbers.
    private JLabel meanLabel;     // A label for displaying the average.
    private JLabel standevLabel;  // A label for displaying the standard
deviation.

    private JLabel message;  // A message at the top of the panel.  It
will
                             //   show an error message if the user's
input is
                             //   not a legal number.  Otherwise, it just
tells
                             //   the user to enter a number and press
return.

    private JButton enterButton;   // A button the user can press to
enter a number.
                                   //    This is an alternative to
pressing return.
    private JButton clearButton;   // A button that clears all the data
that the
                                   //    user has entered.

    private JTextField numberInput;  // The input box where the user
enters numbers.

    private StatCalc stats;  // An object that keeps track of the
statistics
                             //   for all the numbers that have been
entered.

    /**
     * The constructor creates the objects used by the panel.  The panel
     * will listen for action events from the buttons and from the text
     * field.  (A JTextField generates an ActionEvent when the user
presses
     * return while typing in the text field.)
     */
    public StatCalcGUI() {
```

```
      stats = new StatCalc();

      numberInput = new JTextField();
      numberInput.setBackground(Color.WHITE);
      numberInput.addActionListener(this);

      enterButton = new JButton("Enter");
      enterButton.addActionListener(this);

      clearButton = new JButton("Clear");
      clearButton.addActionListener(this);

      JPanel inputPanel = new JPanel();  // A panel that will hold the
                                         //   JTextField and JButtons.
      inputPanel.setLayout( new GridLayout(1,3) );
      inputPanel.add(numberInput);
      inputPanel.add(enterButton);
      inputPanel.add(clearButton);

      countLabel =   makeLabel(" Number of Entries:  0");
      sumLabel =     makeLabel(" Sum:                0.0");
      meanLabel =    makeLabel(" Average:            undefined");
      standevLabel = makeLabel(" Standard Deviation: undefined");

      message = new JLabel("Enter a number, press return:",
                                    JLabel.CENTER);
      message.setBackground(labelBG);
      message.setForeground(Color.BLUE);
      message.setOpaque(true);
      message.setFont(new Font("SansSerif", Font.BOLD, 12));

      /* Use a GridLayout with 6 rows and 1 column, and add all the
         components that have been created to the panel. */

      setBackground(Color.BLUE);
      setLayout( new GridLayout(6,1,2,2) );
      add(message);
      add(inputPanel);
      add(countLabel);
      add(sumLabel);
      add(meanLabel);
      add(standevLabel);

      /* Add a blue border around the panel. */

      setBorder( BorderFactory.createLineBorder(Color.BLUE, 2) );

   } // end constructor


   /**
    * A utility routine for creating the labels that are used
    * for display.  This routine is called by the constructor.
    * @param text The text to show on the label.
    */
   private JLabel makeLabel(String text) {
      JLabel label = new JLabel(text);
      label.setBackground(labelBG);
      label.setForeground(labelFG);
```

```
       label.setFont(labelFont);
       label.setOpaque(true);
       return label;
    }


    /**
     * This is called when the user clicks one of the buttons or
     * presses return in the input box.  The response to clicking
     * on the Enter button is the same as the response to pressing
     * return in the JTextField.
     */
    public void actionPerformed(ActionEvent evt) {

       Object source = evt.getSource();  // Object that generated
                                         //   the action event.

       if (source == clearButton) {
             // Handle the clear button by starting with a new,
             // empty StatCalc object and resetting the display
             // labels to show no data entered.  The TextField
             // is also made empty.
          stats = new StatCalc();
          countLabel.setText(" Number of Entries:  0");
          sumLabel.setText(" Sum:                0.0");
          meanLabel.setText(" Average:            undefined");
          standevLabel.setText(" Standard Deviation: undefined");
          numberInput.setText("");
       }
       else if (source == enterButton || source == numberInput) {
             // Get the user's number, enter it into the StatCalc
             // object, and set the display on the display labels
             // to reflect the new data.
          double num;  // The user's number.
          try {
             num = Double.parseDouble(numberInput.getText());
          }
          catch (NumberFormatException e) {
                 // The user's entry is not a legal number.
                 // Put an error message in the message label
                 // and return without entering a number.
             message.setText("\"" + numberInput.getText() +
                                 "\" is not a legal number.");
             numberInput.selectAll();
             numberInput.requestFocus();
             return;
          }
          stats.enter(num);
          countLabel.setText(" Number of Entries:  " + stats.getCount());
          sumLabel.setText(" Sum:                " + stats.getSum());
          meanLabel.setText(" Average:            " + stats.getMean());
          standevLabel.setText(" Standard Deviation: "
                                                    +
stats.getStandardDeviation());
       }

       /* Set the message label back to its normal text, in case it has
          been showing an error message.  For the user's convenience,
          select the text in the TextField and give the input focus
```

```
          to the text field.  That way the user can just start typing
          the next number. */

       message.setText("Enter a number, press return:");
       numberInput.selectAll();
       numberInput.requestFocus();

    }  // end ActionPerformed

}  // end StatsCalcGUI




/**
 * An object of class StatCalc can be used to compute several simple
statistics
 * for a set of numbers.  Numbers are entered into the dataset using
 * the enter(double) method.  Methods are provided to return the
following
 * statistics for the set of numbers that have been entered: The number
 * of items, the sum of the items, the average, and the standard
deviation
 */

public class StatCalc {

    private int count;   // Number of numbers that have been entered.
    private double sum;  // The sum of all the items that have been
entered.
    private double squareSum;  // The sum of the squares of all the
items.

    /**
     * Add a number to the dataset.  The statistics will be computed for
all
     * the numbers that have been added to the dataset using this method.
     */
    public void enter(double num) {
        count++;
        sum += num;
        squareSum += num*num;
    }

    /**
     * Return the number of items that have been entered into the
dataset.
     */
    public int getCount() {
        return count;
    }

    /**
     * Return the sum of all the numbers that have been entered.
     */
    public double getSum() {
        return sum;
    }
```

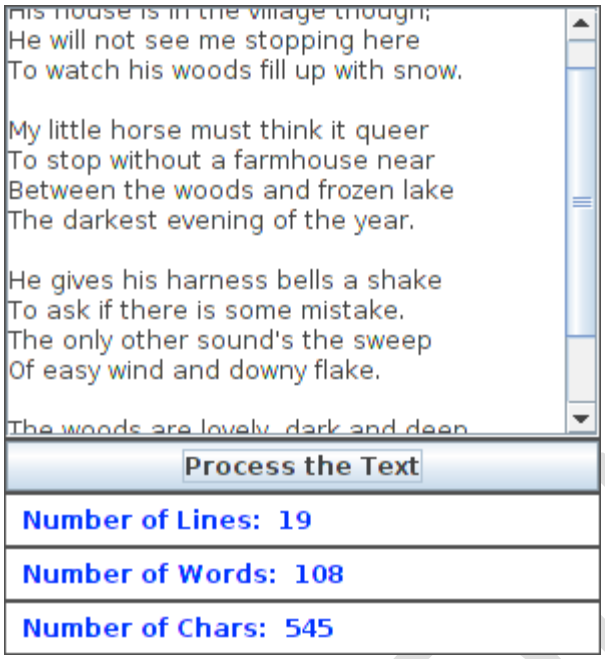
```
    /**
     * Return the average of all the items that have been entered.
     * The return value is Double.NaN if no numbers have been entered.
     */
    public double getMean() {
        return sum / count;
    }

    /**
     * Return the standard deviation of all the items that have been
entered.
     * The return value is Double.NaN if no numbers have been entered.
     */
    public double getStandardDeviation() {
        double mean = getMean();
        return Math.sqrt( squareSum/count - mean*mean );
    }

}  // end class StatCalc
```

Exercise 4.2:

Write a program that has a *JTextArea* where the user can enter some text. Then program should have a button such that when the user clicks on the button, the panel will count the number of lines in the user's input, the number of words in the user's input, and the number of characters in the user's input. This information should be displayed on three labels. Recall that if `textInput` is a *JTextArea*, then you can get the contents of the *JTextArea* by calling the function `textInput.getText()`. Don't forget to put your *JTextArea* in a *JScrollPane*, and add the scroll pane to the container, not the text area. Scrollbars should appear when the user types more text than will fit in the available area. Here is a picture of my solution:



Discussion

The panel contains five components. There are several ways to lay them out. A `GridLayout` with five rows certainly will **not** work, because the *JTextArea* should be taller than the other components. One possible layout is to use a *GridLayout* with two rows. The *JTextArea* would occupy the first row. The bottom half would contain a *JPanel* that holds the other four components. (A *GridLayout* with two columns and one row would also work, if you wanted a panel that was wider and not so tall. You could put the *JTextArea* in the left half and the other components in a *JPanel* in the right half.) However, I decided to use a *BorderLayout*. The *JTextArea* occupies the `CENTER` position, and the `SOUTH` position is occupied by a *JPanel* that contains the other components. The nested *JPanel* uses a *GridLayout* with four rows. My `main()` program sets the size of the window to 300-by-350, and the text area gets the space not occupied by the bottom panel. Once this choice has been made, writing the constructor is not hard.

I use an anonymous inner class to listen for *ActionEvents* from the button. The `actionPerformed()` method of the listener just calls a method named `processInput()` in the main class; this method does the real work. The `processInput()` method just has to get the text from the `JTextArea`, do the counting, and set the labels. The only interesting part is counting the words. Back in Exercise 3.4, words such as "can't", that contain an apostrophe, were counted as

two words. This time around, let's handle this special case. Two letters with an apostrophe between them should be counted as part of the same word. The algorithm for counting words is still

```
wordCt = 0
for each character in the string:
   if the character is the first character of a word:
      Add 1 to wordCt
```

but testing whether a given character is the first character in a word has gotten a little more complicated. To make the test easier, I use a boolean variable, `startOfWord`. The value of this variable is set to true if the character is the start of a word and to false if not. That is, the algorithm becomes:

```
wordCt = 0
for each character in the string:
   Let startOfWord be true if at start of word, false otherwise
   if startOfWord is true:
      Add 1 to wordCt
```

The use of a "flag variable" like `startOfWord` can simplify the calculation of a complicated boolean condition. The value is computed as a series of tests:

```
boolean startOfWord;  // Is character i the start of a word?
if ( Character.isLetter(text.charAt(i)) == false )
   startOfWord = false;  // No.  It's not a letter.
else if (i == 0)
   startOfWord = true;   // Yes.  It's a letter at start of text.
else if ( Character.isLetter(text.charAt(i-1)) )
   startOfWord = false;  // No.  It's a letter preceded by a
letter.
else if ( text.charAt(i-1) == '\'' && i > 1
                     && Character.isLetter(text.charAt(i-2)) )
   startOfWord = false;  // No.  It's a continuation of a word
                         //      after an apostrophe.
else
   startOfWord = true;   // Yes.  It's a letter preceded by
                         //       a non-letter.
```

The first test checks whether the character in position `i` is a letter. If it is not, then we know that it can't be the start of a word, so `startOfWord` is false. If it is a letter, it might be the start of a word, so we go on to make additional tests. Note that if we get to the other tests at all, we already know that the character in position `i` is a letter. And so on. This style of "cascading tests" is very useful. In each test, we already have all the information from the previous tests. Note that the cascade effect works only with "`else if`". Using "`if`" in place of "`else if`" in the preceding code would not give the right answer. (You should be sure to understand why this is so.) You should also note why the test `if (i == 0)` has to be made before the test `if ( Character.isLetter(text.charAt(i-1)) )` -- it's because `text.charAt(i-1)` gives an index-out-of-bounds exception if `i` is zero.

The Solution

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

/**
 * In this panel, the user types some text in a JTextArea and presses
 * a button.  The panel computes and displays the number of lines
 * in the text, the number of words in the text, and the number of
 * characters in the text.  A word is defined to be a sequence of
 * letters, except that an apostrophe with a letter on each side
 * of it is considered to be a letter.  (Thus "can't" is one word,
 * not two.)
 */
public class TextCounter extends JPanel {

    /**
     * A main routine allows this class to be run as an application.
     */
    public static void main(String[] args) {
            JFrame window = new JFrame("Text Counter");
            TextCounter content = new TextCounter();
            window.setContentPane(content);
            window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            window.setLocation(120,70);
            window.setSize(300,350);
            window.setVisible(true);
    }

    //-------------------------------------------------------------------
-

    private JTextArea textInput;     // For the user's input text.

    private JLabel lineCountLabel;   // For displaying the number of
lines.
    private JLabel wordCountLabel;   // For displaying the number of
words.
    private JLabel charCountLabel;   // For displaying the number of
chars.


    /**
     * The constructor creates components and lays out the panel.
     */
    public TextCounter() {

            setBackground(Color.DARK_GRAY);

            /* Create the text input area and make sure it has a
              white background. */

            textInput = new JTextArea();
            textInput.setBackground(Color.WHITE);
```

```
        /* Create a panel to hold the button and three display
          labels.  These will be laid out in a GridLayout with
          4 rows and 1 column. */

        JPanel south = new JPanel();
        south.setBackground(Color.DARK_GRAY);
        south.setLayout( new GridLayout(4,1,2,2) );

        /* Create the button and a listener to listen for
          clicks on the button, and add it to the panel. */

        JButton countButton = new JButton("Process the Text");
        countButton.addActionListener( new ActionListener() {
                public void actionPerformed(ActionEvent evt) {
                        processInput();
                }
        });
        south.add(countButton);

        /* Create each of the labels, set their colors, and
          add them to the panel. */

        lineCountLabel = new JLabel("  Number of lines:");
        lineCountLabel.setBackground(Color.WHITE);
        lineCountLabel.setForeground(Color.BLUE);
        lineCountLabel.setOpaque(true);
        south.add(lineCountLabel);

        wordCountLabel = new JLabel("  Number of words:");
        wordCountLabel.setBackground(Color.WHITE);
        wordCountLabel.setForeground(Color.BLUE);
        wordCountLabel.setOpaque(true);
        south.add(wordCountLabel);

        charCountLabel = new JLabel("  Number of chars:");
        charCountLabel.setBackground(Color.WHITE);
        charCountLabel.setForeground(Color.BLUE);
        charCountLabel.setOpaque(true);
        south.add(charCountLabel);

        /* Use a BorderLayout on the panel.  Although a BorderLayout
          is the default, I want one with a vertical gap of two
          pixels, to let the dark gray background color show through.
          Also add a gray border around the panel. */

        setLayout( new BorderLayout(2,2) );
        setBorder(BorderFactory.createLineBorder(Color.DARK_GRAY));

        /* The text area is put into a JScrollPane to provide
          scroll bars for the TextArea, and the scroll pane is put in
          the Center position.  The panel that holds the button and
          labels is in the South position.  Note that the text area
          will be sized to fill the space that is left after the
          panel is assigned its preferred height. */

        JScrollPane scroller = new JScrollPane( textInput );
        add(scroller, BorderLayout.CENTER);
        add(south, BorderLayout.SOUTH);
```

```
   } // end constructor


   /**
    * This will be called by the action listener for the button when the
user
    * clicks the button.  It gets the text from the text area, counts the
number
    * of chars, words, and lines that it contains, and sets the labels to
    * display the results.
    */
   public void processInput() {

           String text;  // The user's input from the text area.

           int charCt, wordCt, lineCt;  // Char, word, and line counts.

           text = textInput.getText();

           charCt = text.length();  // The number of characters in the
                                    //    text is just its length.

           /* Compute the wordCt by counting the number of characters
              in the text that lie at the beginning of a word.  The
              beginning of a word is a letter such that the preceding
              character is not a letter.  This is complicated by two
              things:  If the letter is the first character in the
              text, then it is the beginning of a word.  If the letter
              is preceded by an apostrophe, and the apostrophe is
              preceded by a letter, than its not the first character
              in a word.
            */

           wordCt = 0;
           for (int i = 0; i < charCt; i++) {
                   boolean startOfWord;  // Is character i the start of a
word?
                   if ( Character.isLetter(text.charAt(i)) == false )
                           startOfWord = false;  // No.  It's not a
letter.
                   else if (i == 0)
                           startOfWord = true;   // Yes.  It's a letter at
start of text.
                   else if ( Character.isLetter(text.charAt(i-1)) )
                           startOfWord = false;  // No.  It's a letter
preceded by a letter.
                   else if ( text.charAt(i-1) == '\'' && i > 1
                                   && Character.isLetter(text.charAt(i-2))
)
                           startOfWord = false;  // No.  It's a
continuation of a word
                                                 //      after an apostrophe.
                   else
                           startOfWord = true;   // Yes.  It's a letter
preceded by
                                                 //       a non-letter.
                   if (startOfWord)
                           wordCt++;
```

```
            }

            /* The number of lines is just one plus the number of times
the
               end of line character, '\n', occurs in the text. */

            lineCt = 1;
            for (int i = 0; i < charCt; i++) {
                    if (text.charAt(i) == '\n')
                            lineCt++;
            }

            /* Set the labels to display the data. */

            lineCountLabel.setText("  Number of Lines:  " + lineCt);
            wordCountLabel.setText("  Number of Words:  " + wordCt);
            charCountLabel.setText("  Number of Chars:  " + charCt);

    }  // end processInput()


} // end class TextCounter
```

E. Kết quả thực hành.

- Sinh viên thực hành ứng dụng trên GUI.
- Thời gian thực hành: 4 tiết.

F. Đánh giá:

- Kiểm tra lại chương trình, thử các kết quả.
- Bắt các lỗi bằng cách sử dụng các phần bắt lỗi: try – catch.

----------Hết----------