Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

The AWT Package, Placing Components in Containers, CardLayout

Java Programming, Lecture Notes # 120, Revised 03/03/99.

- Preface
- Introduction
- CardLayout Manager
- Sample Program
  - Discussion
  - Interesting Code Fragments
  - Program Listing
- Review

Preface

Students in Prof. Baldwin's **Intermediate Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

JDK 1.1 was formally released on February 18, 1997. This lesson was originally written on March 8, 1997 using the software and documentation in the JDK 1.1 download package, and has been revised several times since then.

The sample program in this lesson was confirmed to compile and run properly under JDK 1.2 on 2/3/99.

Introduction

This is one in a series of lessons that concentrate on the package **java.awt** where most of the functionality exists for providing the user interface to your application or applet.

In this lesson, we will learn how to use the **CardLayout** class for creating a layout manager.

Also, we will continue learning how to create user-interface objects by building up a combination of smaller objects into the final composite object.

In particular, we will create a top-level user-interface object that is a **Frame** object containing two **Panel** objects. One of the **Panel** objects will serve as a *display panel* on which we can display any one of several *"cards"* using a **CardLayout** manager.

The other **Panel** object contains several **Button** objects, which we can use to flip through the cards being displayed on the other **Panel** object.

All but one of the cards in the deck are passive (have no registered event listener objects).

One of the cards in the deck contains a **Button** object, which is not passive. In this case, clicking on the button causes the date and time to be displayed on the same card.

This is one of the processes that you can use to create the types of complex User Interface objects that you are accustomed to seeing in typical Graphical User Interfaces.

CardLayout Manager

In earlier lessons, we have learned about absolute positioning and sizing of components in a container, and we have learned about the **BorderLayout** manager, the **FlowLayout** manager, and the **GridLayout** manager.

In this lesson, we will learn about the **CardLayout** manager.

The **CardLayout** manager is a layout manager for a container that contains several *"cards"*. Only one card is visible at a time, allowing you to flip through the cards using methods of the class.

The **CardLayout** class has no *Fields*, has two *Constructors*, and has about twenty *methods*.

You will usually be concerned with the *Constructors* and with several of the methods that are used to display the cards. In the sample program in this lesson, we will use five different methods to display the cards in a deck of cards in various ways.

The constructors are:

- **CardLayout()** -- Creates a new card layout with gaps of size zero.
- **CardLayout(int, int)** -- Creates a card layout with the specified gaps.

The terminology used in the description of these constructors should be familiar to you by now.

The following list of methods that will be used in the sample program in this lesson:

- **first(Container)** -- Flip to the first card.
- **last(Container)** -- Flips to the last card of the specified container.
- **next(Container)** -- Flips to the next card of the specified container.
- **previous(Container)** -- Flips to the previous card of the specified container.
- **show(Container, String)** -- Flips to the specified component name in the specified container.

This lesson contains one sample program and it is fairly long. However, much of the length derives from code that is generally repetitive.

This sample program is more complex that previous programs in the lessons on layout managers. The **CardLayout** manager is slightly more complex than those previously studied, but with that complexity comes considerable power.

The **CardLayout** manager makes it possible for you to create a Graphical User Interfaces consisting of multiple pages for communication with the user, with only one page being visible at a time..

This program has **ActionListener** objects working at two levels. At one level, **ActionListener** objects are used to select a card for viewing.

At another level, an **ActionListener** object is registered to service a button on one of the cards to cause the date and time to be displayed, when that card is displayed, and when the button on that card is clicked.

For clarity, the main sample program in this lesson uses a parameterized constructor for the **ActionListener** object for this button. Another sample program at the end of this lesson shows how to eliminate the use of such parameterized constructors, which is considered by some to represent a better and more object-oriented approach. However, the second approach is a little more complex from a coding viewpoint.

Sample Program

The following sample program illustrates use of the **CardLayout** manager.

Discussion

This program is designed to be compiled and run under JDK 1.1

A top-level **Frame** object is created which contains two **Panel** objects. One of the **Panel** objects is a control panel that will be discussed in more detail later.

The other **Panel** object is a display panel that is used to display each of the following *"cards."* The cards are added to the panel using a **CardLayout** manager.

- Button("First card is a Button object")
- Label("Second card is a Label object")
- Label("Third card is also a Label object")
- Label("Fourth card is a label object")
- timePanel,"time panel"); (see note below)
- TextField("Last card is a TextField Object")

Except for the card identified as a *"timePanel"*, each of the cards is an object of the type indicated and is passive (no event listener objects registered).

The card identified as a *"time panel"* is a **Panel** object to which has been added a **Button** object and a **Label** object.

This card is not passive. Rather, an **ActionListener** object is instantiated and registered on the button such that clicking the button causes the current date and time to be displayed in a **Label** object on the same card.

As mentioned earlier, the **Frame** object contains two panels. The *display panel* is described above. The other panel is a *control panel*, which contains five **Button** objects labeled as shown below:

- "Next"
- "Previous"
- "First"
- "Last"
- "Show Time Panel"

These buttons are all active (have ActionListener objects registered on them). The buttons are used to flip through the deck of cards on the display panel. The action of each button is indicated by its label (for example, the button labeled *"Next"* causes the next card in the deck to be displayed.)

A **windowClosing()** event listener object is instantiated and registered on the frame to terminate the program when the frame is closed.

The program was tested using JDK 1.1.3 running under Win95.

Interesting Code Fragments

Because of its complexity, this program contains a large number of interesting code fragments. This program builds a top-level user-interface object by combining lower-level objects to create the whole. The top-level object is a **Frame** object onto which two **Panel** objects are placed.

One of the **Panel** objects serves as the *display panel* for six different *"cards"* which are placed on the **Panel** object using the **CardLayout** manager.

One of those cards is itself a **Panel** object. A **Button** object and a **Label** object have been placed on that card. When the user displays this card and clicks the button, the date and time are displayed in the **Label** object on the same card.

The other panel that is placed on the top-level Frame object contains five **Button** objects. These **Button** objects are visible any time the top-level user-interface object is visible, and are used to flip through the cards on the other **Panel** object.

The first interesting code fragment is used to create one of the cards which contains a **Button** object and a **Label** object.

An **ActionListener** object is instantiated and registered on the **Button** object. The overridden **actionPerformed()** method in the **ActionListener** object causes the date and time to be displayed in the **Label** object on the card when the button on the card is clicked. That code fragment follows.

```
    Label timeLabel = new Label(
            "_____________________________________________");
    Button timeButton = new Button("Display Date and
Time");
    Panel timePanel = new Panel();
    timePanel.add(timeButton);
    timePanel.add(timeLabel);

    timeButton.addActionListener(
                          new
TimeActionListener(timeLabel));
```

The next interesting code fragment is used to build a display panel for the deck of cards.

First we will instantiate a **CardLayout** manager object and assign it to a reference variable named **myCardLayout**. This cannot be an anonymous object because we will need access to it later when processing events.

```
    CardLayout myCardLayout = new CardLayout();
```

Next, we will

- instantiate our display panel object,
- specify its layout manager, and
- make it yellow so that it can be visually distinguished from the other **Panel** object on the top-level user-interface object.

.

```
    Panel displayPanel = new Panel();
    displayPanel.setLayout(myCardLayout);
    displayPanel.setBackground(Color.yellow);
```

Once the display panel object exists, the next step is to add the cards to the panel using the **CardLayout** manager. In the following usage of the **add()** method from the **Container** class,

- the first parameter is the object to be added and

- the second parameter is the *name* of the object.

All but one of these objects are added as anonymous objects which is OK because they are passive objects (no event listeners registered on them).

The non-anonymous object is the **timePanel** object that was constructed earlier in the program. This cannot be an anonymous object because it is the composite of a **Panel** object, a **Button** object, and a **Label** object. A variable referencing the panel was required in order to construct it.

The *name* of the object specified as a string in the second parameter to the **add()** method will be used in our sample program as a parameter to the **show()** method, by which a specified card can be made visible (brought to the top of the stack).

Please note that I determined the syntax used with the **add()** method in the following code fragment largely by trial and error.

As of 3/8/97, I have been unable to locate definitive information as to how objects should be added to a JDK 1.1 container under the **CardLayout** manager without using deprecated methods from JDK 1.0.

The syntax shown below seems to be the correct syntax (it works), but as near as I can tell, it is not shown in the JDK 1.1 documentation. I have asked for information on one of the Java newsgroups and hopefully I will be able to update this lesson to be more definitive about this syntax later.

```
displayPanel.add(new Button(
                 "First card is a Button
object"),"first");
displayPanel.add(new Label(
                "Second card is a Label
object"),"second");
displayPanel.add(new Label(
             "Third card is also a Label
object"),"third");
displayPanel.add(new Label(
                "Fourth card is a label
object"),"fourth");
displayPanel.add(
   timePanel,"time panel");//special panel
defined earlier
displayPanel.add(new TextField(
              "Last card is a TextField
Object"),"sixth");
```

That completes the construction of the display panel. The next task is to construct the control panel.

Construction of the control panel is pretty straightforward using capabilities that we have studied in previous lessons.

The following two statements are typical of the largely repetitive statements used to instantiate the **Button** objects for the control panel and register **ActionListener** objects on those buttons.

```
    Button nextButton = new Button("Next");

    firstButton.addActionListener(
            new
FirstListener(myCardLayout,displayPanel));
```

The following code fragment is typical of the largely repetitive code used to instantiate the control panel object and place the five buttons on it.

```
    Panel controlPanel = new Panel();
    controlPanel.add(firstButton);
```

We have now reached the point where we can construct the top-level user-interface object by placing the display panel and the control panel objects on the **Frame** object using the default **BorderLayout** manager of the **Frame** object.

```
    Frame myFrame = new Frame(
                            "Copyright 1997,
R.G.Baldwin");

    myFrame.add(displayPanel,"North");
    myFrame.add(controlPanel,"South");
```

That completes the interesting code fragments from the user-interface (GUI) class.

Now we need to define the Listener classes containing overridden **actionPerformed()** methods to satisfy the Listener object instantiation and registration code shown above.

The first **ActionListener** class that we will study is the class used to service the action event on the **Button** object, which is on one of the cards in the deck of cards.

That class is named **TimeActionListener**, It contains

- one instance variable,
- one constructor, and
- an overridden **actionPerformed()** method.

There is nothing particularly interesting about the instance variable or the constructor. As mentioned earlier, another sample program at the end of this lesson shows how to eliminate the use of the instance variable and the parameterized constructor.

The overridden **actionPerformed()** method is shown in the following code fragment.

```
      public void actionPerformed(ActionEvent
e){
        myLabelObject.setText(new
Date().toString());
      }//end actionPerformed()
```

As you can see, this overridden method instantiates a new instance of the **Date** class.

Whenever an instance of the **Date** class is instantiated, it contains information that can be used to determine the date and time at which it was instantiated.

This information is extracted using the **toString()** method and used with the **setText()** method of the **Label** class to deposit the date and time information in the **Label** object.

This class definition is followed by five very similar **ActionListener** classes, which are used to flip through the deck of cards whenever one of the **Button** objects on the control panel is clicked.

These class definitions are almost identical, differing only in terms of the method called inside of the overridden **actionPerformed()** method.

None of the rest of the code in the classes is particularly interesting. Therefore, in this case, I am simply going to show the individual statements which make those method calls inside the overridden **actionPerformed()** methods of the five classes. **(Note that these five statements appear inside five different methods.)**

```
    myCardLayoutObject.first(myPanelObject);
    myCardLayoutObject.next(myPanelObject);
    myCardLayoutObject.previous(myPanelObject);
    myCardLayoutObject.last(myPanelObject);
    myCardLayoutObject.show(myPanelObject,"time
panel");
```

You can tell from the highlighted method name how the method is used to select another card for viewing.

Pay particular attention to the **show()** method which takes an additional parameter relative to the others. This parameter is the *name* of the component that is established when the component is added to its container under control of the **CardLayout** manager. This method can be used to

display any card in the deck regardless of its position relative to the card currently being displayed.

Program Listing

This section contains a complete listing of the program with a large number of explanatory comments.

```
/*File Layout07.java Copyright 1997, R.G.Baldwin
Revised 10/29/97 to make it fit on the printed
page better.

This program is designed to be compiled and run under
JDK 1.1

This program illustrates use of the CardLayout manager.

A top-level Frame object is created which contains two
Panel objects.

One of the panels is a control panel that will be discussed
in more detail later.

The other panel is a display panel that is used to display
each of the following "cards" which are added to the panel
using a CardLayout manager.

Button("First card is a Button object")
Label("Second card is a Label object")
Label("Third card is also a Label object")
Label("Fourth card is a label object")
timePanel,"time panel");  (see note below)
TextField("Last card is a TextField Object")

Except for the card identified as a "timePanel", each of
the cards is an object of the type indicated and is
passive (no event listener objects registered).

The card identified as a "time panel" is a Panel object to
which has been added a Button object and a Label object.
This card is not passive.  Rather, an ActionListener
object is instantiated and registered on the button such
that clicking the button causes the current date and time
to be displayed in the Label object.

As mentioned earlier, the Frame object contains two
panels.  The display panel is described above.  The other
panel is a control panel which contains five buttons
labeled as shown below:

"Next"
"Previous"
"First"
```

```
"Last"
"Show Time Panel"

These buttons are all active (have ActionListener objects
registered on them). The buttons are used to iterate
through the deck of cards on the display panel.  The
action of each button is indicated by its label (for
example, the button labeled "Next" causes the next card
in the deck to be displayed.)

A windowClosing() event listener object is instantiated
and registered on the frame to terminate the program when
the frame is closed.

The program was tested using JDK 1.1 running under Win95.
*/
//=========================================================//

import java.awt.*;
import java.awt.event.*;
import java.util.*;
//=========================================================//
public class Layout07 {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class Layout07
//=========================================================//

class GUI {
  public GUI(){//constructor

    //Build a card with GUI components to be added to a
    // deck of cards using a CardLayout manager.

    Label timeLabel = new Label(
                           "_______________________________");

    Button timeButton = new Button(
                                  "Display Date and Time");
    Panel timePanel = new Panel();
    timePanel.add(timeButton);
    timePanel.add(timeLabel);

    //Instantiate ActionListener object and register it on
    // the button This event handler will display the date
    // and time.
    timeButton.addActionListener(
                       new TimeActionListener(timeLabel));

    //===Build a display panel for the deck of cards===

    //Instantiate a layout manager object to be used with
    // a Panel object
    CardLayout myCardLayout = new CardLayout();
```

```
    //Instantiate a display Panel object that will be
    // composited onto a Frame object.
    Panel displayPanel = new Panel();

    //Specify a CardLayout manager for the Panel object
    displayPanel.setLayout(myCardLayout);
    //make the display panel visible
    displayPanel.setBackground(Color.yellow);

    //Add objects to the display panel using the specified
    // CardLayout manager
    displayPanel.add(new Button(
                 "First card is a Button object"),"first");
    displayPanel.add(new Label(
                "Second card is a Label object"),"second");
    displayPanel.add(new Label(
             "Third card is also a Label object"),"third");
    displayPanel.add(new Label(
                "Fourth card is a label object"),"fourth");
    //special panel defined earlier
    displayPanel.add(timePanel,"time panel");
    displayPanel.add(new TextField(
               "Last card is a TextField Object"),"sixth");


    //======== Build the control panel ======//

    //Instantiate button objects that will be used to
    // iterate through the cards in the deck.
    Button nextButton = new Button("Next");
    Button prevButton= new Button("Previous");
    Button firstButton= new Button("First");
    Button lastButton= new Button("Last");
    Button showButton= new Button("Show Time Panel");

    //Instantiate action listener objects and register on
    // the buttons
    firstButton.addActionListener(
             new FirstListener(myCardLayout,displayPanel));
    nextButton.addActionListener(
              new NextListener(myCardLayout,displayPanel));
    prevButton.addActionListener(
              new PrevListener(myCardLayout,displayPanel));
    lastButton.addActionListener(
              new LastListener(myCardLayout,displayPanel));
    showButton.addActionListener(
              new ShowListener(myCardLayout,displayPanel));


    //Instantiate a control Panel object using default
    // FlowLayout and place the Button objects on it.
    // These buttons are functional because ActionListener
    // objects have been registered on them.
    Panel controlPanel = new Panel();
    controlPanel.add(firstButton);
    controlPanel.add(nextButton);
```

```
    controlPanel.add(prevButton);
    controlPanel.add(lastButton);
    controlPanel.add(showButton);


    //== Build the Top-Level User-Interface Object ==

    //Instantiate a Frame object
    Frame myFrame = new Frame(
                            "Copyright 1997, R.G.Baldwin");

    //Add the display panel and the control panel objects
    // to the Frame object to create the composite
    // user-interface object.
    myFrame.add(displayPanel,"North");
    myFrame.add(controlPanel,"South");

    myFrame.setSize(500,150);//set the size
    myFrame.setVisible(true);//make it visible


    //Instantiate and register a window listener to
    // terminate the program when the Frame is closed.
    myFrame.addWindowListener(new Terminate());
  }//end constructor
}//end class GUI definition
//=======================================================//

//An object of this ActionListener class is registered on
// the Button object on the Time panel which is one of the
// cards in the deck of cards.  When an event occurs, this
// handler causes the date and time to be displayed on a
// Label object on the same panel.

    class TimeActionListener implements ActionListener{
      Label myLabelObject;

      TimeActionListener(Label inLabel){//constructor
        myLabelObject = inLabel;
      }//end constructor

      public void actionPerformed(ActionEvent e){
        myLabelObject.setText(new Date().toString());
      }//end actionPerformed()
    }//end class TimeActionListener

//=======================================================//

//Objects of the next five ActionListener classes are
// registered on the Button objects on the control panel.
// When an event occurs, the event handler causes a card
// in the deck of cards to be displayed on the display
// panel.  For example, an object of the following class
// causes the first card in the deck to be displayed.  All
// five of the classes are very similar, differing only by
// one statement in the overridden ActionPerformed
```

```
// method which specifies the action to be taken.

class FirstListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;

  //constructor
  FirstListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.first(myPanelObject);
  }//end actionPerformed()
}//end class NextListener
//=======================================================//

//See comments above in class FirstListener
class NextListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;

  //constructor
  NextListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.next(myPanelObject);
  }//end actionPerformed()
}//end class NextListener
//=======================================================//

//See comments above in class FirstListener
class PrevListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;

  //constructor
  PrevListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.previous(myPanelObject);
  }//end actionPerformed()
}//end class NextListener
//=======================================================//

//See comments above in class FirstListener
class LastListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;
```

```
  //constructor
  LastListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.last(myPanelObject);
  }//end actionPerformed()
}//end class NextListener

//=======================================================//

//See comments above in class FirstListener
class ShowListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;

  //constructor
  ShowListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    //The following method invocation will display the
    // card whose name matches the second parameter.
    myCardLayoutObject.show(myPanelObject,"time panel");
  }//end actionPerformed()
}//end class NextListener

//=======================================================//

class Terminate extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    //terminate the program when the window is closed
    System.exit(0);
  }//end windowClosing
}//end class Terminate
//=======================================================//
```

.

Review

Q - Without viewing the solution that follows, write a Java application that meets the specifications provided by the opening comments in the following application. Note in particular the specification regarding **parameterized constructors and listener objects**.

A - See solution below.

```
/*File SampProg138.java Copyright 1997, R.G.Baldwin
From lesson 120

Without viewing the solution that follows, write a Java
application that meets the following specifications.

When you start the program, a Frame object appears on the
screen with a width of 350 pixels and a height of 200
pixels.

The bottom portion of the client area of the Frame object
is yellow.  The top portion of the client area of the Frame
object is red.

At startup, the text string

First Card

appears in the red portion of the Frame object.

At startup, and throughout the running of the program, the
yellow portion of the Frame object contains two buttons,
one above the other. These buttons are labeled (from top to
bottom) as follows:

Show First Card
Show Next Card

Some yellow background shows through in the space between
the two buttons.

The yellow portion of the Frame object and the associated
buttons provide a control panel that is used to display
information in the red portion of the Frame object.

The red portion of the Frame object is a display area
that has the ability to display three different cards of
information.

The First Card of information shows at startup with the
label

First Card

The Last Card of information can be exposed by clicking the
button "Show Next Card" several times in succession, and
that card is labeled

Last Card

Between the first and last cards, there is an additional
card that can be exposed by clicking on the
"Show Next Card" button.

The labels on the buttons indicate which card will be
exposed when you click on a button.
```

```
The "Show Next Card" button cycles through all three cards,
exposing them in order.

When you expose the card between the first and last cards,
the red portion of the Frame object contains three
components in a vertical column:  two buttons and a green
rectangle.  A small portion of the red background shows
through between the components.

The top button in the red portion of the Frame object is
labeled:

Display Output

and the other button is labeled

Clear Output

When you click on the "Display Output" button the word
"Ouch" is displayed in the green rectangle.

When you click on the "Clear Output" button all text is
removed from the green rectangle.

Do not use parameterized constructors for the listener
objects for these two buttons.

When you press the close button on the Frame object, the
program terminates and returns control to the operating
system.

Your name should appear in the banner at the top of the
Frame object.

The program was tested using JDK 1.1.3 running under
Win95.
*/
//=======================================================//

import java.awt.*;
import java.awt.event.*;
import java.util.*;
//=======================================================//
public class SampProg138 {
  public static void main(String[] args){
    GUI gui = new GUI();
  }//end main
}//end class SampProg138
//=======================================================//

class GUI {

  public GUI(){//constructor

    //Build an active card.
```

```
    Label outputLabel = new Label("     ");
    outputLabel.setBackground(Color.green);
    Button setOutputButton = new Button("Display Output");
    Button clearOutputButton = new Button("Clear Output");
    Panel outputPanel = new Panel();
    GridLayout outputLayout = new GridLayout(3,1);
    outputLayout.setVgap(5);
    outputPanel.setLayout(outputLayout);
    outputPanel.add(setOutputButton);
    outputPanel.add(clearOutputButton);
    outputPanel.add(outputLabel);
    outputPanel.setName("Output");

    //Instantiate ActionListener objects and register them
    // on the set and clear buttons. These event handlers
    // will display the output or clear the display.
    setOutputButton.addActionListener(
                      new SetOutputActionListener());
    clearOutputButton.addActionListener(
                      new ClrOutputActionListener());
    //=================================================//

    //===Build a display panel for the deck of cards===

    //Instantiate a layout manager object to be used with
    // a Panel object
    CardLayout myCardLayout = new CardLayout();

    //Instantiate a display Panel object that will be
    // composited onto a Frame object.
    Panel displayPanel = new Panel();

    //Specify a CardLayout manager for the Panel object
    displayPanel.setLayout(myCardLayout);
    //make the display panel visible
    displayPanel.setBackground(Color.red);

    //Add objects to the display panel using the specified
    // CardLayout manager
    displayPanel.add(new Label("First Card"),"first");
    displayPanel.add(outputPanel,"output panel");
    displayPanel.add(new Label("Last Card"),"last");

    //======== Build the control panel ======//

    //Instantiate button objects that will be used to
    // cycle through the cards in the deck.
    Button firstButton= new Button("Show First Card");
    Button nextButton = new Button("Show Next Card");

    //Instantiate action listener objects and register on
    // the buttons on the control panel
    firstButton.addActionListener(
             new FirstListener(myCardLayout,displayPanel));
    nextButton.addActionListener(
```

```
              new NextListener(myCardLayout,displayPanel));

    //Instantiate a control Panel object and place the
    // Button objects on it.
    Panel controlPanel = new Panel();
    BorderLayout controlLayout = new BorderLayout();
    controlLayout.setVgap(5);
    controlPanel.setLayout(controlLayout);
    controlPanel.add(firstButton,"North");
    controlPanel.add(nextButton,"South");
    controlPanel.setBackground(Color.yellow);

    //== Build the Top-Level User-Interface Object ==

    //Instantiate a Frame object
    Frame myFrame = new Frame(
                           "Copyright 1997, R.G.Baldwin");

    //Add the display panel and the control panel objects
    // to the Frame object to create the composite
    // user-interface object.
    myFrame.add(displayPanel,"North");
    myFrame.add(controlPanel,"South");

    myFrame.setSize(350,200);//set the size
    myFrame.setVisible(true);//make it visible


    //Instantiate and register a window listener to
    // terminate the program when the Frame is closed.
    myFrame.addWindowListener(new Terminate());
  }//end constructor
}//end class GUI definition
//=========================================================//

//An object of this ActionListener class is registered on
// a Button object on a Panel object.  When an event
// occurs, this handler modifies the text in a Label object
// on the same panel.
    class SetOutputActionListener implements
ActionListener{
      public void actionPerformed(ActionEvent e){
        //Construct an array of the components on the
        // panel that is the parent object of the source
        // of the event.  Note the requirement for casting.
        Component[] comps = ((Component)e.getSource()).
                              getParent().getComponents();

        //Identify the Label component in the array and
        // modify its text.
        for(int cnt = 0; cnt < comps.length; cnt++){
          if(comps[cnt].toString().indexOf("Label") != -1)
            ((Label)comps[cnt]).setText("Ouch");
        }//end for loop
      }//end actionPerformed()
    }//end class SetOutputActionListener
```

```
//=======================================================//
//An object of this ActionListener class is registered on
// a Button object on a Panel object.  When an event
// occurs, this handler clears the text in a Label object
// on the same panel. This ActionListener is used to clear
// the names.

    class ClrOutputActionListener implements
ActionListener{
      public void actionPerformed(ActionEvent e){
        //Construct an array of the components on the
        // panel that is the parent object of the source
        // of the event.  Note the requirement for casting.
        Component[] comps = ((Component)e.getSource()).
                              getParent().getComponents();

        //Identify the Label component in the array and
        // modify its text.
        for(int cnt = 0; cnt < comps.length; cnt++){
          if(comps[cnt].toString().indexOf("Label") != -1)
            ((Label)comps[cnt]).setText("");
        }//end for loop
      }//end actionPerformed()
    }//end class ClrOutputActionListener
//=======================================================//

//Objects of the next two ActionListener classes are
// registered on the Button objects on the control panel.
// When an event occurs, the event handler causes a card
// in the deck of cards to be displayed on the display
// panel.  For example, an object of the following class
// causes the first card in the deck to be displayed.  All
// of the classes are very similar, differing only by
// one statement in the overridden ActionPerformed
// method which specifies the action to be taken.

class FirstListener implements ActionListener{
  Panel myPanelObject;
  CardLayout myCardLayoutObject;

  //constructor
  FirstListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.first(myPanelObject);
  }//end actionPerformed()
}//end class NextListener
//=======================================================//

//See comments above in class FirstListener
class NextListener implements ActionListener{
  Panel myPanelObject;
```

```
  CardLayout myCardLayoutObject;

  //constructor
  NextListener(CardLayout inCardLayout,Panel inPanel){
    myCardLayoutObject = inCardLayout;
    myPanelObject = inPanel;
  }//end constructor

  public void actionPerformed(ActionEvent e){
    myCardLayoutObject.next(myPanelObject);
  }//end actionPerformed()
}//end class NextListener
//=========================================================//

class Terminate extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    //terminate the program when the window is closed
    System.exit(0);
  }//end windowClosing
}//end class Terminate
//=========================================================//
```

-end-