JAVA NOTES

GRAPHICAL USER INTERFACES

Terry Marris 18 June 2001

3 BUTTONS

3.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- create buttons
- place buttons in panels
- nest panels
- change a panel's layout format to border layout
- program a response when a button is clicked

3.2 INTRODUCTION

In the last chapter we looked at panels. In this chapter we see how to put buttons in panels and how to program a response when a button is "pressed" by moving a mouse pointer over it and clicking the left hand button on the mouse.

3.3 EXIT BUTTON

Perhaps the simplest button is an exit button: when clicked the frame closes and the program terminates - just like when the frame-close icon is clicked

We put a button in a panel, and the panel inside a frame, as shown in Figure 3.1 below.

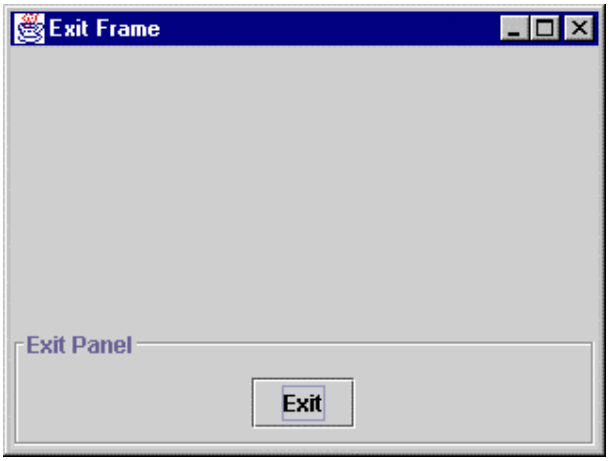


Figure 3.1 *An Exit Button*

To create a button is easy; we just supply its caption or label as an argument value to its constructor.

```
JButton exitButton = new JButton("Exit");
```

To add it to a panel is just as easy: we use the panel's *add(Component)* method.

```
add(exitButton);
```

3.4 IMPLEMENTING A BUTTON ACTION

When the exit button is clicked we want some action to take place. Associating some action with a button is easy:

1 choose a class to be responsible for listening for the button click event. Any object could be chosen but it usually convenient to choose one that can carry out the required action.

2 have the class implement the *ActionListener* interface. The *ActionListener* interface has just one method, namely *void actionPerformed(ActionEvent)*.

3 give the responsible object access to the button

4 register the responsible object to be that action listener for the button

5 write the code to implement the *actionPerformed(ActionEvent)* method

1 *choose a class to be responsible for listening for the button event* Looking at Figure 3.1 the Exit Panel seems an appropriate choice because it contains the exit button and can issue the command to close and terminate, namely *System.exit(0)*.

2 *have the class implement the ActionListener interface*

```
class ExitPanel extends JPanel implements ActionListener {
```

3 *give the responsible object access to the button* *ExitPanel* has an *exitButton*, so access is direct; we do not need to provide a method to access the button.

```
class ExitPanel extends JPanel implements ActionListener {
  private JButton exitButton;
```

4 *register the responsible object to be a listener for the button*

```
     exitButton.addActionListener(this);
```

5 *implement actionPerformed(ActionEvent)*

```
  public void actionPerformed(ActionEvent e)
  {
    Object source = e.getSource();
    if (source == exitButton)
      System.exit(0);
  }
```

When the user clicks the exit button, the *JButton* object creates an *ActionEvent* object and calls *actionPerformed*, passing that event into parameter e. The *getSource()* method will tell you the source of the event. If the *exitButton* was the source of the event, call *System.exit()* to close and terminate the application.

Shown below are the classes *ExitPanel* and *ExitFrame*. Notice that we use the convention that an object is defined before it is used. *ExitFrame* has an *ExitPanel* and so we write *ExitPanel* first, and *ExitFrame* last. Although the order does not matter to the compiler, it helps us humans to understand the code as we read through it.

```
/* ExitFrame.java
   Terry Marris  17 June 2001
*/


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


class ExitPanel extends JPanel implements ActionListener {
  private JButton exitButton;

  public ExitPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(
                                          etched, "Exit Panel");
    setBorder(titled);

    exitButton = new JButton("Exit");
    add(exitButton);
    exitButton.addActionListener(this);
  }


  public void actionPerformed(ActionEvent e)
  {
    Object source = e.getSource();
    if (source == exitButton)
      System.exit(0);
  }
}
```

```
public class ExitFrame extends JFrame {
  public ExitFrame()
  {
    setTitle("Exit Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
     });

    Container contentPane = getContentPane();

    JPanel exitPanel = new ExitPanel();
    contentPane.add(exitPanel, "South");
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new ExitFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of the *ExitPanel* and *ExitFrame* classes shown above.

3.5 RETRY BUTTON

We have seen that the default layout style for a frame is a border layout, with regions labelled North, South, West, East and Center. Well, we can make a panel adopt the same layout style.

Also in this example, we see that panels can be nested; we can have panels inside panels. And we show how one panel can implement behaviour when the user clicks a button in another panel.

The layout we are trying to achieve is shown below in Figure 3.2.

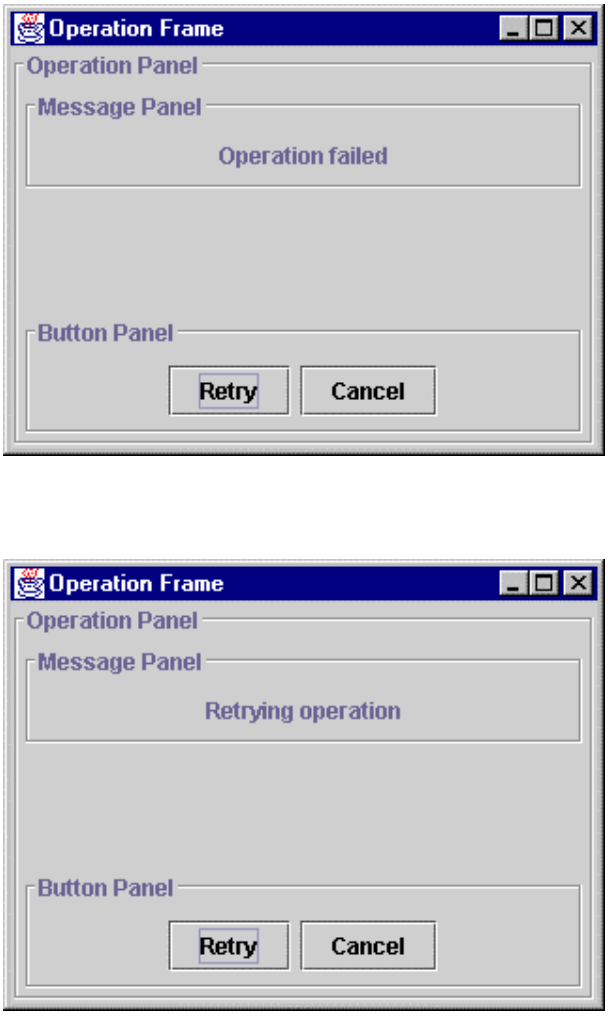


Figure 3.2 *Before and after the Retry button is clicked*

Some operation, it could be a disk format operation or a file read operation, has failed. The user is given the option to either retry the operation, or just cancel it. If the user clicks on *Retry*, the operation is attempted again. If the user clicks on *Cancel*, the frame is closed.

We see that an operation frame has an operation panel, and that an operation panel has a message panel and a button panel. We obtain the class diagram shown in Figure 3.3 below.

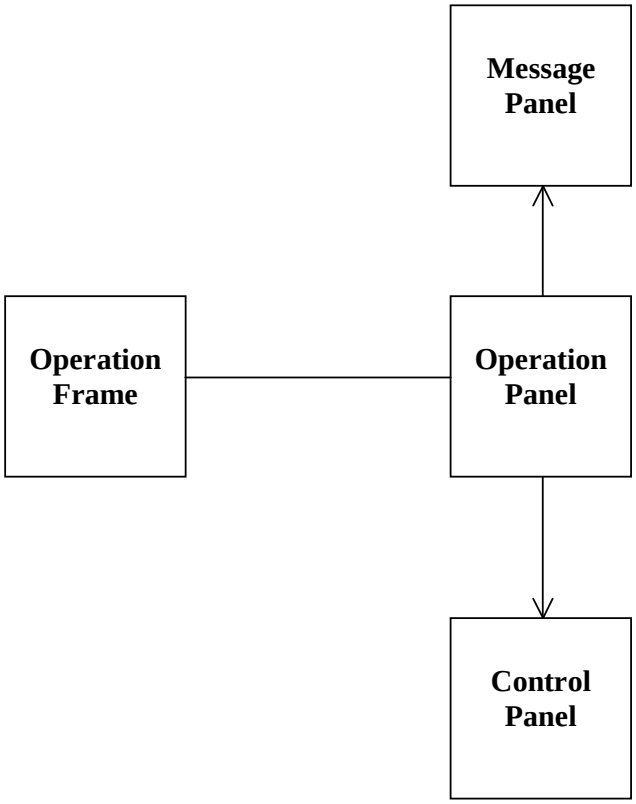
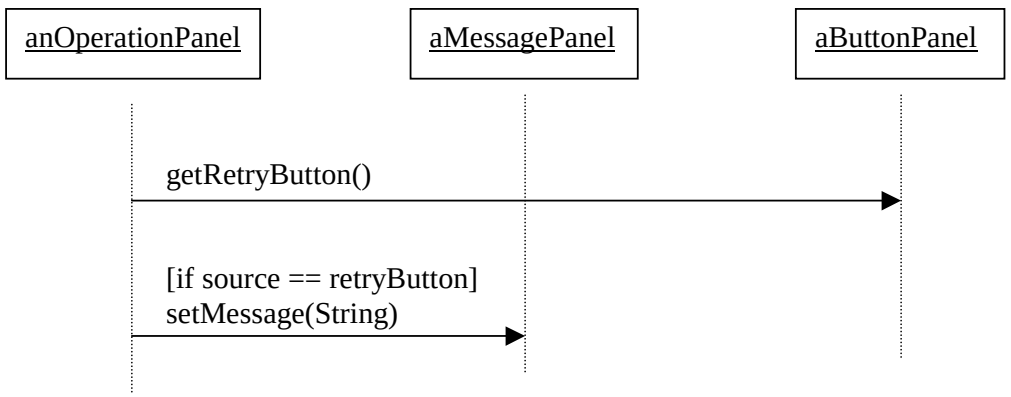


Figure 3.3 *An operation frame has an operation panel. An operation panel has a message panel and a control panel.*

We choose to make the operation panel responsible for listening for button events because an operation panel can say to its button panel *get me your buttons*, and an operation panel can say to its message panel *change your message*.



Following the principle of define it before you use it, we write the classes in this order:

message panel, button panel, control panel, control frame.

```
/* OperationFrame.java
   Terry Marris  17 June 2001
*/

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


class MessagePanel extends JPanel {
  private JLabel messageLabel;

  public MessagePanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(
                                  etched, "Message Panel");
    setBorder(titled);

    messageLabel = new JLabel("Operation failed");
    add(messageLabel);
  }


  public void setMessage(String message)
  {
    messageLabel.setText(message);
  }
}
```

The message panel has a label as its only field. The *messageLabel*'s initial value is *Operation failed*. But it can be assigned any other string value with the *setMessage(String)* method.

```
class ButtonPanel extends JPanel {
  private JButton retryButton;
  private JButton cancelButton;

  public ButtonPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(
                                        etched, "Button Panel");
    setBorder(titled);

    retryButton = new JButton("Retry");
    add(retryButton);

    cancelButton = new JButton("Cancel");
    add(cancelButton);
  }


  public JButton getRetryButton()
  {
    return retryButton;
  }


  public JButton getCancelButton()
  {
    return cancelButton;
  }
}
```

The button panel has two *JButtons*: *retryButton* and *cancelButton*. *get* methods are provided for each button because the operation panel (which contains a button panel) needs access to these buttons if it is to manage the response to button clicks.

```
class OperationPanel extends JPanel implements ActionListener
{
  private ButtonPanel buttonPanel;
  private MessagePanel messagePanel;

  public OperationPanel()
  {
    Border etched = BorderFactory.createEtchedBorder();
    Border titled = BorderFactory.createTitledBorder(
                                    etched, "Operation Panel");
    setBorder(titled);

    setLayout(new BorderLayout());

    messagePanel = new MessagePanel();
    add(messagePanel, "North");

    buttonPanel = new ButtonPanel();
    add(buttonPanel, "South");

    buttonPanel.getCancelButton().addActionListener(this);
    buttonPanel.getRetryButton().addActionListener(this);
  }


  public void actionPerformed(ActionEvent e)
  {
    Object source = e.getSource();
    if (source == buttonPanel.getCancelButton())
      System.exit(0);
    else if (source == buttonPanel.getRetryButton())
      messagePanel.setMessage("Retrying operation");
    repaint();
  }
}
```

The operation panel's layout is set to border layout with

```
setLayout(new BorderLayout());
```

The call to *repaint()* made in the *actionPerformed(ActionEvent)* method causes the screen to be re-drawn with the updated message in the message panel.

Exercise: Explain each line of the *OperationPanel* class shown above.

```
public class OperationFrame extends JFrame {
  public OperationFrame()
  {
    setTitle("Operation Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
     });

    Container contentPane = getContentPane();

    JPanel operationPanel = new OperationPanel();
    contentPane.add(operationPanel);
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new OperationFrame();
    aFrame.show();
  }
}
```

3.6 FURTHER READING

HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 316
www.java.sun.com/docs/books/tutorial

In the next chapter we look at text input/output via GUI.

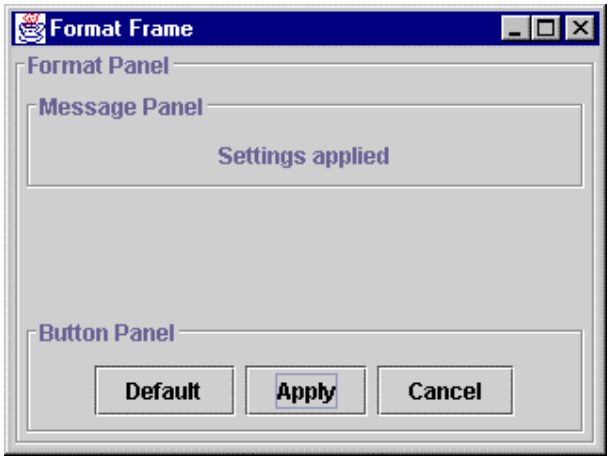
3.7 REVIEW

3.8 EXERCISES

1 Try out program *ExitFrame.java* shown in §3.4 above.

2 Try out program *OperationFrame.java* shown in §3.5 above.

3 Implement a program that produces the GUI shown below.



Initially, the MessagePanel is empty - no message is shown.

When the default button is clicked the message *Settings stored* should be displayed in the message panel.

When the apply button is clicked the message *Settings applied* should be shown.

When the cancel button is clicked the frame should close and the program terminate.