Chapter 14

Graphical User Interfaces

14.1 Introduction

☞ Java has had windowing capabilities since its earliest days. The first version made public was the Abstract Windowing Toolkit, or AWT. Because it used the native toolkit components, AWT was relatively small and simple.

☞ The second major implementation was the Swing classes, released in 1998 as part of the Java Foundation Classes. Swing is a full-function, professional-quality GUI toolkit designed to enable almost any kind of client-side GUI-based interaction.

☞ AWT lives inside, or rather underneath, Swing, and, for this reason, many programs begin by importing both java.awt and javax.swing.

***Containers and Components:** There are two types of GUI elements:

- Component: Components are elementary GUI entities, such as Button, Label, and TextField.
- Container: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.

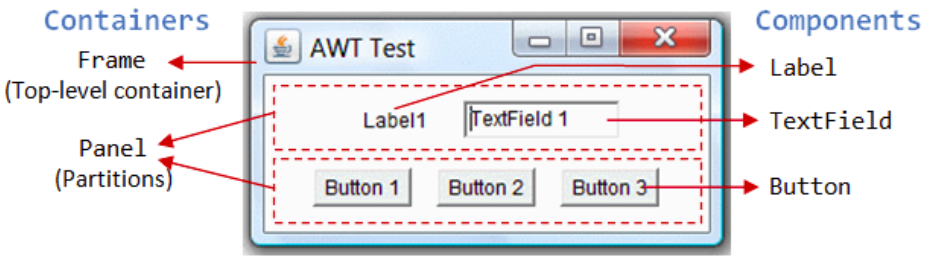


Figure 14.1: Containers and Components

***AWT Container Classes** Each GUI program has a top-level container. The commonly-used top-level containers in AWT are Frame, Dialog and Applet:

- A Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.
- An AWT Dialog is a "pop-up window" used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.

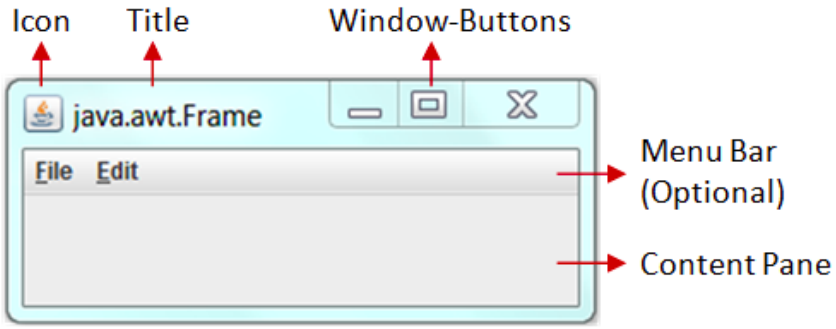


Figure 14.2: AWT Container Classes

14.1.1 Displaying GUI Components

Problem

You want to create some GUI components and have them appear in a window.

Solution

Create a JFrame and add the components to its ContentPane. **Discussion**

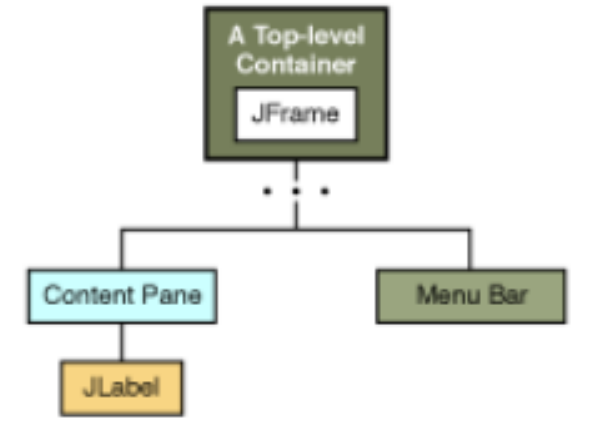


Figure 14.3: GUI

☞ The older AWT had a simple Frame component for making main windows; this allowed you to add components directly to it. “Good” programs usually created a panel to fit inside and populate the frame.

☞ The Swing JFrame is more complex—it comes with not one but two containers already constructed inside it.

☞ The ContentPane is the main container; you should normally use it as your JFrame's main container.

☞ The GlassPane has a clear background and sits over the top of the ContentPane ; its primary use is in temporarily painting something over the top of the main ContentPane.

☞ You can add any number of components (including containers) into this existing container, using the ContentPane add() method.

Example: JFrame Demo:

Code:

```
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JFrameDemo extends JFrame{
    public JFrameDemo() {
        Container cp = getContentPane();
        // now add Components to Container
        cp.add(new JLabel("A Really Simple Demo",
           JLabel.CENTER));
        }
    public static void main(String[] args) {
        JFrameDemo jfd=new JFrameDemo();
        jfd.setSize(500,100);
        jfd.setTitle("JFrame Demo");
        jfd.setVisible(true);
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Example: JFrame with Quit Button:

Code:

```
package com.GUI;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JFrameDemo2 extends JFrame{
JButton quitButton;
    public JFrameDemo2() {
        Container cp = getContentPane();
        // now add Components to Container
        cp.add(new JLabel("A Really Simple Demo",
           JLabel.CENTER));
        cp.setLayout(new FlowLayout());
        cp.add(quitButton = new JButton("Exit"));
        quitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
            setVisible(false);
            dispose();
            System.exit(0);
            }
        });
    }
    public static void main(String[] args) {
        JFrameDemo2 jfd=new JFrameDemo2();
        jfd.setSize(500,100);
        jfd.setTitle("JFrame Demo");
        jfd.setVisible(true);
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }}
```

14.1.2 Run Your GUI on the Event Dispatching Thread

Problem

Your application fails to start, with a message like Running on UI thread when not expected . Or, your application crashes very sporadically.

Solution

Run your UI on the UI thread, which Java names the “Event Dispatching Thread” or EDT.

Example: Dialog box message demo

```
package com.GUI;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
public class MessageDisplay {
    public static void main(String[] args) throws Exception {
        System.out.println("Program quite during the Run the
           main()");
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
        JOptionPane.showMessageDialog(null, "Do You Want To
           Exit");
                } catch (Exception e) {
                e.printStackTrace();
}}});}}
```

Assignment:

1. Demonstrate the use of GUI Components

Solution:

```
Panel pnl = new Panel();           // Panel is a container
Button btn = new Button("Press"); // Button is a component
pnl.add(btn);     // The Panel container adds a Button component
```

14.1.3 Designing a Window Layout

Problem

The default layout isn't good enough.

Solution

Learn to deal with a layout manager.

Discussion

- The container classes such as Panel have the capability to contain a series of components, but you can arrange components in a window in many ways. Rather than clutter up each container with a variety of different layout computations, the designers of the Java API used a sensible design pattern to divide the labor.
- A layout manager is an object that performs the layout computations for a container. The AWT package has five common layout manager classes, and Swing has a few more.
- If your JFrame is full of well-behaved components, you can set its size to be "just the size of all included components, plus a bit for padding," just by calling the pack() method, which takes no arguments.
- The pack() method goes around and asks each embedded component for its preferred size.

- The JFrame is then set to the best size to give the components their preferred sizes as much as is possible. If not using pack() , you need to call the setSize() method, which requires either a width and a height, or a Dimension object containing this information.

- JFrameDemo in one of two ways: either we can use a FlowLayout or specify BorderLayout regions for the label and the button.

Example: Layout Manager Demo

```
package com.GUI;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Container;
import java.awt.FlowLayout;
public class JFrameLayout extends JFrame{
    public JFrameLayout() {
        Container cp = getContentPane();
        // Make sure it has a FlowLayout layout manager.
        cp.setLayout(new FlowLayout());
        // now add Components to container
        cp.add(new JLabel("FlowLayout manager"));
        cp.add(new JButton("Yes!"));
        pack(); //frame size or preference size of frame
        }
    public static void main(String[] args) {
        JFrameLayout jfd=new JFrameLayout();
        jfd.setVisible(true);
        jfd.setSize(300,100);
        jfd.setTitle("JFrame Layout");
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }}
```

14.1.4 A Tabbed View of Life

Problem

These layouts don't include a tab layout, and you need one.

Solution

Use a JTabbedPane .

Discussion

The JTabbedPane class acts as a combined container and layout manager. It implements a conventional tab layout.

Example: Tab Demo

Code:

```
package com.GUI;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTabbedPane;
public class TabPaneDemo {
    protected JTabbedPane tabPane;
    public TabPaneDemo() {
        tabPane = new JTabbedPane();
        tabPane.add(new JLabel("One", JLabel.CENTER), "First");
        tabPane.add(new JLabel("Two", JLabel.CENTER), "Second");
        tabPane.add(new JLabel("Three", JLabel.CENTER),
           "third");
    }
    public static void main(String[] a) {
        JFrame f = new JFrame("Tab Demo");
        f.getContentPane().add(new TabPaneDemo().tabPane);
        f.setSize(300, 200);
        f.setVisible(true);
    }}
```

14.1.5 Action Handling: Making Buttons Work

Problem

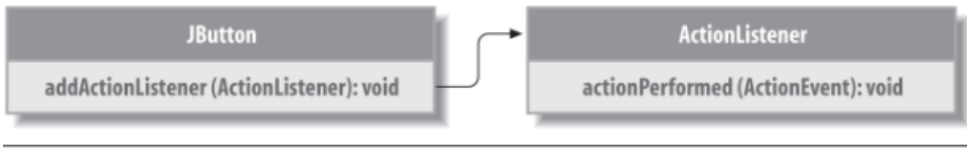
Your button doesn't do anything when the user presses it.

Solution

Add an ActionListener to do the work.

Discussion

AWT listener relationships



Example: pushing a button causes the program to print a friendly message

```
package com.GUI;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
/** Demonstrate simple use of Button */
public class ButtonDemo extends JFrame implements
    ActionListener {
     public ButtonDemo() {
          setLayout(new FlowLayout());
          JButton b1=new JButton("A button");
          add(b1);
          b1.addActionListener(this);
          setSize(300, 200);
     }
     public void actionPerformed(ActionEvent event) {
          System.out.println("Thanks for pushing my button!");
     }
     public static void main(String[] unuxed) {
          new ButtonDemo().setVisible(true);
     }}
```

Assignment

1. Demonstrate the use of JTabbedPane.

2. Demonstrate the BorderLayout to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

   Hints

   ```
   public static final int NORTH
   public static final int SOUTH
   public static final int EAST
   public static final int WEST
   public static final int CENTER
   ```

14.1.6 Action Handling Using Lambdas

Problem

You want to use Java 8's lambda expressions to simplify GUI programming.

Solution

Write the lambda expression as the argument to, for example, JComponent.addActionListener() .

Syntax: (lambda operator) -⟩ body (p) -⟩ System.out.println("One parameter:" +p);

Example: Demonstrate a JButton with Lambda Action Listeners

Code:

```
package com.GUI;
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class ButtonDemo3L extends JFrame{
    public ButtonDemo3L() {  //constructor
        super("ButtonDemo Lambda");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout());
        JButton b;
        add(b = new JButton("A button"));
        // Minimalist style
        b.addActionListener(e ->
           JOptionPane.showMessageDialog(this,
                "Thanks for pushing my first button!"));
        add(b = new JButton("Another button"));
        // Longer style, with { } around body.
        b.addActionListener(e ->
           {JOptionPane.showMessageDialog(this,
                    "Thanks for pushing my second button!");}
        );
        pack();
    }
    public static void main(String[] args) {
        new ButtonDemo3L().setVisible(true);
        }
}
```

14.1.7 Terminating a Program with "Window Close"

Problem

Nothing happens when you click the close button on the title bar of an AWT Frame . When you do this on a Swing JFrame , the window disappears but the application does not exit.

Solution

Use JFrame's **setDefaultCloseOperation()** method or **add a WindowListener** and have it exit the application.

Discussion

The Swing JFrame has a **setDefaultCloseOperation() method**, which controls the default behavior. You can pass it one of the values defined in the Swing WindowConstants class:

WindowConstants.DO_NOTHING_ON _CLOSE

Ignore the request. The window stays open. Useful for critical dialogs; probably antisocial for most "main application" type windows.

WindowConstants.HIDE_ON_CLOSE

Hide the window (default).

WindowConstants.DISPOSE_ON_CLOSE

Hide and dispose the window.

WindowConstants.EXIT_ON_CLOSE
Exit the application on close, obviating the need for a WindowListener ! Does not give you a chance to save data; for that, you need a **WindowListener**.
Assignment

1. Demonstrate JTextArea class that display text.

   Hints

   ```
   public class JTextArea extends JTextComponent
   //main code for JTextArea
     JFrame f= new JFrame();
     JTextArea area=new JTextArea("Welcome to
        javatpoint");
     area.setBounds(10,30, 200,200);
     f.add(area);
     f.setSize(300,300);
     f.setLayout(null);
     f.setVisible(true);
   ```

2. Program to demonstrate JRadioButton.
   public class JRadioButton extends JToggleButton implements Accessible

3. Set frame.dispose() on the click of a button to close JFrame.
   JFrame frame = new JFrame();
   JButton button = new JButton("Click to Close!");
   Close the JFrame on the click of the above button with Action Listener:
   button.addActionListener(e –⟩{ frame.dispose(); });

Example: Class WindowDemo puts up a JFrame and closes when you ask it to.

```
Code:
package com.GUI;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import javax.swing.WindowConstants;
public class WindowDemo extends JFrame{
    public static void main(String[] argv) {
        JFrame f = new WindowDemo();
            f.setVisible(true);
        }
public WindowDemo() {
            setSize(200, 100);
            setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
            addWindowListener(new WindowDemoAdapter());
        }
        /** Named Inner class that closes a Window. */
        class WindowDemoAdapter extends WindowAdapter {

            public void windowClosing(WindowEvent e) {
                // whimsy - close randomly, ~ 1 times in 3
                if (Math.random() > 0.666) {
                    System.out.println("Goodbye!");
                    WindowDemo.this.setVisible(false); //
                       window will close
                    WindowDemo.this.dispose(); // and be freed
                       up.
                    System.exit(0);
                }
        System.out.println("You asked me to close, but not to I
           chose.");
            }}}
Output:
You asked me to close, but not to I chose.

Goodbye!
```

14.1.8 Dialogs: When Later Just Won't Do

Problem

You need a bit of feedback from the user right now.

Solution: Use a **JOptionPane** method to show a prebuilt dialog. Or subclass **JDialog**.

Example

Code:

```
package com.GUI;
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
public class JOptionDemo extends JFrame{
    // Constructor
        JOptionDemo(String s) {
            super(s);
            Container cp = getContentPane();
            cp.setLayout(new FlowLayout());
            JButton b = new JButton("A button"); //one button
            b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    JOptionPane.showMessageDialog(
                    JOptionDemo.this,
                    "dialog1.text",
                    "dialog1.title",
                    JOptionPane.INFORMATION_MESSAGE);
                }});
            cp.add(b);   //other button
            b = new JButton("GoodBye");
            b.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                }});
            cp.add(b);
            setSize(200, 150); // the main window
            pack();
        }
        public static void main(String[] arg) {
            JOptionDemo x = new JOptionDemo("Testing 1 2 3...");
            x.setVisible(true);
        }}
```

14.1.9 Choosing a Value with JSpinner

Problem

You want to let the user choose from a fixed set of values, but do not want to use a Jlist or JComboBox because they take up too much "screen real estate."

Solution

Use a JSpinner .

Discussion

The JSpinner class lets the user click up or down to cycle through a set of values. The values can be of any type because they are managed by a helper of type SpinnerModel and displayed by another helper of type SpinnerEditor . A series of predefined SpinnerModels handle Number's, Date's, and List's (which can be arrays or Collections).

Example: Jspinner Demo

Code:

```
package com.GUI;
import java.awt.Container;
import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JSpinner;
import javax.swing.SpinnerDateModel;
import javax.swing.SpinnerListModel;

public class SpinnerDemo {
    public static void main(String[] args) {
        JFrame jf = new JFrame("It Spins");
        Container cp = jf.getContentPane();
        cp.setLayout(new GridLayout(0,1));

        // Create a JSpinner using one of the pre-defined
           SpinnerModels
        JSpinner dates = new JSpinner(new SpinnerDateModel());
        cp.add(dates);

        // Create a JSPinner using a SpinnerListModel.
        String[] data = { "One", "Two", "Three" };
        JSpinner js = new JSpinner(new SpinnerListModel(data));
        cp.add(js);
        jf.setSize(200, 80);
        jf.setVisible(true);
    }}
```

14.1.10 Choosing a File with JFileChooser

Problem

You want to allow the user to select a file by name using a traditional windowed file dialog.

Solution

Use a JFileChooser .

Discussion

- The JFileChooser dialog provides a fairly standard file chooser. It has elements of both a Windows chooser and a Mac chooser, with more resemblance to the former than the latter.
- If you want to have control over which files appear, you need to provide one or more FileFilter subclasses.
- Each FileFilter subclass instance passed into the JFileChooser's addChoosableFileFilter() method becomes a selection in the chooser's Files of Type: choice. The default is All Files (.).

Example: Choosing a File

Code:

```
package com.GUI;
import java.io.File;
import javax.swing.JButton;
import javax.swing.JfileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class FileChooseDemo extends JPanel{
    public FileChooseDemo(JFrame f) {
        final JFrame frame = f;
        final JFileChooser chooser = new JFileChooser();
          JButton b = new JButton("Choose file...");
          add(b);
          b.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
              int returnVal = chooser.showOpenDialog(frame);
              if (returnVal == JFileChooser.APPROVE_OPTION) {
                  File file = chooser.getSelectedFile();
                  System.out.println("You chose a " +
                  (file.isFile() ? "file" : "directory") +
                  " named: " + file.getPath());
              } else {
                  System.out.println("You did not choose a
                     filesystem object.");
                }
            }
          });}
    public static void main(String[] args) {
        JFrame f = new JFrame("JFileChooser Demo");
        f.getContentPane().add(new FileChooseDemo(f));
        f.pack();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }}
Output:
You chose a file named: /home/chukhu/Desktop/JavadocDemo.java
You did not choose a filesystem object.
```

Assignment

1. Demonstrate use of JOptionPane.

   The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. **public class JOptionPane extends JComponent implements Accessible**

Solution

```
    import javax.swing.*;
    public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){f=new JFrame();
String name=JOptionPane.showInputDialog(f,"Enter Name");}
public static void main(String[] args) {new
   OptionPaneExample();}}
```

2. Demonstrate JScrollBar.
   The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.
   public class JScrollBar extends JComponent implements Adjustable, Accessible

14.1.11 Choosing a Color

Problem
You want to allow the user to select a color from all the colors available on your computer.
Solution: Use Swing's JColorChooser .
Discussion:

- Construct it and place it in a panel.
- Call its createDialog() and get a JDialog back.
- Call its showDialog() and get back the chosen color.

Methods of operating the chooser:
• Swatches mode: The user can pick from one of a few hundred color variants.
• HSB mode: The user picks one of Hue, Saturation, or Brightness, a standard way of representing color value. The user can adjust each value by slider.
• RGB mode: The user picks Red, Green, and Blue components by sliders.

Example

```
Code:
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JColorChooserDemo extends JFrame {
/** A canvas to display the color in. */
protected JLabel demo;
/** Constructor - set up the entire GUI for this program */
public JColorChooserDemo() {
super("Swing Color Demo illustrated");
    Container cp = getContentPane();
    JButton jButton;
    cp.add(jButton = new JButton("Change Color..."),
       BorderLayout.NORTH); //button
    jButton.setToolTipText(""); //text
    jButton.addActionListener(new ActionListener() {  //action
       on button
public void actionPerformed(ActionEvent actionEvent)
{
        Color ch = JColorChooser.showDialog(    //show the
           dialog of color
        JColorChooserDemo.this,
// parent
        "Swing Demo Color Popup",
//title
        demo.getForeground());  //color in which text is shown
        System.out.println("Your selected color is " + ch);
           //default
if (ch != null) {
    demo.setForeground(ch);
    demo.repaint();  //repaint the color of text
    }}});
        cp.add(BorderLayout.CENTER, demo =
                new JLabel("Your One True Color", JLabel.LEFT));
                demo.setToolTipText("This is the last color you
                   chose");
                pack();
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);}
public static void main(String[] argv) {
new JColorChooserDemo().setVisible(true);
}}
Output:
Your selected color is java.awt.Color[r=255,g=51,b=51]
Your selected color is java.awt.Color[r=153,g=204,b=0]
Your selected color is java.awt.Color[r=255,g=0,b=51]
```

14.1.12 Formatting Jcomponents with HTML

Problem

You want more control over the formatting of text in JLabel and friends. **Solution**

Use HTML in the text of the component.

Discussion

The Swing components that display text, such as JLabel , format the text as HTML—instead of as plain text—if the first six characters are the obvious tag <html> . The program JLabelHTMLDemo just puts up a JLabel formatted using this Java code:

Formatting Jcomponents with HTML

```
Code:
package Graphics;
import java.awt.BorderLayout;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;

public class JLabelHTMLDemo extends JFrame{
    /** Construct the object including its GUI */
    public JLabelHTMLDemo() {
    super("JLabelHTMLDemo");
    Container cp = getContentPane();
    JButton component = new JButton(
    "<html>" +
    "<body bgcolor='white'>" +
    "<h1><font color='red'>Welcome</font></h1>" +
    "<p>This button will be formatted according to the usual " +
    "HTML rules for formatting of paragraphs.</p>" +
    "</body></html>");
    component.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
    System.out.println("Thank you!");
    }
    });
    cp.add(BorderLayout.CENTER, component);
    setSize(200, 400);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
    new JLabelHTMLDemo().setVisible(true);
    }}
```

Assignment

1. Using a JLabel and pass in HTML for the text.

Centering a Main Window:

```
JFrame frame = new JFrame();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JLabel label = new JLabel("<html>bold <br>
   plain</html>");
frame.add(label);
frame.setSize(300, 200);
frame.setVisible(true);
```

14.1.13 Centering a Main Window:

Problem

You want to change the look and feel of an application.

Solution

Use the static UIManager.setLookAndFeel() method. Maybe.

Centering a Main Window:

Code:

```
package Graphics;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
public class LNFSwitcher {
    protected JFrame theFrame; /** The frame. */
    protected Container cp; /** Its content pane */
    /** Start with the Java look-and-feel, if possible */
    final static String PREFERREDLOOKANDFEELNAME =
    "javax.swing.plaf.metal.MetalLookAndFeel";
    protected String curLF = PREFERREDLOOKANDFEELNAME;
    protected JRadioButton previousButton;
    /** Construct a program... */
    public LNFSwitcher() {
        super();
        theFrame = new JFrame("LNF Switcher");
        theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cp = theFrame.getContentPane();
        cp.setLayout(new FlowLayout());
        ButtonGroup bg = new ButtonGroup();
        JRadioButton bJava = new JRadioButton("Java");
        bJava.addActionListener(new LNFSetter(
        "javax.swing.plaf.metal.MetalLookAndFeel", bJava));
        bg.add(bJava);
        cp.add(bJava);
        JRadioButton bMSW = new JRadioButton("MS-Windows");
        bMSW.addActionListener(new LNFSetter(
        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel",
           bMSW));
        bg.add(bMSW);
        cp.add(bMSW);
        JRadioButton bMotif = new JRadioButton("Motif");
        bMotif.addActionListener(new LNFSetter(
        "com.sun.java.swing.plaf.motif.MotifLookAndFeel",
           bMotif));
        bg.add(bMotif);
        cp.add(bMotif);
```

continue...

```
JRadioButton bMac = new JRadioButton("Sun-MacOS");
        bMac.addActionListener(new LNFSetter(
        "com.sun.java.swing.plaf.mac.MacLookAndFeel", bMac));
        bg.add(bMac);
        cp.add(bMac);
        String defaultLookAndFeel =
           UIManager.getSystemLookAndFeelClassName();
        // System.out.println(defaultLookAndFeel);
        JRadioButton bDefault = new JRadioButton("Default");
        bDefault.addActionListener(new LNFSetter(
        defaultLookAndFeel, bDefault));
        bg.add(bDefault);
        cp.add(bDefault);
        (previousButton = bDefault).setSelected(true);
        theFrame.pack();}
        /* Class to set the Look and Feel on a frame */
        class LNFSetter implements ActionListener {
        String theLNFName;
        JRadioButton thisButton;
        /** Called to setup for button handling */
        LNFSetter(String lnfName, JRadioButton me) {
            theLNFName = lnfName;
            thisButton = me;                          }
            /** Called when the button actually gets pressed. */
            public void actionPerformed(ActionEvent e) {
            try {
            UIManager.setLookAndFeel(theLNFName);
            SwingUtilities.updateComponentTreeUI(theFrame);
            theFrame.pack();
            } catch (Exception evt) {
            JOptionPane.showMessageDialog(null,
            "setLookAndFeel didn't work: " + evt,
            "UI Failure", JOptionPane.INFORMATION_MESSAGE);
            previousButton.setSelected(true); // reset the GUI
               to agree
            }
            previousButton = thisButton;
            }}
            public static void main(String[] argv) {
            LNFSwitcher o = new LNFSwitcher();
            o.theFrame.setVisible(true);
            }}
```

14.1.14 Program : Custom Front Chooser

Problem

You want to allow the user to select a font, but standard Java doesn't yet include a Font Chooser dialog. **Solution**

Use my FontChooser dialog class.

Discussion

As we saw in Recipe 12.3, you can manually select a font by calling the java.awt.Font class constructor, passing in the name of the font, the type you want (plain, bold, italic, or bold+italic), and the point size: Font f = new Font("Helvetica", Font.BOLD, 14); setfont(f);

Custom Front Chooser

Code:

```
public class FontChooser extends JDialog {
private static final long serialVersionUID =
    5363471384675038069L;
public static final String DEFAULT_TEXT = "Lorem ipsem dolor";
/** The font the user has chosen */
protected Font resultFont = new Font("Serif", Font.PLAIN, 12);
/** The resulting font name */
protected String resultName;
/** The resulting font size */
protected int resultSize;
/** The resulting boldness */
protected boolean isBold;
/** The resulting italicness */
protected boolean isItalic;
/** Display text */
protected String displayText = DEFAULT_TEXT;
/** The font name chooser */
protected JList fontNameChoice;
/** The font size chooser */
protected JList fontSizeChoice;
/** The bold and italic choosers */
JCheckBox bold, italic;
/** The list of font sizes */
protected Integer fontSizes[] = {
8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, 36, 40, 48, 60, 72
};
/** The index of the default size (e.g., 14 point == 4) */
```

continue...

```
  protected static final int DEFAULT_SIZE = 4;
/** The font display area.*/
protected JLabel previewArea;
/** Construct a FontChooser -- Sets title and gets
* array of fonts on the system. Builds a GUI to let
* the user choose one font at one size.
*/
public FontChooser(JFrame f) {
super(f, "Font Chooser", true);
Container cp = getContentPane();
JPanel top = new JPanel();
top.setBorder(new TitledBorder(new EtchedBorder(), "Font"));
top.setLayout(new FlowLayout());
String[] fontList =
   GraphicsEnvironment.getLocalGraphicsEnvironment().
getAvailableFontFamilyNames();
fontNameChoice = new JList(fontList);
top.add(new JScrollPane(fontNameChoice));
fontNameChoice.setVisibleRowCount(fontSizes.length);
fontNameChoice.setSelectedValue("Serif", true);
fontSizeChoice = new JList(fontSizes);
top.add(fontSizeChoice);
fontSizeChoice.setSelectedIndex(fontSizes.length * 3 / 4);
cp.add(top, BorderLayout.NORTH);
JPanel attrs = new JPanel();
top.add(attrs);
attrs.setLayout(new GridLayout(0,1));
attrs.add(bold =new JCheckBox("Bold", false));
attrs.add(italic=new JCheckBox("Italic", false));
// Make sure that any change to the GUI will trigger a font
   preview.
ListSelectionListener waker = new ListSelectionListener() {
public void valueChanged(ListSelectionEvent e) {
previewFont();
}};
bold.addItemListener(waker2);
italic.addItemListener(waker2);
previewArea = new JLabel(displayText, JLabel.CENTER);
previewArea.setSize(200, 50);
```

continue...

```
cp.add(previewArea, BorderLayout.CENTER);
JPanel bot = new JPanel();
JButton okButton = new JButton("Apply");
bot.add(okButton);
okButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
previewFont();
dispose();
setVisible(false);
}});
JButton canButton = new JButton("Cancel");bot.add(canButton);
canButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
// Set all values to null. Better: restore previous.
resultFont = null;
resultName = null;
resultSize = 0;
isBold = false;
isItalic = false;
dispose();
setVisible(false);
}});
cp.add(bot, BorderLayout.SOUTH);
previewFont(); // ensure view is up to date!
pack();
setLocation(100, 100);}
/** Called from the action handlers to get the font info,
* build a font, and set it.*/
protected void previewFont() {
resultName = (String)fontNameChoice.getSelectedValue();
String resultSizeName =
   fontSizeChoice.getSelectedValue().toString();
int resultSize = Integer.parseInt(resultSizeName);
isBold = bold.isSelected();
isItalic = italic.isSelected();
int attrs = Font.PLAIN;
if (isBold) attrs = Font.BOLD;
if (isItalic) attrs |= Font.ITALIC;
resultFont = new Font(resultName, attrs, resultSize);
// System.out.println("resultName = " + resultName + "; " +
//"resultFont = " + resultFont);
previewArea.setFont(resultFont);
pack();}
/** Retrieve the selected font name. */
public String getSelectedName() {
return resultName;              }
```

continue...

```
/** Retrieve the selected size */
public int getSelectedSize() {
return resultSize;           }
/** Retrieve the selected font, or null */
public Font getSelectedFont() {
return resultFont;           }
public String getDisplayText() {
return displayText;          }
public void setDisplayText(String displayText) {
this.displayText = displayText;
previewArea.setText(displayText);
previewFont();                                  }
public JList getFontNameChoice() {
return fontNameChoice;          }
public JList getFontSizeChoice() {
return fontSizeChoice;          }
public boolean isBold() {
return isBold;          }
public boolean isItalic() {
return isItalic;
}}
```

Assignment

1. Creat Edit menu for Notepad:Using JMenuBar, JMenu and JMenuItem.

2. Program to select a color from all color available in computer using Swings's JColorChooser.

3. Demonstrate the slider by using JSlider, a user can select a value from a specific range.

   Solution

   ```
   import javax.swing.*;
   public class SliderExample extends JFrame{
   public SliderExample() {
   JSlider slider = new JSlider(JSlider.HORIZONTAL, 0,
      50, 25);
   slider.setMinorTickSpacing(2);
   slider.setMajorTickSpacing(10);
   slider.setPaintTicks(true);
   slider.setPaintLabels(true);
   JPanel panel=new JPanel();
   panel.add(slider);
   add(panel);
   }
   public static void main(String s[]) {
   SliderExample frame=new SliderExample();
   frame.pack();
   frame.setVisible(true);
   }  }
   ```

4. Program to use JFileChooser to select directory only.

Hints

```
chooser = new JFileChooser();
chooser.setCurrentDirectory(new java.io.File("."));
chooser.setDialogTitle(choosertitle);
chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
// disable the "All files" option.
chooser.setAcceptAllFileFilterUsed(false);
    if (chooser.showOpenDialog(this) ==
       JFileChooser.APPROVE_OPTION){
      System.out.println("getCurrentDirectory(): "
         +  chooser.getCurrentDirectory());
      System.out.println("getSelectedFile() : "
         +  chooser.getSelectedFile());
      }
    else {
      System.out.println("No Selection ");
      }}
```