COP 4610L: Applications in the Enterprise
Fall 2005

GUI Components: Part 2

Instructor :	Mark Llewellyn
	markl@cs.ucf.edu
	CSB 242, 823-2790
	http://www.cs.ucf.edu/courses/cop4610L/fall2005

School of Computer Science
University of Central Florida

Mouse Event Handling

- The `MouseListener` and the `MouseMotionListener` event-listener interfaces are designed to handle mouse events. Mouse events can be trapped for any GUI component that derives from `java.awt.Component`.
- Package `javax.swing.event` contains interface `MouseInputListener`, which extends the two interfaces to create a single interface which contains all the methods of both interfaces.
- The `MouseListener` and the `MouseMotionListener` methods are called with the mouse interacts with a Component if the appropriate event-listener objects are registered for that Component.

Mouse Event Handling (cont.)

- Each of the mouse event-handling methods takes a `MouseEvent` object as its argument. A `MouseEvent` object contains information about the mouse event that occurred, including the x- and *y*-coordinates of the location where the event occurred.
- These coordinates are measured form the upper-left corner of the GUI component on which the event occurred.
  - The *x*-coordinates begin a 0 and increase from left to right.
  - The *y*-coordinates begin an 0 and increase from top to bottom.
- In addition, the methods and constants of class InputEvent (MouseEvent's superclass) enable an application to determine which mouse button was clicked.

MouseListener and MouseMotionListener Interface Methods

Methods of Interface **`MouseListener`**

`public void mousePressed(MouseEvent event)` called when a mouse button is pressed while the mouse cursor is on a component.

`public void mouseClicked(MouseEvent event)` called when a mouse button is pressed and released while the mouse cursor remains stationary on a component. This event is always preceded by a call to `mousePressed`.

`public void mouseReleased(MouseEvent event)`called when a mouse button is released after being pressed. This event is always preceded by a call to `mousePressed` and one or more calls to `mouseDragged`.

`public void mouseEnetered(MouseEvent event)` called when the mouse cursor enters the bounds of a component.

`public void mouseExited(MouseEvent event)`called when the mouse cursor leaves the bounds of a component.

MouseListener and MouseMotionListener Interface Methods

Methods of Interface `MouseMotionListener`

`public void mouseDragged(MouseEvent event)` called when the mouse button is pressed while the mouse cursor is on a component and the mouse is moved while the mouse button remains pressed. This event is always preceded by a call to `mousePressed`. All drag events are sent to the component on which the user began to drag the mouse.

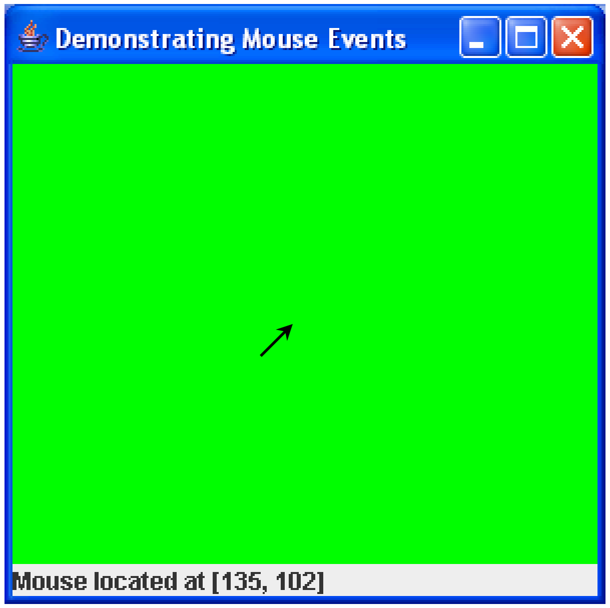
`public void mouseMoved(MouseEvent event)` called when the mouse is moved when the mouse cursor is on a component. All move events are sent to the component over which the mouse is currently positioned.

Java also provides interface `MouseWheelListener` to enable applications to respond to the rotation of a mouse wheel. This interface declares method `mouseWheelMoved`, which receives a `MouseWheelEvent` as its argument. Class `MouseWheelEvent` (a subclass of `MouseEvent`) contains methods that enable the event handler to obtain information about the amount of wheel rotation.

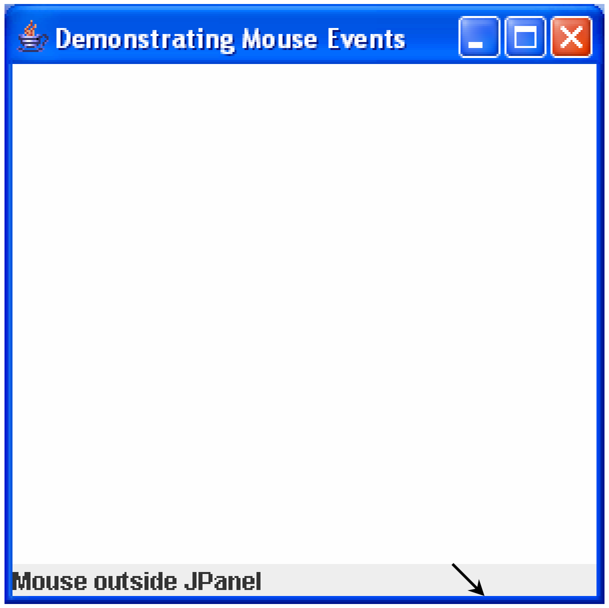
GUI Example illustrating Mouse Events

```
// Demonstrating mouse events.
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class MouseTrackerFrame extends JFrame
{
   private JPanel mousePanel; // panel in which mouse events will occur
   private JLabel statusBar; // label that displays event information
   // MouseTrackerFrame constructor sets up GUI and registers mouse event handlers
   public MouseTrackerFrame()
   {
      super( "Demonstrating Mouse Events" );
      mousePanel = new JPanel(); // create panel
      mousePanel.setBackground( Color.WHITE ); // set background color
      add( mousePanel, BorderLayout.CENTER ); // add panel to JFrame
      statusBar = new JLabel( "Mouse outside JPanel" );
      add( statusBar, BorderLayout.SOUTH ); // add label to JFrame
      // create and register listener for mouse and mouse motion events
      MouseHandler handler = new MouseHandler();
      mousePanel.addMouseListener( handler );
      mousePanel.addMouseMotionListener( handler );
   } // end MouseTrackerFrame constructor
```

```
private class MouseHandler implements MouseListener,
      MouseMotionListener
   {
      // MouseListener event handlers handle event when mouse released immediately
     // after press
      public void mouseClicked( MouseEvent event )
      {
         statusBar.setText( String.format( "Clicked at [%d, %d]",
            event.getX(), event.getY() ) );
      } // end method mouseClicked

      // handle event when mouse pressed
      public void mousePressed( MouseEvent event )
      {
         statusBar.setText( String.format( "Pressed at [%d, %d]",
            event.getX(), event.getY() ) );
      } // end method mousePressed

      // handle event when mouse released after dragging
      public void mouseReleased( MouseEvent event )
      {
         statusBar.setText( String.format( "Released at [%d, %d]",
            event.getX(), event.getY() ) );
      } // end method mouseReleased
      // handle event when mouse enters area
      public void mouseEntered( MouseEvent event )
      {
         statusBar.setText( String.format( "Mouse entered at [%d, %d]",
            event.getX(), event.getY() ) );
         mousePanel.setBackground( Color.GREEN );
      } // end method mouseEntered
      // handle event when mouse exits area
      public void mouseExited( MouseEvent event )
      {
         statusBar.setText( "Mouse outside JPanel" );
         mousePanel.setBackground( Color.WHITE );
      } // end method mouseExited
      // MouseMotionListener event handlers handle event when user drags mouse with
     // button pressed
      public void mouseDragged( MouseEvent event )
      {
         statusBar.setText( String.format( "Dragged at [%d, %d]",
            event.getX(), event.getY() ) );
      } // end method mouseDragged
      // handle event when user moves mouse
      public void mouseMoved( MouseEvent event )
      {
         statusBar.setText( String.format( "Mouse located at [%d, %d]",
            event.getX(), event.getY() ) );
      } // end method mouseMoved
   } // end inner class MouseHandler
} // end class MouseTrackerFrame
```

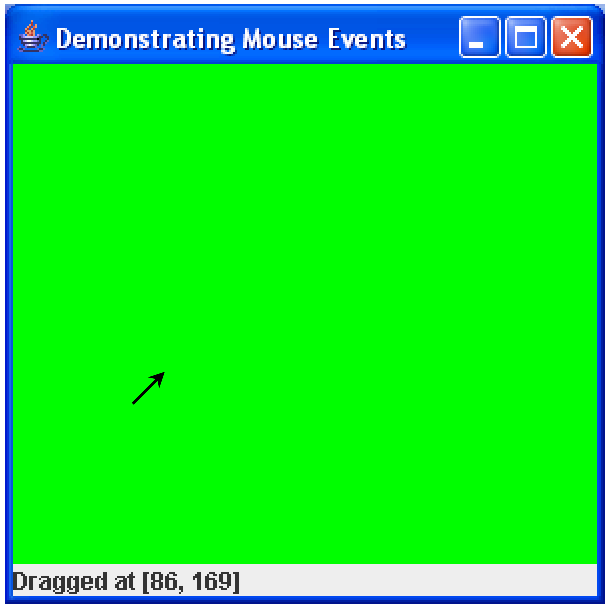


Mouse located at [135, 102]

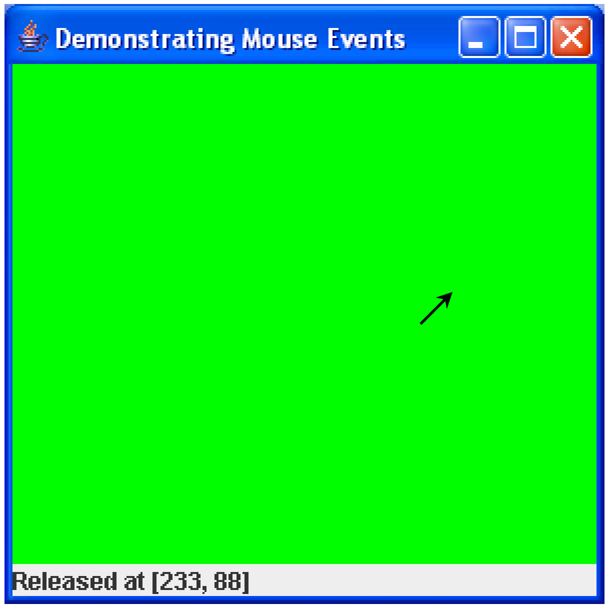


Mouse outside JPanel

Some sample screen shots showing capture of mouse motion



Dragged at [86, 169]



Released at [233, 88]

Adapter Classes

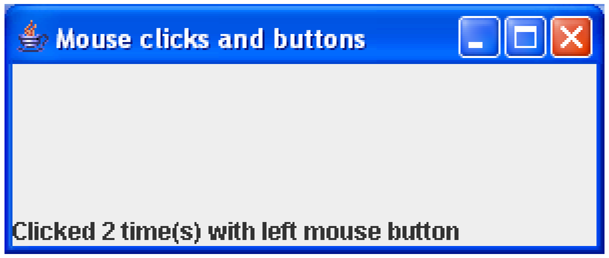
- Many event-listener interfaces, such as `MouseListener` and `MouseMotionListener`, contain multiple methods. It is not always a good thing to declare every methods in an event-listener interface.
  - For example, an application may need only the `mouseClicked` handler from `MouseListener` or the `mouseDragged` handler from `MouseMotionListener`. Interface `WindowListener` specifies seven window event-handling methods.
- For many of the listener interfaces that have multiple methods, packages `java.awt.event` and `javax.swing.event` provide event-listener adapter classes.
- An adapter class implements an interface and provides a default implementation (with an empty method body) of each method in the interface. You can extend an adapter class to inherit the default implementation of every method and subsequently override only the method(s) that you need for event handling.

Event-Adapter Classes in `java.awt.event`

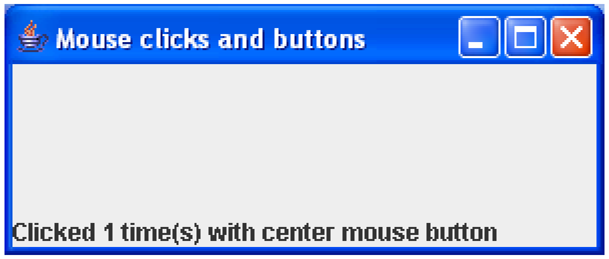
Event-adapter class in `java.awt.event`	Implements interface
`ComponentAdapter`	`ComponentListener`
`ContainerAdapter`	`ContainerListener`
`FocusAdapter`	`FocusListener`
`KeyAdapter`	`KeyListener`
`MouseAdapter`	`MouseListener`
`MouseMotionAdapter`	`MouseMotionListener`
`WindowAdapter`	`WindowListener`

GUI Example illustrating Adapter Class for Mouse Events

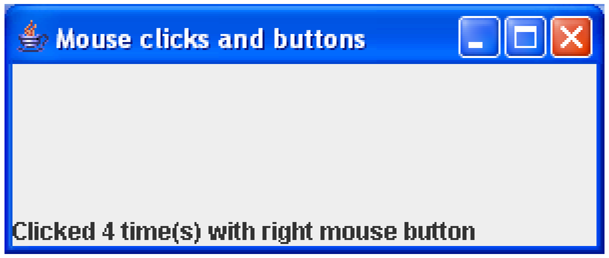
```
// Demonstrating mouse clicks and distinguishing between mouse buttons.
import java.awt.BorderLayout;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.JFrame;
import javax.swing.JLabel;

public class MouseDetailsFrame extends JFrame
{
   private String details; // String representing movement
   private JLabel statusBar; // JLabel that appears at bottom of window
   // constructor sets title bar String and register mouse listener
   public MouseDetailsFrame()
   {
      super( "Mouse clicks and buttons" );
      statusBar = new JLabel( "Click the mouse" );
      add( statusBar, BorderLayout.SOUTH );
      addMouseListener( new MouseClickHandler() ); // add handler
   } // end MouseDetailsFrame constructor
// inner class to handle mouse events
  private class MouseClickHandler extends MouseAdapter
  {
    // handle mouse click event and determine which button was pressed
    public void mouseClicked( MouseEvent event )
    {
      int xPos = event.getX(); // get x position of mouse
      int yPos = event.getY(); // get y position of mouse

      details = String.format( "Clicked %d time(s)",
        event.getClickCount() );

      if ( event.isMetaDown() ) // right mouse button
        details += " with right mouse button";
      else if ( event.isAltDown() ) // middle mouse button
        details += " with center mouse button";
      else // left mouse button
        details += " with left mouse button";

      statusBar.setText( details ); // display message in statusBar
    } // end method mouseClicked
  } // end private inner class MouseClickHandler
} // end class MouseDetailsFrame
```



Clicked 2 time(s) with left mouse button



Clicked 1 time(s) with center mouse button



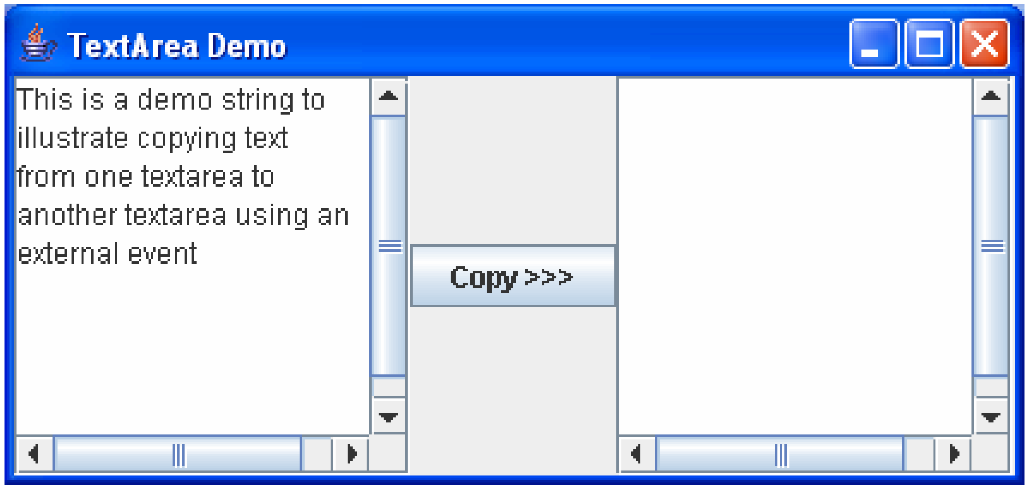
Clicked 4 time(s) with right mouse button

JTextArea

- A `JTextArea` provides an area for manipulating multiple lines of text. Like class `JTextField`, `JTextArea` is a subclass of `JTextComponent`.
- Recall that `JTextComponent` declares common methods for `JTextFields`, `JTextAreas`, and several other text-based GUI components.
- The next example illustrates the use of a `JTextArea` and is similar in nature to the previous example involving multiple selection lists.
- Note that by default, a `JTextArea` does not automatically wrap lines. To turn line wrapping on for a `JTextArea` invoke `JTextArea` method `setLineWrap` with a *true* argument.

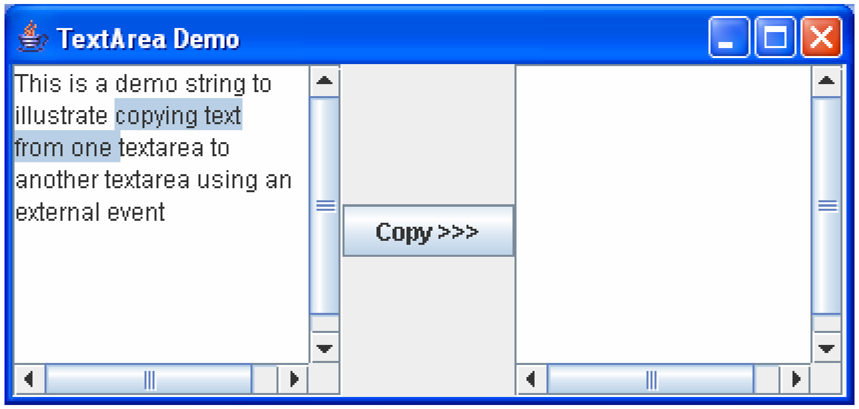
GUI Example illustrating JTextArea Class

```
// Example of JTextArea - Copying selected text from one
// textarea to another.
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JScrollPane;

public class TextAreaFrame extends JFrame
 {
   private JTextArea textArea1; // displays demo string
   private JTextArea textArea2; // highlighted text is copied here
   private JButton copyJButton; // initiates copying of text

   // no-argument constructor
   public TextAreaFrame()
 {
      super( "TextArea Demo" );
      Box box = Box.createHorizontalBox(); // create box
      String demo = "This is a demo string to\n" +
         "illustrate copying text\nfrom one textarea to \n" +
         "another textarea using an\nexternal event\n";
      textArea1 = new JTextArea( demo, 10, 15 ); // create textarea1
      box.add( new JScrollPane( textArea1 ) ); // add scrollpane
      copyJButton = new JButton( "Copy >>>" ); // create copy button
   box.add( copyJButton ); // add copy button to box

   copyJButton.addActionListener(
   new ActionListener() // anonymous inner class
     {
       // set text in textArea2 to selected text from textArea1
       public void actionPerformed( ActionEvent event )  {
         textArea2.setText( textArea1.getSelectedText() );
       } // end method actionPerformed
     } // end anonymous inner class
   ); // end call to addActionListener

   textArea2 = new JTextArea( 10, 15 ); // create second textarea
   textArea2.setEditable( false ); // disable editing
   box.add( new JScrollPane( textArea2 ) ); // add scrollpane
   add( box ); // add box to frame

 } // end TextAreaFrame constructor
} // end class TextAreaFrame
```



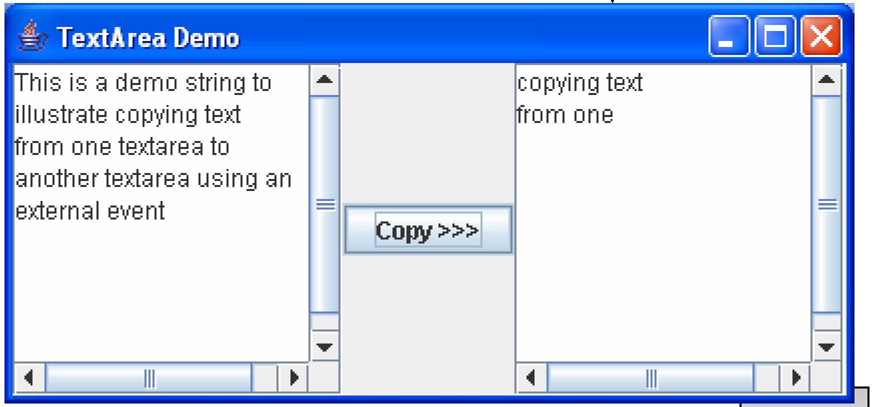
This is a demo string to illustrate copying text from one textarea to another textarea using an external event

Initial GUI



This is a demo string to illustrate copying text from one textarea to another textarea using an external event

User selects text to be copied.
Clicking “COPY” button copies text to second JTextArea



This is a demo string to illustrate copying text from one textarea to another textarea using an external event

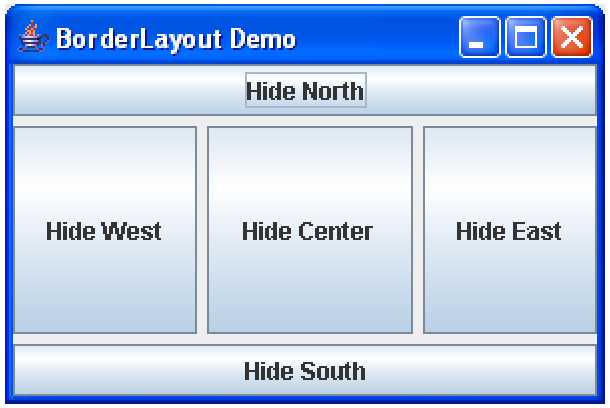
copying text
from one

Another Layout Manager: `BorderLayout`

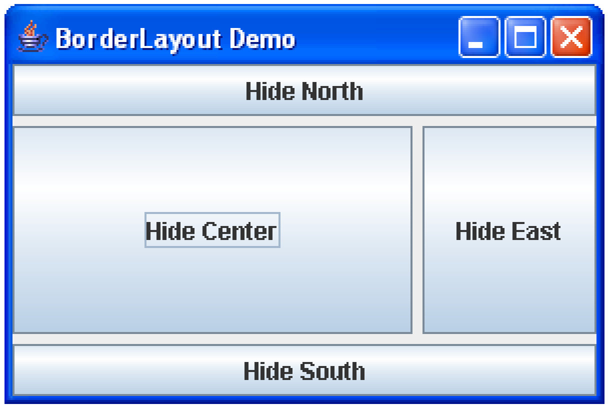
- The `BorderLayout` manager, which is the default layout manager for `JFrame` windows, arranges components into five regions: NORTH, SOUTH, EAST, WEST, and CENTER.
- NORTH corresponds to the top of the container.
- A `BorderLayout` limits a container to containing at most 5 components – one in each region. However, the component in each region can be a container to which other components are attached.
- The EAST and WEST regions expand vertically between the NORTH and SOUTH regions and are as wide as the components placed in those regions.
- The example on the next page illustrates the `BorderLayout` manager.

GUI Example illustrating BorderLayout Manager

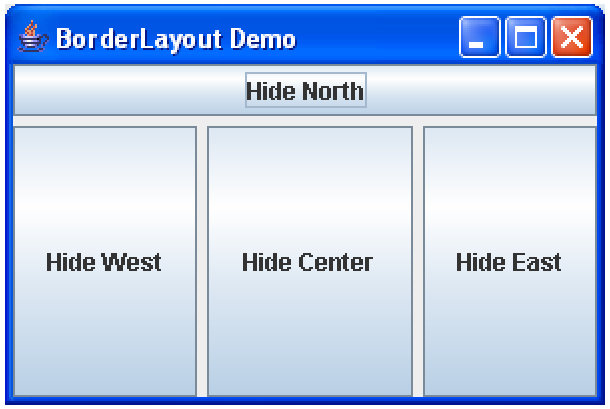
```
// Demonstrating BorderLayout.
import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.JFrame;
import javax.swing.JButton;
public class BorderLayoutFrame extends JFrame implements ActionListener
{
   private JButton buttons[]; // array of buttons to hide portions
   private final String names[] = { "Hide North", "Hide South",
      "Hide East", "Hide West", "Hide Center" };
   private BorderLayout layout; // borderlayout object
   // set up GUI and event handling
   public BorderLayoutFrame()
   {
      super( "BorderLayout Demo" );
      layout = new BorderLayout( 5, 5 ); // 5 pixel gaps
      setLayout( layout ); // set frame layout
      buttons = new JButton[ names.length ]; // set size of array
      // create JButtons and register listeners for them
      for ( int count = 0; count < names.length; count++ )
      {
         buttons[ count ] = new JButton( names[ count ] );
         buttons[ count ].addActionListener( this );
      } // end for
      add( buttons[ 0 ], BorderLayout.NORTH ); // add button to north
      add( buttons[ 1 ], BorderLayout.SOUTH ); // add button to south
      add( buttons[ 2 ], BorderLayout.EAST ); // add button to east
      add( buttons[ 3 ], BorderLayout.WEST ); // add button to west
      add( buttons[ 4 ], BorderLayout.CENTER ); // add button to center
   } // end BorderLayoutFrame constructor

   // handle button events
   public void actionPerformed( ActionEvent event )
   {
      // check event source and layout content pane correspondingly
      for ( JButton button : buttons )
      {
         if ( event.getSource() == button )
            button.setVisible( false ); // hide button clicked
         else
            button.setVisible( true ); // show other buttons
      } // end for

      layout.layoutContainer( getContentPane() ); // layout content pane
   } // end method actionPerformed
} // end class BorderLayoutFrame
```



Initial GUI



GUI after clicking on “Hide West” button



GUI after clicking on “Hide South” button

Advanced GUIs

- In the notes up to this point, we have examined a number of different capabilities for GUI programming that are available in Java. Most of the examples have simply illustrated the some of the options which are available for designing and manipulating GUIs.

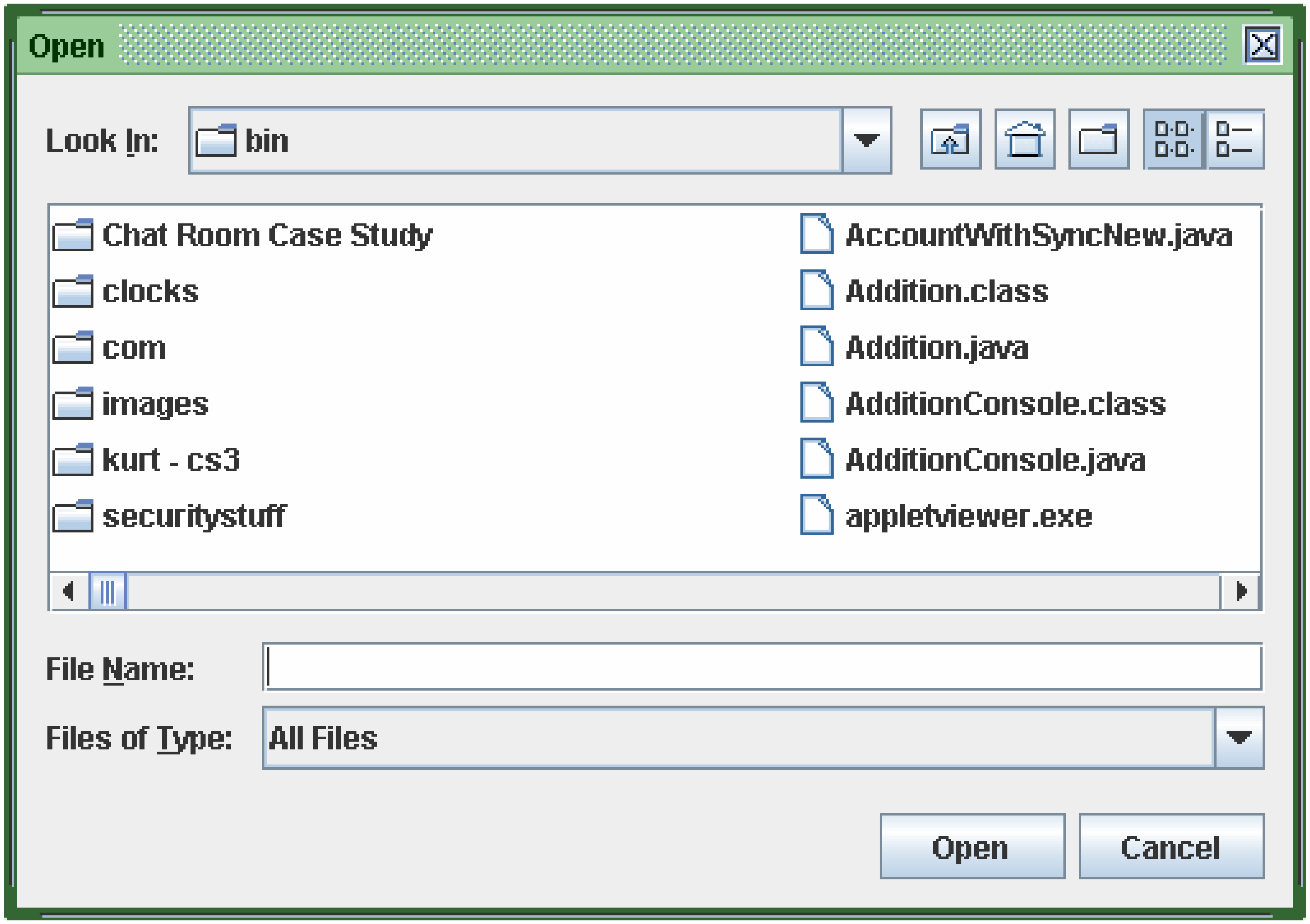
- At this point, we'll begin to look at more sophisticated applications for the GUIs which are more along the lines of what you will be programming in this course.

- As before, many of the examples in the notes will simply illustrate one option from many that are available. I encourage you to look at the Java documentation and experiment either by modifying the code from the notes or constructing your own GUIs using some of these additional options.

File Choosers

- File choosers provide a GUI for navigating the file system and then selecting a file or directory from a list (or by directly entering the name of a file or directory).
- The `JFileChooser` API makes it easy to bring up open and save dialogs. The look and feel determines what the standard dialogs look like and how they differ.
- In the Java look and feel, the save dialog looks the same as the open dialog except for the title on the dialog's window and the text on the button that approves the operation.
- The next slide illustrates the Java look and feel's standard open dialog.

Standard Java Look and Feel Open File Dialog



Chat Room Case Study
clocks
com
images
kurt - cs3
securitystuff
AccountWithSyncNew.java
Addition.class
Addition.java
AdditionConsole.class
AdditionConsole.java
appletviewer.exe

File Name:
Files of Type: All Files

File Chooser Demo

- The code on the following page demonstrates some of the features of the JFileChooser open and save dialogs.
- Try some of the options which are listed in the comments in the code.

FileChooserDemo

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

/*
 * FileChooserDemo.java is a 1.5.0 application that uses these files:
 *   images/Open16.gif
 *   images/Save16.gif
 */
public class FileChooserDemo extends JPanel implements ActionListener {
    static private final String newline = "\n";
    JButton openButton, saveButton;
    JTextArea log;
    JFileChooser fc;

    public FileChooserDemo() {
        super(new BorderLayout());

        //Create the log first, because the action listeners need to refer to it.
        log = new JTextArea(5,20);
        log.setMargin(new Insets(5,5,5,5));
        log.setEditable(false);
        JScrollPane logScrollPane = new JScrollPane(log);
```

```
//Create a file chooser
        fc = new JFileChooser(".");  //this constructor allows you to specify the directory to be opened
                                     // "." is the current default directory, ".." would be the parent of the
                                     //default or current directory.
        //Uncomment one of the following lines to try a different
        //file selection mode.  The first allows just directories
        //to be selected (and, at least in the Java look and feel,
        //shown).  The second allows both files and directories
        //to be selected.  If you leave these lines commented out,
        //then the default mode (FILES_ONLY) will be used.
        //
        //fc.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        //fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);

        //Create the open button.  We use the image from the JLF
        //Graphics Repository (but we extracted it from the jar).
        openButton = new JButton("Open a File...", createImageIcon("images/Open16.gif"));
        openButton.addActionListener(this);

        // Create the save button.  We use the image from the JLF
        // Graphics Repository (but we extracted it from the jar).
        saveButton = new JButton("Save a File...", createImageIcon("images/Save16.gif"));
        saveButton.addActionListener(this);

        // For layout purposes, put the buttons in a separate panel
        JPanel buttonPanel = new JPanel(); //use FlowLayout
        buttonPanel.add(openButton);
        buttonPanel.add(saveButton);
```

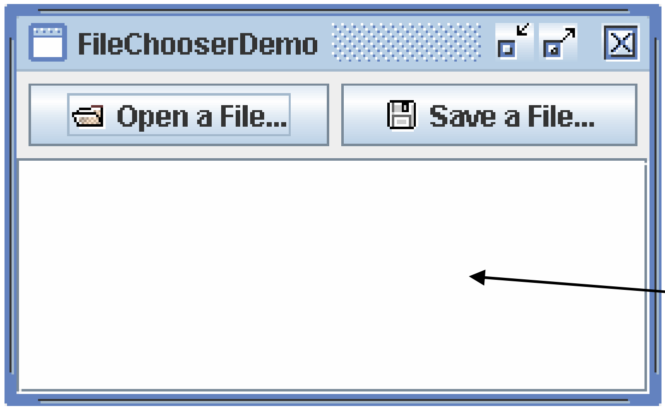
Change the argument to the file chooser constructor to set the default directory which is opened.

Uncomment these lines to change the file selection mode.

```
        //Add the buttons and the log to this panel.
        add(buttonPanel, BorderLayout.PAGE_START);
        add(logScrollPane, BorderLayout.CENTER);
    }
    public void actionPerformed(ActionEvent e) {
        //Handle open button action.
        if (e.getSource() == openButton) {
            int returnVal = fc.showOpenDialog(FileChooserDemo.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File file = fc.getSelectedFile();
                //This is where a real application would open the file.
                log.append("Opening: " + file.getName() + "." + newline);
            } else {
                log.append("Open command cancelled by user." + newline);
            }
            log.setCaretPosition(log.getDocument().getLength());

        //Handle save button action.
        } else if (e.getSource() == saveButton) {
            int returnVal = fc.showSaveDialog(FileChooserDemo.this);
            if (returnVal == JFileChooser.APPROVE_OPTION) {
                File file = fc.getSelectedFile();
                //This is where a real application would save the file.
                log.append("Saving: " + file.getName() + "." + newline);
            } else {
                log.append("Save command cancelled by user." + newline);
            }
            log.setCaretPosition(log.getDocument().getLength());
        }
```

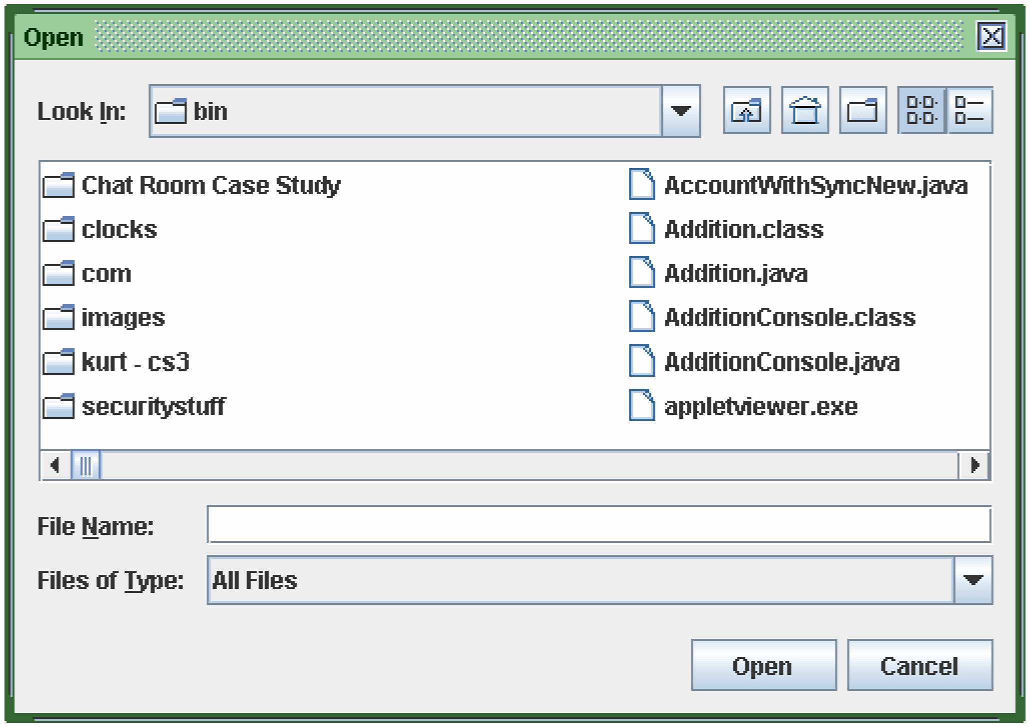
```
}
/** Returns an ImageIcon, or null if the path was invalid. */
protected static ImageIcon createImageIcon(String path) {
   java.net.URL imgURL = FileChooserDemo.class.getResource(path);
   if (imgURL != null) {
      return new ImageIcon(imgURL);
   } else {
      System.err.println("Couldn't find file: " + path);
      return null;
   }
}
// Create the GUI and show it.  For thread safety, this method should be invoked from the
// event-dispatching thread.
   private static void createAndShowGUI() {
   //Make sure we have nice window decorations.
   JFrame.setDefaultLookAndFeelDecorated(true);
   JDialog.setDefaultLookAndFeelDecorated(true);
   //Create and set up the window.
   JFrame frame = new JFrame("FileChooserDemo");
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   //Create and set up the content pane.
   JComponent newContentPane = new FileChooserDemo();
   newContentPane.setOpaque(true); //content panes must be opaque
   frame.setContentPane(newContentPane);
   //Display the window.
   frame.pack();
   frame.setVisible(true);
}
```

```
public static void main(String[] args) {
        //Schedule a job for the event-
dispatching thread:
        //creating and showing this
application's GUI.

javax.swing.SwingUtilities.invokeLate
r(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

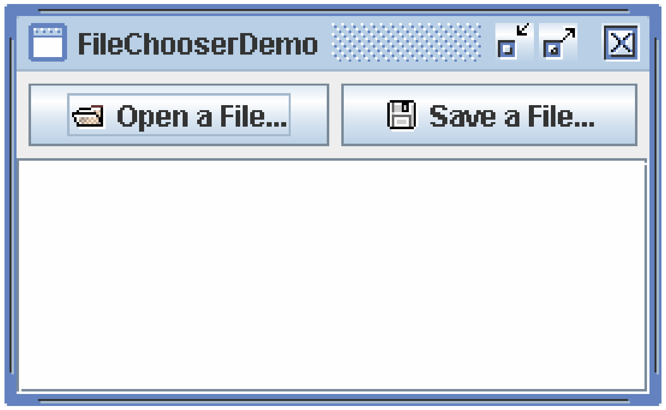


Initial FileChooserDemo dialog

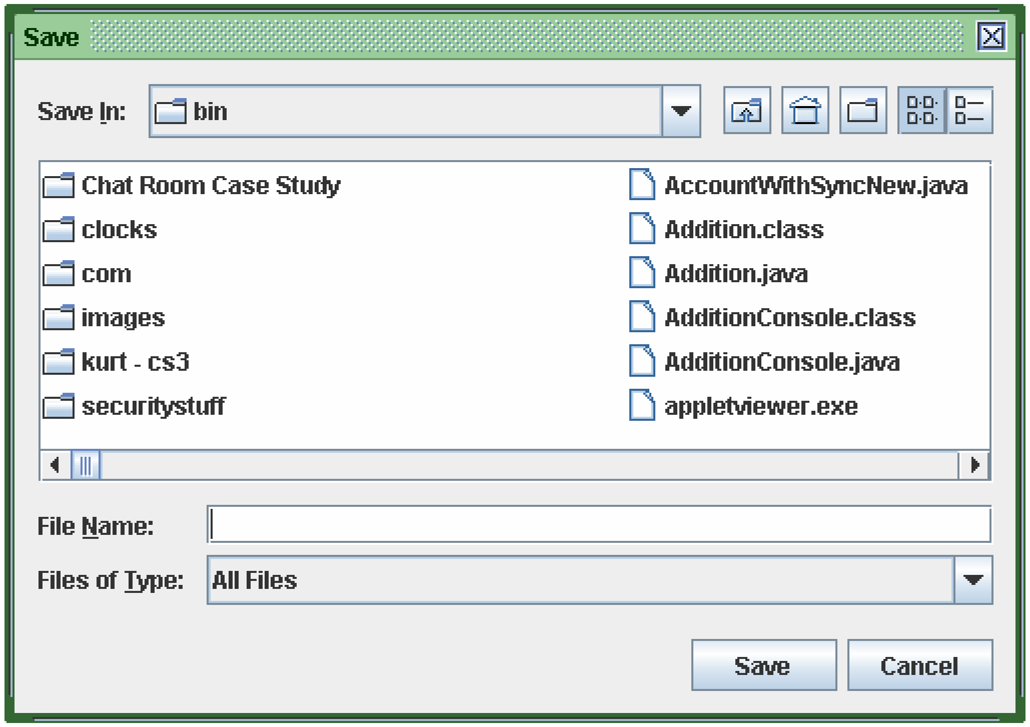
This window is used to return status messages from the FileChooser



Open dialog which appears after user clicks open button. Note that the argument to the `showOpenDialog` method specifies the parent component for the dialog. The parent component affects the position of the dialog and the frame that it depends on. The Java look and feel places the dialog directly over the parent component.



Initial FileChooserDemo dialog



Save dialog which appears after user clicks save button. Note that cursor is automatically placed in the file name box to await user entry of the name of the file to be saved.

Swing Text Components and HTML Rendering

- Some of our previous examples illustrated the two of the three basic text components used for presenting and editing text. We've already seen `JTextField` and `JTextArea`, and now we'll look at a more sophisticated text component called a `JEditorPane`.
- JEditorPane provides enhanced text-rendering capabilities. It supports styled documents that include formatting, font and color information.
- JEditorPane is capable of rendering HTML documents as well as Rich Text Format (RTF) documents.
- The following example utilizes the JEditorPane class to render HTML pages for a simple web browser application.

The Components of the Web Browser

- The next example consists of three basic components:

1. A WebBrowserPane which is an extension of class JEditorPane. WebBrowserPane creates a web browsing component that maintains a history of visited URLs.

2. WebToolBar is an extension of class JToolBar (JToolBar allows developers to add toolbars to GUIs to provide common functions such as cut, copy, paste, and navigation). This class provides commonly used navigation components for a WebBrowserPane. In this case a back button and forward button are provided.

3. Class WebBrowser uses a WebBrowserPane and a WebToolBar to create a simple web-browser application.

WebBrowserPane Class

```
// WebBrowserPane.java
// WebBrowserPane is a simple Web-browsing component that
// extends JEditorPane and maintains a history of visited URLs.
// Java core packages
import java.util.*;
import java.net.*;
import java.io.*;
// Java extension packages
import javax.swing.*;
import javax.swing.event.*;

public class WebBrowserPane extends JEditorPane {

   private List history = new ArrayList();
   private int historyIndex;

   // WebBrowserPane constructor
   public WebBrowserPane()
   {
      // disable editing to enable hyperlinks
      setEditable( false );
   }

   // display given URL and add it to history
   public void goToURL( URL url )
   {
      displayPage( url );
```

JEditorPane enables hyperlinks in HTML documents only if the JEditorPane is not editable.

```
      history.add( url );
      historyIndex = history.size() - 1;
   }
      // display next history URL in editorPane
   public URL forward()
   {
      historyIndex++;
      // do not go past end of history
      if ( historyIndex >= history.size() )
         historyIndex = history.size() - 1;

      URL url = ( URL ) history.get( historyIndex );
      displayPage( url );
     return url;
   }
   // display previous history URL in editorPane
   public URL back()
   {
      historyIndex--;
       // do not go past beginning of history
      if ( historyIndex < 0 )
         historyIndex = 0;
       // display previous URL
      URL url = ( URL ) history.get( historyIndex );
      displayPage( url );
      return url;
   }
```

```
// display given URL in JEditorPane
   private void displayPage( URL pageURL )
   {
      // display URL
      try {
         setPage( pageURL );
      }
      // handle exception reading from URL
      catch ( IOException ioException ) {
         ioException.printStackTrace();
      }
   }
}
```

WebToolBar Class

```
// WebToolBar.java
// WebToolBar is a JToolBar subclass that contains components
// for navigating a WebBrowserPane. WebToolBar includes back
// and forward buttons and a text field for entering URLs.
// Java core packages
import java.awt.*;
import java.awt.event.*;
import java.net.*;
// Java extension packages
import javax.swing.*;
import javax.swing.event.*;

public class WebToolBar extends JToolBar   implements HyperlinkListener {
   private WebBrowserPane webBrowserPane;
   private JButton backButton;
   private JButton forwardButton;
   private JTextField urlTextField;
   // WebToolBar constructor
   public WebToolBar( WebBrowserPane browser )
   {
      super( "Web Navigation" );
      // register for HyperlinkEvents
      webBrowserPane = browser;
      webBrowserPane.addHyperlinkListener( this );
      // create JTextField for entering URLs
      urlTextField = new JTextField( 25 );
```

```
urlTextField.addActionListener(
   new ActionListener() {
      // navigate webBrowser to user-entered URL
      public void actionPerformed( ActionEvent event )
      {
         // attempt to load URL in webBrowserPane
         try {
            URL url = new URL( urlTextField.getText() );
            webBrowserPane.goToURL( url );
         }
         // handle invalid URL
         catch ( MalformedURLException urlException ) { urlException.printStackTrace();
         }
      }
   }
);
      // create JButton for navigating to previous history URL
backButton = new JButton( new ImageIcon( getClass().getResource( "images/back.gif" ) ) );
backButton.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent event )
      {
         // navigate to previous URL
         URL url = webBrowserPane.back();
         // display URL in urlTextField
         urlTextField.setText( url.toString() );
      }
   }     );
```

```
      // create JButton for navigating to next history URL
      forwardButton = new JButton( new ImageIcon( getClass().getResource( "images/forward.gif" ) ) );
      forwardButton.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent event )
            {
               // navigate to next URL
               URL url = webBrowserPane.forward();
                // display new URL in urlTextField
               urlTextField.setText( url.toString() );
            }
         }
      );
      // add JButtons and JTextField to WebToolBar
      add( backButton );     add( forwardButton );     add( urlTextField );
     } // end WebToolBar constructor

   // listen for HyperlinkEvents in WebBrowserPane
   public void hyperlinkUpdate( HyperlinkEvent event )
   {
      // if hyperlink was activated, go to hyperlink's URL
      if ( event.getEventType() ==  HyperlinkEvent.EventType.ACTIVATED ) {
          // get URL from HyperlinkEvent
         URL url = event.getURL();
          // navigate to URL and display URL in urlTextField
         webBrowserPane.goToURL( url );
         urlTextField.setText( url.toString() );
      }  }  }
```

WebBrowser Class

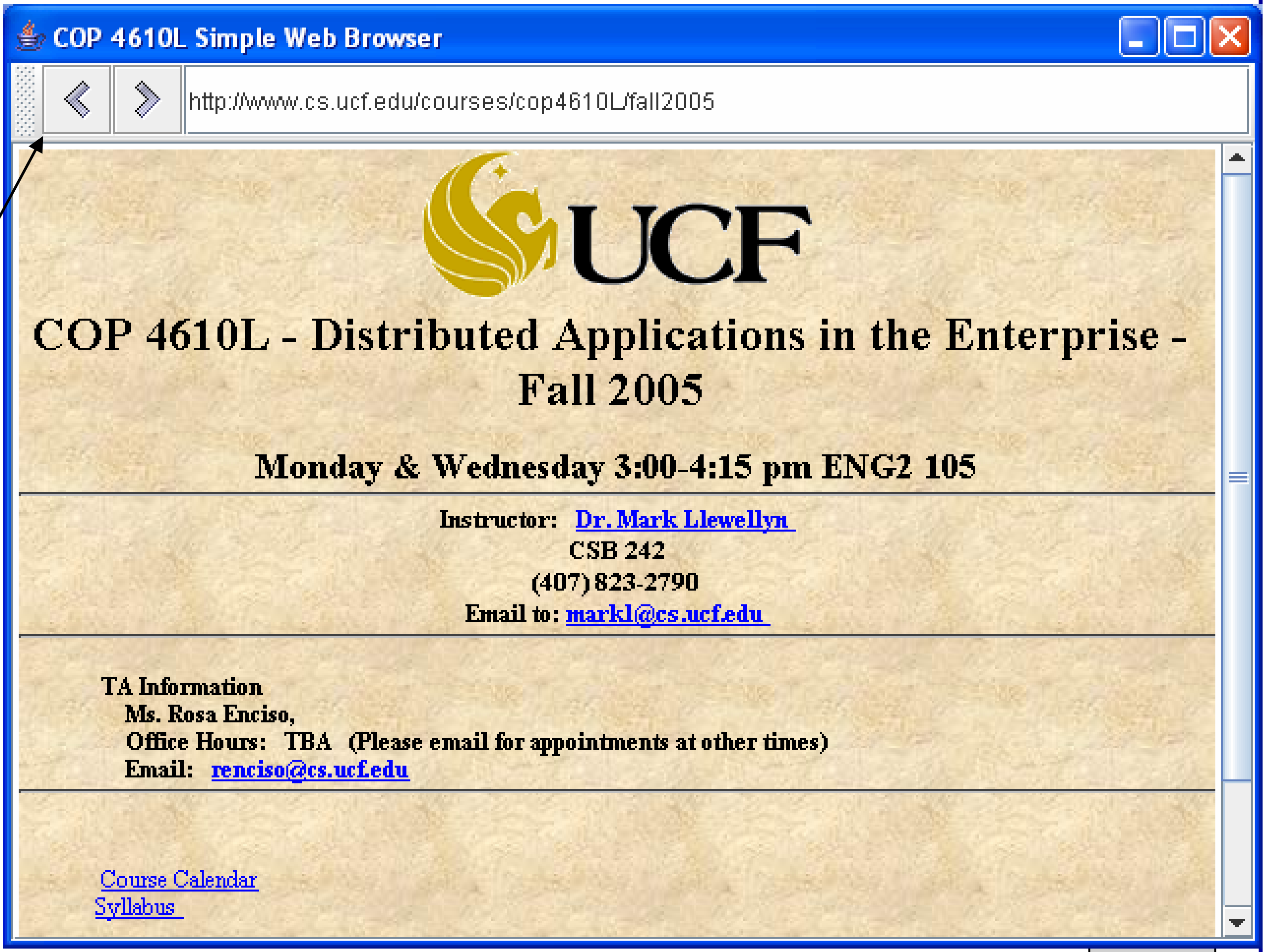
```
// WebBrowser.java
// WebBrowser is an application for browsing Web sites using
// a WebToolBar and WebBrowserPane.
// Java core packages
import java.awt.*;
import java.awt.event.*;
import java.net.*;
// Java extension packages
import javax.swing.*;
import javax.swing.event.*;

public class WebBrowser extends JFrame {
   private WebToolBar toolBar;
   private WebBrowserPane browserPane;
   // WebBrowser constructor
   public WebBrowser()
   {
      super( "COP 4610L Simple Web Browser" );
      // create WebBrowserPane and WebToolBar for navigation
      browserPane = new WebBrowserPane();
      toolBar = new WebToolBar( browserPane );
      // lay out WebBrowser components
      Container contentPane = getContentPane();
      contentPane.add( toolBar, BorderLayout.NORTH );
      contentPane.add( new JScrollPane( browserPane ), BorderLayout.CENTER );
   }
```

The previously defined classes

Recall that the default layout manager for JFrame is a BorderLayout.

```
// execute application
   public static void main( String args[] )
   {
      WebBrowser browser = new WebBrowser();
      browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
      browser.setSize( 640, 480 );
      browser.setVisible( true );
   }
}
```

WebToolBar instance


COP 4610L Simple Web Browser
http://www.cs.ucf.edu/courses/cop4610L/fall2005
UCF

COP 4610L - Distributed Applications in the Enterprise - Fall 2005

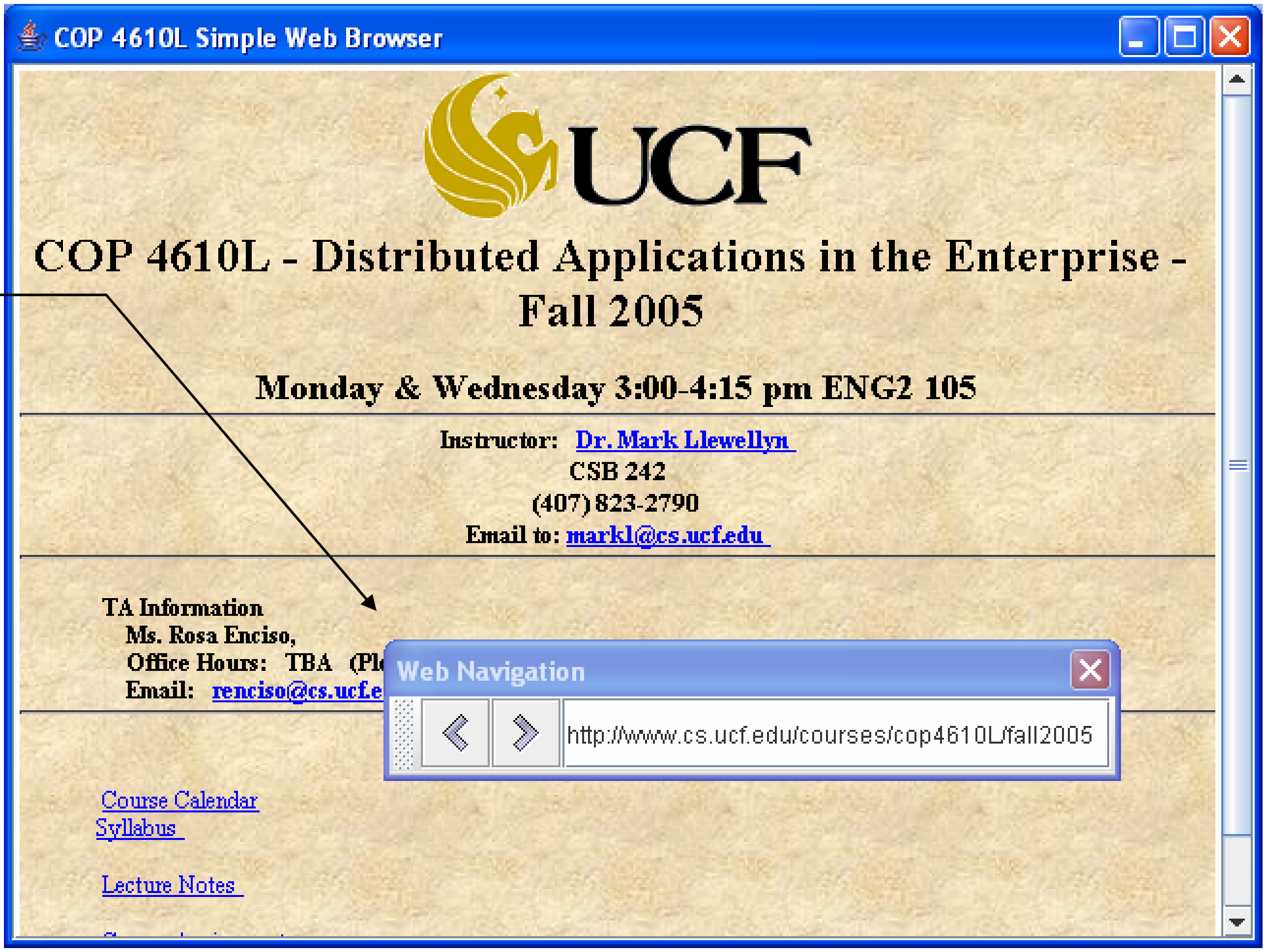
Monday & Wednesday 3:00-4:15 pm ENG2 105

Instructor: Dr. Mark Llewellyn
CSB 242
(407) 823-2790
Email to: markl@cs.ucf.edu

TA Information
Ms. Rosa Enciso,
Office Hours: TBA (Please email for appointments at other times)
Email: renciso@cs.ucf.edu

Course Calendar
Syllabus

WebToolBar instance has been moved by the user.



COP 4610L - Distributed Applications in the Enterprise - Fall 2005

Monday & Wednesday 3:00-4:15 pm ENG2 105

Instructor: Dr. Mark Llewellyn
CSB 242
(407) 823-2790
Email to: markl@cs.ucf.edu

TA Information
Ms. Rosa Enciso,
Office Hours: TBA (Pl
Email: renciso@cs.ucf.e

http://www.cs.ucf.edu/courses/cop4610L/fall2005

Course Calendar
Syllabus

Lecture Notes

JSplitPane and JTabbedPane Components

- JSplitPane and JTabbedPane are container components that enable application developers to present large amounts of information in a small screen area.
- JSplitPane handles this by dividing two components with a divider the user can reposition to expand and contract the visible area of the JSplitPane's child components.
  - A JSplitPane can contain only two child components, however, each child component may contain nested components.
- We'll look at an example using a JSplitPane component and then we'll examine the JTabbedPane component.

FavoritesWebBrowser Class

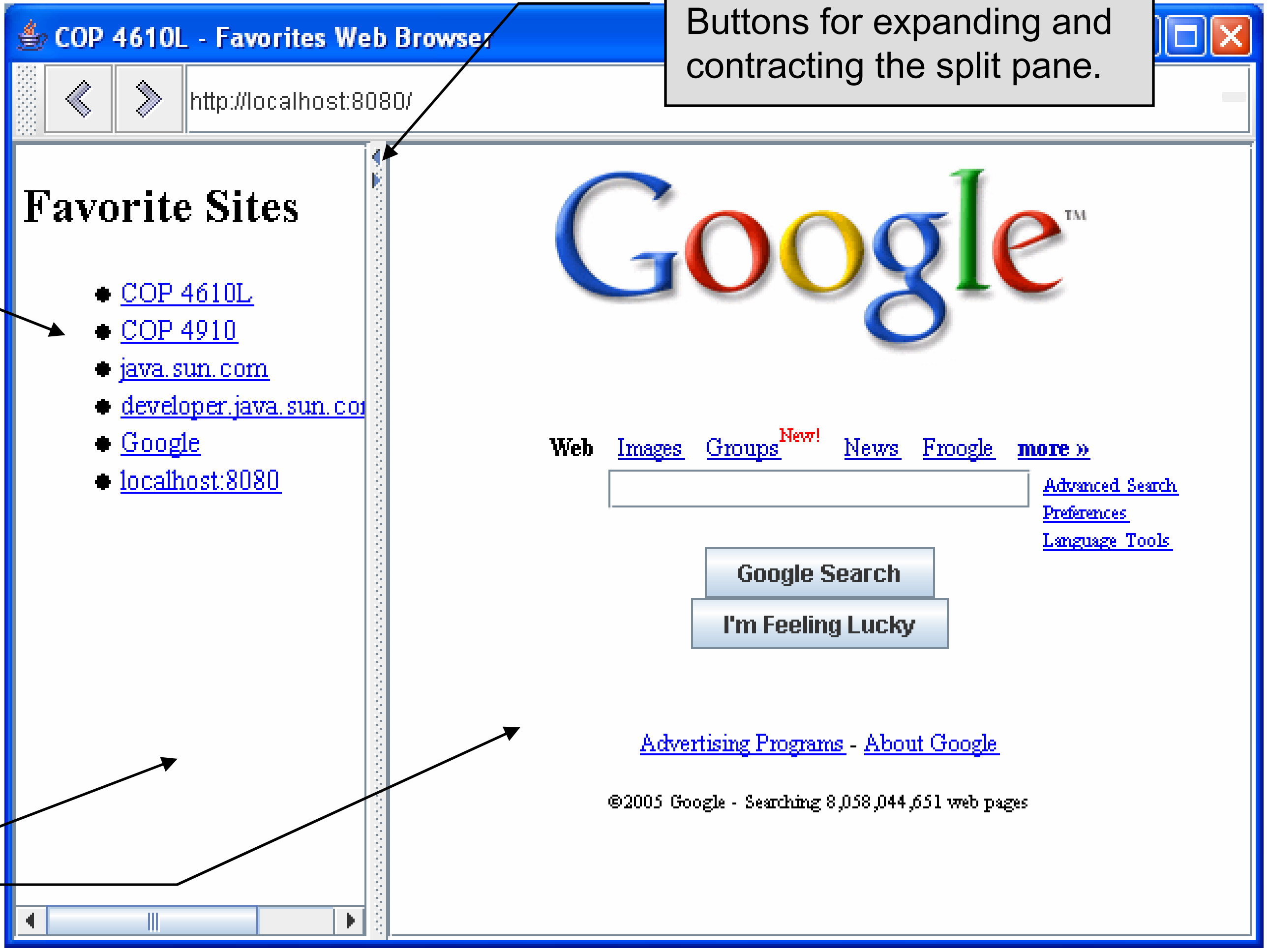
```
// FavoritesWebBrowser.java
// FavoritesWebBrowser is an application for browsing Web sites
// using a WebToolBar and WebBrowserPane and displaying an HTM
// page containing links to favorite Web sites.
// Java core packages
import java.awt.*;
import java.awt.event.*;
import java.net.*;
// Java extension packages
import javax.swing.*;
import javax.swing.event.*;

public class FavoritesWebBrowser extends JFrame {
  private WebToolBar toolBar;
  private WebBrowserPane browserPane;
  private WebBrowserPane favoritesBrowserPane;

  // WebBrowser constructor
  public FavoritesWebBrowser()
  {
    super( "COP 4610L - Favorites Web Browser" );
    // create WebBrowserPane and WebToolBar for navigation
    browserPane = new WebBrowserPane();
    toolBar = new WebToolBar( browserPane );

    // create WebBrowserPane for displaying favorite sites
    favoritesBrowserPane = new WebBrowserPane();
    // add WebToolBar as listener for HyperlinkEvents in favoritesBrowserPane
    favoritesBrowserPane.addHyperlinkListener( toolBar );
    // display favorites.html in favoritesBrowserPane
    favoritesBrowserPane.goToURL(
      getClass().getResource( "favorites.html" ) );
     // create JSplitPane with horizontal split (side-by-side)
    // and add WebBrowserPanes with JScrollPanes
    JSplitPane splitPane = new JSplitPane(
      JSplitPane.HORIZONTAL_SPLIT,
      new JScrollPane( favoritesBrowserPane ),
      new JScrollPane( browserPane ) );
     // position divider between WebBrowserPanes
    splitPane.setDividerLocation( 210 );
     // add buttons for expanding/contracting divider
    splitPane.setOneTouchExpandable( true );
     // lay out WebBrowser components
    Container contentPane = getContentPane();
    contentPane.add( toolBar, BorderLayout.NORTH );
    contentPane.add( splitPane, BorderLayout.CENTER );
  }
  // execute application
  public static void main( String args[] )  {
    FavoritesWebBrowser browser = new FavoritesWebBrowser();
    browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
    browser.setSize( 640, 480 );
    browser.setVisible( true );
  }  }
```

HTML file containing a list of favorite URLs.

The first argument indicates that the JSplitPane should display its child components side by side. A vertical-split would display the two components one on top of the other. The second two arguments are the components to be divided in the JSplitPane.

Sets the position of the divider between the two components.

Adds two buttons to the divider to allow the user to expand or contract the divider to one side or the other with a single click.



Buttons for expanding and contracting the split pane.

Favorites menu in JSplitPane

Two separate panes in the JSplitPane

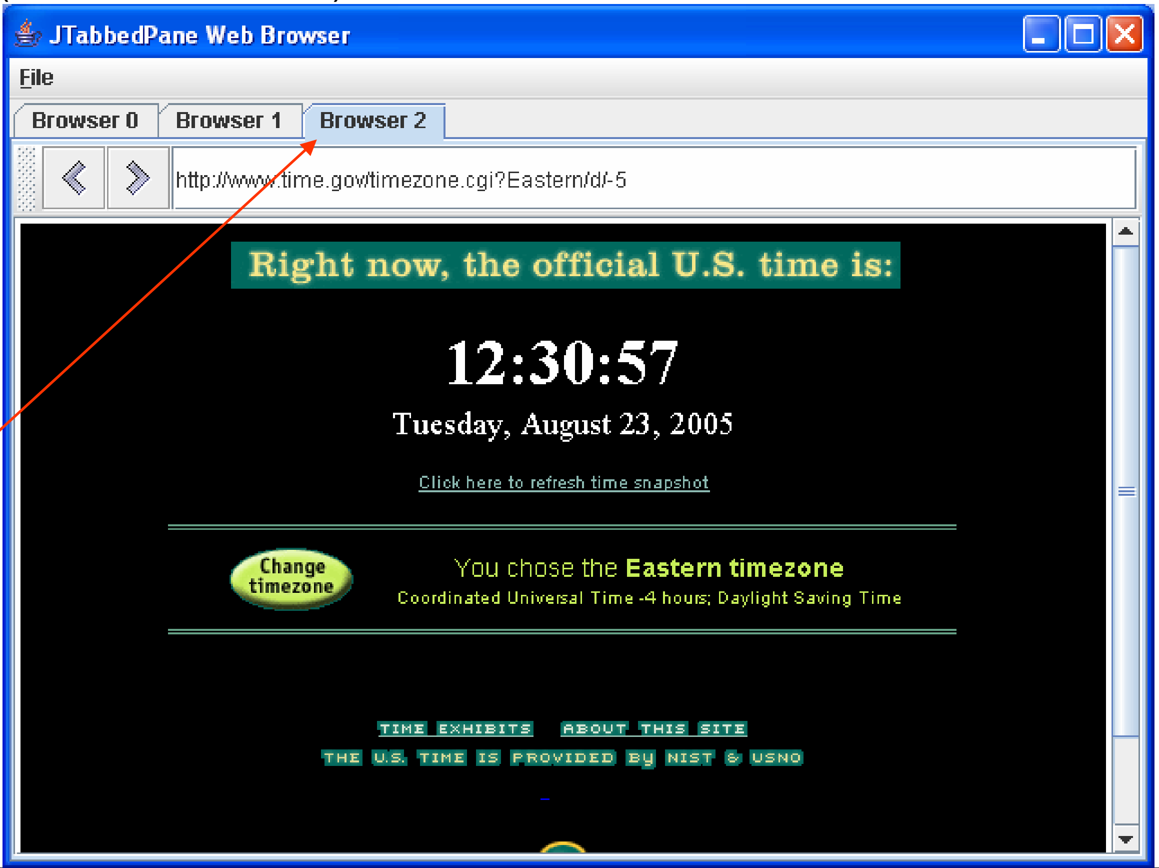
JTabbedPane Component

- `JTabbedPane` presents multiple components in separate tabs, which the user navigates between using a mouse or the keyboard.
- The example application `TabbedPaneWebBrowser` uses a `JTabbedPane` to enable a user to browse multiple webpages at one time within a single application window.
- The user invokes an `Action` to add a new `WebBrowserPane` to the `JTabbedPane`. Each time the user adds a new `WebBrowserPane`, the `JTabbedPane` creates a new tab and places the `WebBrowserPane` in this new tab.

TabbedPaneWebBrowser Class

```
// TabbedPaneWebBrowser.java
// TabbedPaneWebBrowser is an application that uses a
// JTabbedPane to display multiple Web browsers.
// Java core packages
import java.awt.*;
import java.awt.event.*;
// Java extension packages
import javax.swing.*;

public class TabbedPaneWebBrowser extends JFrame {

  // JTabbedPane for displaying multiple browser tabs
  private JTabbedPane tabbedPane = new JTabbedPane();

  // TabbedPaneWebBrowser constructor
  public TabbedPaneWebBrowser()
  {
    super( "JTabbedPane Web Browser" );

    // create first browser tab
    createNewTab();

    // add JTabbedPane to contentPane
    getContentPane().add( tabbedPane );
    // create File JMenu for creating new browser tabs and exiting application
    JMenu fileMenu = new JMenu( "File" );
    fileMenu.add( new NewTabAction() );
    fileMenu.addSeparator();
    fileMenu.add( new ExitAction() );
    fileMenu.setMnemonic( 'F' );

    JMenuBar menuBar = new JMenuBar();
    menuBar.add( fileMenu );
    setJMenuBar( menuBar );
  } // end TabbedPaneWebBrowser constructor

  // create new browser tab
  private void createNewTab()
  {
    // create JPanel to contain WebBrowserPane and WebToolBar
    JPanel panel = new JPanel( new BorderLayout() );

    // create WebBrowserPane and WebToolBar
    WebBrowserPane browserPane = new WebBrowserPane();
    WebToolBar toolBar = new WebToolBar( browserPane );

    // add WebBrowserPane and WebToolBar to JPanel
    panel.add( toolBar, BorderLayout.NORTH );
    panel.add( new JScrollPane( browserPane ),  BorderLayout.CENTER );
    // add JPanel to JTabbedPane
    tabbedPane.addTab( "Browser " + tabbedPane.getTabCount(), panel );
  }
  // Action for creating new browser tabs
  private class NewTabAction extends AbstractAction {
     // NewTabAction constructor
    public NewTabAction()
    {
      // set name, description and mnemonic key
      putValue( Action.NAME, "New Browser Tab" );
      putValue( Action.SHORT_DESCRIPTION, "Create New Web Browser Tab" );
      putValue( Action.MNEMONIC_KEY, new Integer( 'N' ) );
    }
    // when Action invoked, create new browser tab
    public void actionPerformed( ActionEvent event )
    {       createNewTab();     }
  }
   // Action for exiting application
  private class ExitAction extends AbstractAction {
    // ExitAction constructor
    public ExitAction()
    {
      // set name, description and mnemonic key
      putValue( Action.NAME, "Exit" );
      putValue( Action.SHORT_DESCRIPTION, "Exit Application" );
      putValue( Action.MNEMONIC_KEY, new Integer( 'x' ) );
    }
```

```
// when Action invoked, exit application
      public void actionPerformed( ActionEvent event )
      {        System.exit( 0 );      }
   }
   // execute application
   public static void main( String args[] )   {
      TabbedPaneWebBrowser browser = new TabbedPaneWebBrowser();
      browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
      browser.setSize( 640, 480 );
      browser.setVisible( true );
   }
}
```


JTabbedPane Web Browser
File
Browser 0
Browser 1
Browser 2
http://www.time.gov/timezone.cgi?Eastern/d/-5
Right now, the official U.S. time is:
12:30:57
Tuesday, August 23, 2005
Click here to refresh time snapshot
Change timezone
You chose the Eastern timezone
Coordinated Universal Time -4 hours; Daylight Saving Time
TIME EXHIBITS
ABOUT THIS SITE
THE U.S. TIME IS PROVIDED BY NIST & USNO

Example with three tabbed browsers active.

Drag and Drop

- Drag and drop is a common way to manipulate data in a GUI. Most GUIs emulate real-world desktops, with icons that represent the objects on a virtual desk.

- Drag and drop enables users to move items around the desktop and to move and copy data among applications using mouse gestures.

- A mouse gesture is a mouse movement that correpsonds to a drag and drop operation, such as dragging a file from one folder location and dropping the file into another folder.

- Two Java APIs enable drag and drop data transfer between applications.

The Data Transfer API and Drag and Drop API

- The data transfer API – package `java.awt.datatransfer` – enables copying and moving data within a single application or among multiple applications.
- The drag and drop API enables Java applications to recognize drag and drop gestures and to respond to drag and drop operations.
- A drag and drop operation uses the data transfer API to transfer the data from the drag source to the drop target. The application which is the drop target would use the drag and drop API to recognize that a drag and drop operation occurred and would use the data transfer API to retrieve the data transferred through the drag and drop operation.

A Drag and Drop Version of Our WebBrowser

- The last example in this section of notes presents a drag and drop version of the web browser that we have been developing.
- In this case the application DnDWebBrowser is an extension of our original web browser that also allows the user to drop a file onto the WebBrowserPane to view the file contents.
  - The user could drag and drop an HTML file from the host computer’s desktop (or other location) and drop the file on the WebBrowserPane to render the HTML.
  - The second method would be to open an HTML file containing URLs, then select a specific URL to drag and drop onto the web browser’s tool bar. Then from within the web browser, the user clicks the window and the web site contents are displayed.

A Drag and Drop WebBrowser Example

```
// DnDWebBrowser.java
// DnDWebBrowser is an application for viewing Web pages using dra
// Java core packages
import java.awt.*;
import java.awt.dnd.*;
import java.awt.datatransfer.*;
import java.util.*;
import java.io.*;
import java.net.*;
// Java extension packages
import javax.swing.*;
import javax.swing.event.*;

public class DnDWebBrowser extends JFrame {
   private WebToolBar toolBar;
   private WebBrowserPane browserPane;
   // DnDWebBrowser constructor
   public DnDWebBrowser()
   {
      super( "Drag-and-Drop Web Browser" );
      // create WebBrowserPane and WebToolBar for navigation
      browserPane = new WebBrowserPane();
      toolBar = new WebToolBar( browserPane );
       // enable WebBrowserPane to accept drop operations, using
      // DropTargetHandler as the DropTargetListener
      browserPane.setDropTarget( new DropTarget( browserPane,
         DnDConstants.ACTION_COPY, new DropTargetHandler() ) );
```

Create a WebBrowserPane and a WebToolBar

Create a drop target within the browserPane object. The first argument is the GUI component onto which the user can drop objects. The second argument is the type of dnd operations supported. Third argument is the listener to be notified of dnd operation events.

```
      // lay out WebBrowser components
      Container contentPane = getContentPane();
      contentPane.add( toolBar, BorderLayout.NORTH );
      contentPane.add( new JScrollPane( browserPane ),
         BorderLayout.CENTER );
   }
   // inner class to handle DropTargetEvents
   private class DropTargetHandler implements DropTargetListener {
      // handle drop operation
      public void drop( DropTargetDropEvent event )
      {
         // get dropped Transferable object
         Transferable transferable = event.getTransferable();
         // if Transferable is a List of Files, accept drop
         if ( transferable.isDataFlavorSupported( DataFlavor.javaFileListFlavor ) ) {
            // accept the drop operation to copy the object
            event.acceptDrop( DnDConstants.ACTION_COPY );
            // process list of files and display each in browser
            try {
               // get List of Files
               java.util.List fileList =
                  ( java.util.List ) transferable.getTransferData( DataFlavor.javaFileListFlavor );
               Iterator iterator = fileList.iterator();
```

DropTargetHandler implements interface DropTargetListener to listen for dnd operation events related to a DropTarget.

Handle drop event

Get the transferable object the user dropped.

Represents a list of files

```
         while ( iterator.hasNext() ) {
            File file = ( File ) iterator.next();
            // display File in browser and complete drop
            browserPane.goToURL( file.toURL() );
         }
         // indicate successful drop
         event.dropComplete( true );
      }
      // handle exception if DataFlavor not supported
      catch ( UnsupportedFlavorException flavorException ) {
         flavorException.printStackTrace();
         event.dropComplete( false );
      }
      // handle exception reading Transferable data
      catch ( IOException ioException ) {
         ioException.printStackTrace();
         event.dropComplete( false );
      }
   }
   // if dropped object is not file list, reject drop
   else
      event.rejectDrop();
}
```

Drag and drop operation was successful so close with argument *true*.

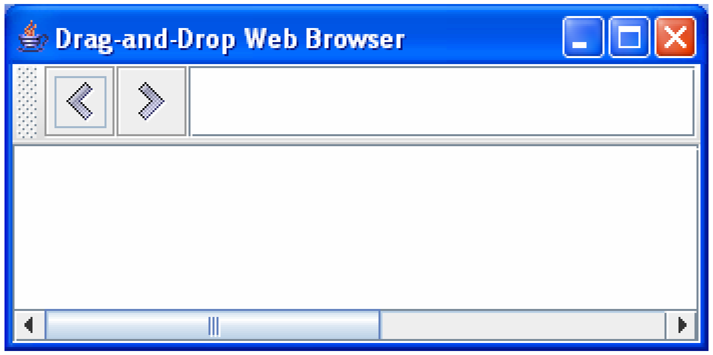
Dropped object was not a list of files so do not accept the dropped object.

```
// handle drag operation entering DropTarget
  public void dragEnter( DropTargetDragEvent event )
  {
    // if data is javaFileListFlavor, accept drag for copy
    if ( event.isDataFlavorSupported(
      DataFlavor.javaFileListFlavor ) )
      event.acceptDrag( DnDConstants.ACTION_COPY );
    // reject all other DataFlavors
    else
      event.rejectDrag();
  }
  // invoked when drag operation exits DropTarget
  public void dragExit( DropTargetEvent event ) { }

  // invoked when drag operation occurs over DropTarget
  public void dragOver( DropTargetDragEvent event ) { }

  // invoked if dropAction changes (e.g., from COPY to LINK)
  public void dropActionChanged( DropTargetDragEvent event )  { }
 } // end class DropTargetHandler
  // execute application
 public static void main( String args[] )   {
  DnDWebBrowser browser = new DnDWebBrowser();
  browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
  browser.setSize( 640, 480 );
  browser.setVisible( true );        }   }
```

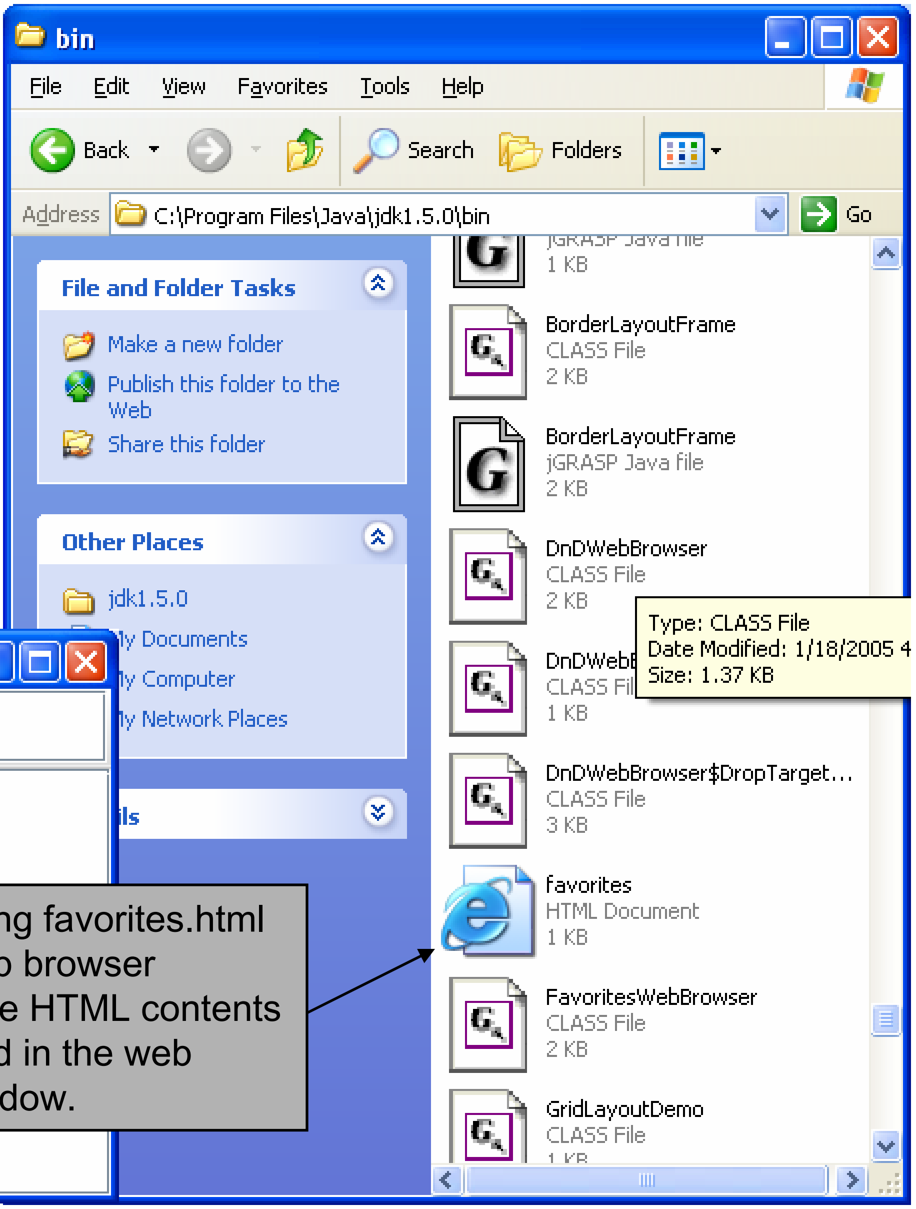
Invoked when the dnd operation enters a DropTarget

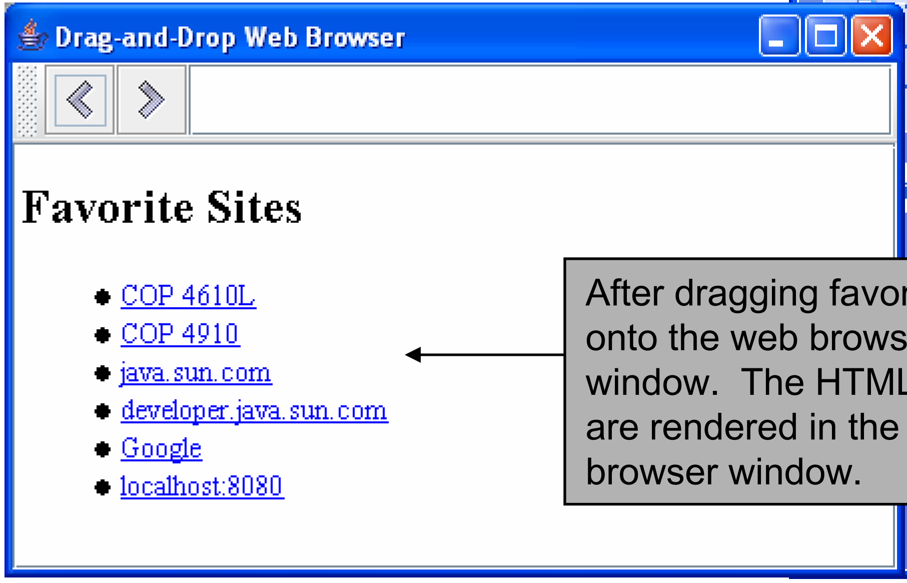
DnDConstants include ACTION_COPY for copying a dragged object.. ACTION_MOVE for moving a dropped object. ACTION_LINK for creating a link to an object.

Interface methods that do nothing in this case.



Initial Browser Window

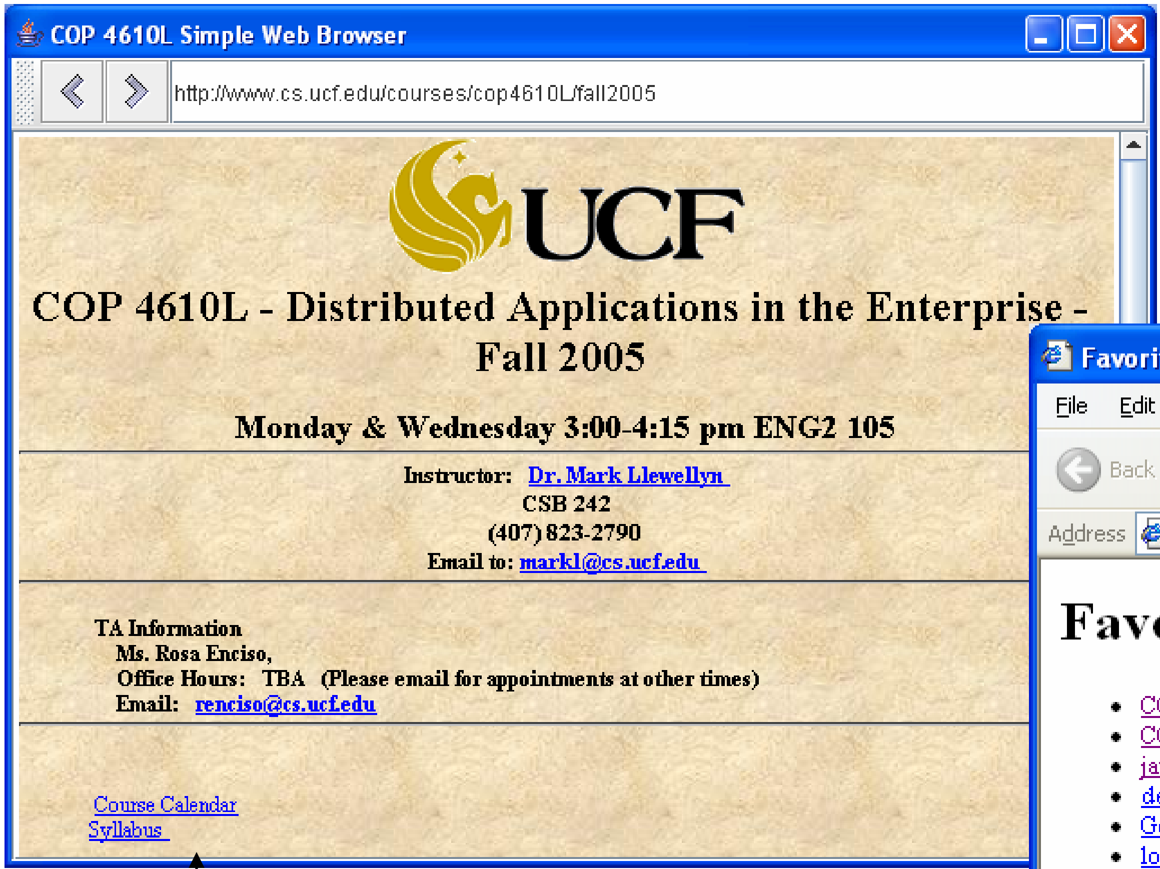

bin
File Edit View Favorites Tools Help
Back Search Folders
Address C:\Program Files\Java\jdk1.5.0\bin Go
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
Other Places
jdk1.5.0
My Documents
My Computer
My Network Places
BorderLayoutFrame CLASS File 2 KB
BorderLayoutFrame jGRASP Java file 2 KB
DnDWebBrowser CLASS File 2 KB
Type: CLASS File
Date Modified: 1/18/2005 4
Size: 1.37 KB
DnDWebBrowser$DropTarget... CLASS File 3 KB
favorites HTML Document 1 KB
FavoritesWebBrowser CLASS File 2 KB
GridLayoutDemo CLASS File



Favorite Sites

- COP 4610L
- COP 4910
- java.sun.com
- developer.java.sun.com
- Google
- localhost:8080

After dragging favorites.html onto the web browser window. The HTML contents are rendered in the web browser window.



COP 4610L Simple Web Browser

http://www.cs.ucf.edu/courses/cop4610L/fall2005

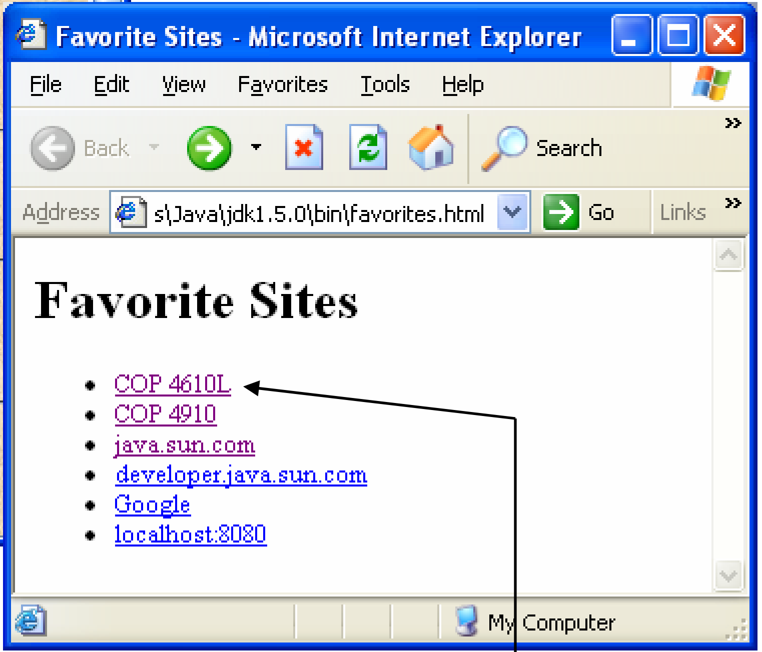
COP 4610L - Distributed Applications in the Enterprise - Fall 2005

Monday & Wednesday 3:00-4:15 pm ENG2 105

Instructor: Dr. Mark Llewellyn
CSB 242
(407) 823-2790
Email to: markl@cs.ucf.edu

TA Information
Ms. Rosa Enciso,
Office Hours: TBA (Please email for appointments at other times)
Email: renciso@cs.ucf.edu

Course Calendar
Syllabus



Favorite Sites - Microsoft Internet Explorer

Address s\Java\jdk1.5.0\bin\favorites.html

Favorite Sites

- COP 4610L
- COP 4910
- java.sun.com
- developer.java.sun.com
- Google
- localhost:8080

User opens favorites.html in another window to display the URLs contained in the file. Then a specific URL is selected and dragged onto the web browser tool bar and dropped. Once the URL is located in the window, the user clicks on this link and the web content is displayed in the browser window.