Chapter 2 - Advanced Swing Graphical User Interface Components

Outline

2.1 Introduction

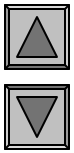
- Graphical user interface components
  - Swing components
  - Abstract Windowing Toolkit (AWT)
- Sample swing components
  - **`JEditorPane`**
  - **`JSplitPane`**
  - **`JTabbedPane`**
  - Swing **`Action`**s
- Using swing components to build applications for users with disabilities

2.2 WebBrowser Using `JEditorPane` and `JToolBar`

- Web browser application
  - Swing text components
  - Swing container components

2.2.1 Swing Text Components and HTML Rendering

- Swing Text Components
  - Base class **`JTextComponent`**
  - **`JTextField`** – single-line text component
  - **`JTextArea`** – multiple lines text component
  - **`JEditorPane`** – rendering HTML documents and Rich Text Format documents



Outline

```
1    // WebBrowserPane.java
2    // WebBrowserPane is a simple Web-browsing component that
3    // extends JEditorPane and maintains a history of visited URLs.
4    package com.deitel.advjhtp1.gui.webbrowser;
5
6    // Java core packages
7    import java.util.*;
8    import java.net.*;
9    import java.io.*;
10
11   // Java extension packages
12   import javax.swing.*;
13
14   public class WebBrowserPane extends JEditorPane {
15
16      private List history = new ArrayList();
17      private int historyIndex;
18
19      // WebBrowserPane constructor
20      public WebBrowserPane()
21      {
22         // disable editing to enable hyperlinks
23         setEditable( false );
24      }
25
26      // display given URL and add it to history
27      public void goToURL( URL url )
28      {
29         displayPage( url );
30         history.add( url );
31         historyIndex = history.size() - 1;
32      }
33
```

JEditorPane used to render HTML pages

Disable text editing

Fig. 2.1 WebBrowserPane subclass of JEditorPane for viewing Web and maining URL history.

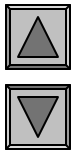
Line 14

Line 23

```
34      // display next history URL in editorPane
35      public URL forward()
36      {
37         historyIndex++;
38
39         // do not go past end of history
40         if ( historyIndex >= history.size() )
41            historyIndex = history.size() - 1;
42
43         URL url = ( URL ) history.get( historyIndex );
44         displayPage( url );
45
46         return url;
47      }
48
49      // display previous history URL in editorPane
50      public URL back()
51      {
52         historyIndex--;
53
54         // do not go past beginning of history
55         if ( historyIndex < 0 )
56            historyIndex = 0;
57
58         // display previous URL
59         URL url = ( URL ) history.get( historyIndex );
60         displayPage( url );
61
62         return url;
63      }
64
65      // display given URL in JEditorPane
66      private void displayPage( URL pageURL )
67      {
```

Retrieve the URL from the history **`List`** and display the URL in **`WebBrowserPane`**

Ensure **`historyIndex`** does not fall below 0



Outline

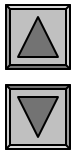
Fig. 2.1 WebBrowserPane subclass of JEditorPane for viewing Web nd ning URL .

Line 43-44 and 59-60

e 56

```
68          // display URL
69          try {
70             setPage( pageURL );
71          }
72
73          // handle exception reading from URL
74          catch ( IOException ioException ) {
75             ioException.printStackTrace();
76          }
77       }
78    }
```

Method **setPage** of class **JEditorPane** displays the page referenced by **pageURL**



Outline

Fig. 2.1 WebBrowserPane subclass of JEditorPane for viewing Web sites and maintaining URL history.

Line 70

2.2.2 Swing Toolbars

- Toolbar – GUI container
  - Buttons
  - Other GUI components
- Adding toolbars with class **`JToolBar`**

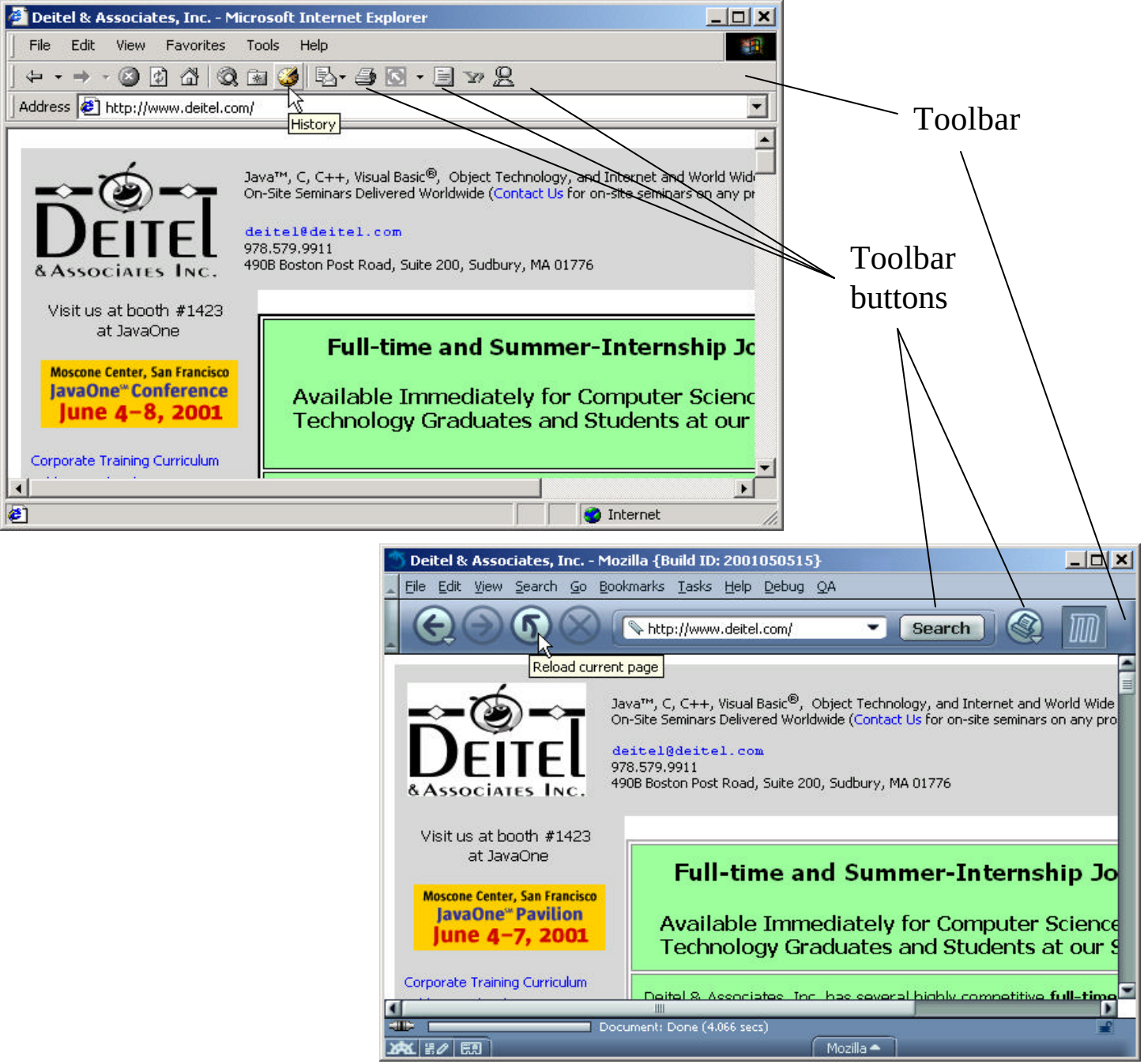
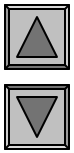


Fig. 2.2 Toolbars for navigating the Web in Internet Explorer and Mozilla.

Fig. 2.2 Toolbars for navigating the Web in Internet Explorer and Mozilla.



Outline

Fig. 2.3 WebToolBar JToolBar subclass for navigating URLs

wserPane.

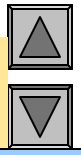
Line 16

Line 25-113

```
1    // WebToolBar.java
2    // WebToolBar is a JToolBar subclass that contains components
3    // for navigating a WebBrowserPane. WebToolBar includes back
4    // and forward buttons and a text field for entering URLs.
5    package com.deitel.advjhtp1.gui.webbrowser;
6
7    // Java core packages
8    import java.awt.*;
9    import java.awt.event.*;
10   import java.net.*;
11
12   // Java extension packages
13   import javax.swing.*;
14   import javax.swing.event.*;
15
16   public class WebToolBar extends JToolBar
17      implements HyperlinkListener {
18
19      private WebBrowserPane webBrowserPane;
20      private JButton backButton;
21      private JButton forwardButton;
22      private JTextField urlTextField;
23
24      // WebToolBar constructor
25      public WebToolBar( WebBrowserPane browser )
26      {
27         super( "Web Navigation" );
28
29         // register for HyperlinkEvents
30         webBrowserPane = browser;
31         webBrowserPane.addHyperlinkListener( this );
32
```

WebToolBar extends class **JToolBar** to provide commonly used navigation components for a **WebBrowserPane**.

WebToolBar constructor takes as an argument a **WebBrowserPane** for displaying Web pages.



Outline

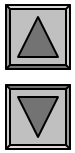
```
33          // create JTextField for entering URLs
34          urlTextField = new JTextField( 25 );
35          urlTextField.addActionListener(
36             new ActionListener() {
37
38                // navigate webBrowser to user-entered U
39                public void actionPerformed( ActionEvent
40                {
41                   // attempt to load URL in webBrowserPane
42                   try {
43                      URL url = new URL( urlTextField.getText() );
44                      webBrowserPane.goToURL( url );
45                   }
46
47                   // handle invalid URL
48                   catch ( MalformedURLException urlException ) {
49                      urlException.printStackTrace();
50                   }
51                }
52             }
53          );
54
55          // create JButton for navigating to previous history URL
56          backButton = new JButton( new ImageIcon(
57             getClass().getResource( "images/back.gif" ) ) );
58
59          backButton.addActionListener(
60             new ActionListener() {
61
```

Create **urlTextField** and its associated **ActionListener**.

2.3 olBar Bar subclass for navigating URLs in a WebBrowserPane.

Lines 34-53

Create **backButton** and its associated **ActionListener**.



Outline

Fig. 2.3 WebToolBar JToolBar subclass for navigating URLs in a WebBrowserPane.

```
62             public void actionPerformed( ActionEvent event )
63             {
64                // navigate to previous URL
65                URL url = webBrowserPane.back();
66
67                // display URL in urlTextField
68                urlTextField.setText( url.toString() );
69             }
70          }
71       );
72
73       // create JButton for navigating to next history URL
74       forwardButton = new JButton( new ImageIcon(
75          getClass().getResource( "images/forward.gif" ) ) );
76
77       forwardButton.addActionListener(
78          new ActionListener() {
79
80             public void actionPerformed( ActionEvent event )
81             {
82                // navigate to next URL
83                URL url = webBrowserPane.forward();
84
85                // display new URL in urlTextField
86                urlTextField.setText( url.toString() );
87             }
88          }
89       );
90
91       // add JButtons and JTextField to WebT
92       add( backButton );
93       add( forwardButton );
94       add( urlTextField );
95
96    } // end WebToolBar constructor
```

Create **forwardButton** and its associated **ActionListener**.

Lines 92-94

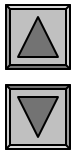
Add **backButton**, **forwardButton** and **urlTextField** to the **WebToolBar** by invoking method **add** of class **JToolBar**

```
97
98       // listen for HyperlinkEvents in WebBrowserPane
99       public void hyperlinkUpdate( HyperlinkEvent event )
100      {
101         // if hyperlink was activated, go to hyperlinked URL
102         if ( event.getEventType() ==
103            HyperlinkEvent.EventType.ACTIVATED ) {
104
105            // get URL from HyperlinkEvent
106            URL url = event.getURL();
107
108            // navigate to URL and display URL in urlTextField
109            webBrowserPane.goToURL( url );
110            urlTextField.setText( url.toString() );
111         }
112      }
113   }
```

Method **hyperlinkUpdate** invokes method **getEventType** of class **HyperlinkEvent** to check the event type and retrieves the **HyperlinkEvent's** URL.

Fig. 2.3
r
URLs
WebBrowserPane.

Lines 99-112



Outline

Fig. 2.4 WebBrowser application for browsing Web sites using WebBrowserPane and WebToolBar.

Lines 26-27

Lines 30-33

```
1  // WebBrowser.java
2  // WebBrowser is an application for browsing Web sites using
3  // a WebToolBar and WebBrowserPane.
4  package com.deitel.advjhtp1.gui.webbrowser;
5
6  // Java core packages
7  import java.awt.*;
8  import java.awt.event.*;
9  import java.net.*;
10
11 // Java extension packages
12 import javax.swing.*;
13 import javax.swing.event.*;
14
15 public class WebBrowser extends JFrame {
16
17    private WebToolBar toolBar;
18    private WebBrowserPane browserPane;
19
20    // WebBrowser constructor
21    public WebBrowser()
22    {
23       super( "Deitel Web Browser" );
24
25       // create WebBrowserPane and WebToolBar for navigat
26       browserPane = new WebBrowserPane();
27       toolBar = new WebToolBar( browserPane );
28
29       // lay out WebBrowser components
30       Container contentPane = getContentPane();
31       contentPane.add( toolBar, BorderLayout.NORTH );
32       contentPane.add( new JScrollPane( browserPane ),
33          BorderLayout.CENTER );
34    }
35
```

Create a **WebBrowserPane** and a **WebToolBar**.

Add the **WebBrowserPane** and **WebToolBar** to the **WebBrowser's** content pane.

```
36       // execute application
37       public static void main( String args[] )
38       {
39          WebBrowser browser = new WebBrowser();
40          browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
41          browser.setSize( 640, 480 );
42          browser.setVisible( true );
43       }
44    }
```

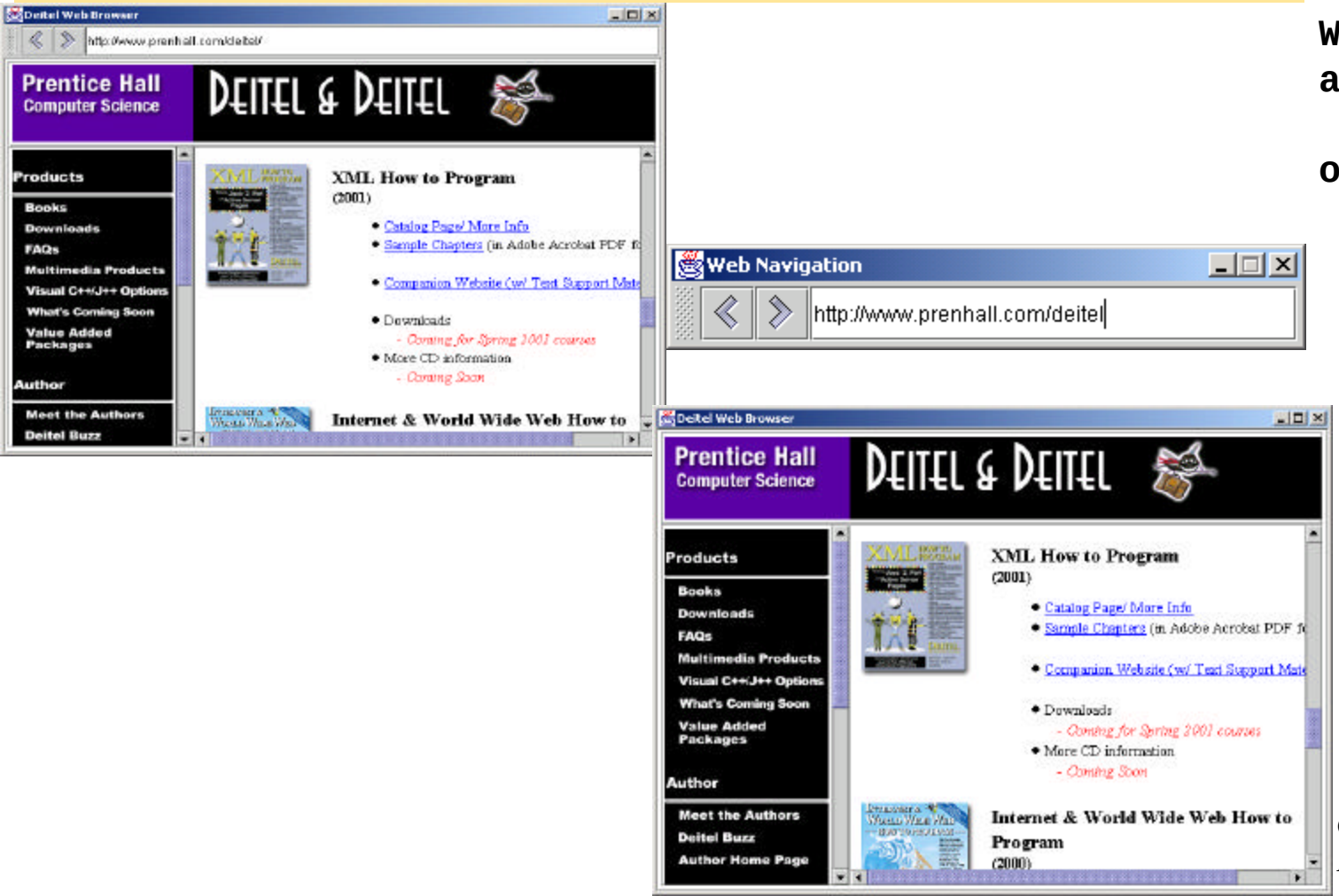
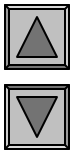


Fig. 2.4 WebBrowser application for browsing Web sites using WebBrowserPane and WebToolBar.

output

2.3 Swing Actions

- Command design pattern
  - Define functionality once in a reusable object
- **`Action`** interface
  - Required method of the Command design pattern
  - Process **`ActionEvent`**s generated by GUI components
- Easy to enable or disable actions



Outline

Fig. 2.5 ActionSample application demonstrating the Command design pattern with Swing Actions.

Lines 15-16

```
1  // ActionSample.java
2  // Demonstrating the Command design pattern with Swing Actions.
3  package com.deitel.advjhtp1.gui.actions;
4
5  // Java core packages
6  import java.awt.*;
7  import java.awt.event.*;
8
9  // Java extension packages
10 import javax.swing.*;
11
12 public class ActionSample extends JFrame {
13
14    // Swing Actions
15    private Action sampleAction;
16    private Action exitAction;
17
18    // ActionSample constructor
19    public ActionSample()
20    {
21       super( "Using Actions" );
22
23       // create AbstractAction subclass for sampleAction
24       sampleAction = new AbstractAction() {
25
26          public void actionPerformed( ActionEvent event )
27          {
28             // display message indicating sampleAction invok
29             JOptionPane.showMessageDialog( ActionSample.this
30                "The sampleAction was invoked" );
31
32             // enable exitAction and associated GUI
33             exitAction.setEnabled( true );
34          }
35       };
```

Declare **Action** references **sampleAction** and **exitAction**.

Create an anonymous inner class **Abs** assig refer

Swing event mechanism invokes method **actionPerformed** when the user activates a GUI component associated with

Display a **JOptionPane** message dialog to inform the user that **sampleAction**

Enables the **exitAction** and its associated GUI components.

Outline

```
36
37          // set Action name
38          sampleAction.putValue( Action.NAME, "Sample Action" );
39
40          // set Action Icon
41          sampleAction.putValue( Action.SMALL_ICON, new ImageIcon(
42             getClass().getResource( "images/Help24.gif" ) ) );
43
44          // set Action short description (tooltip text)
45          sampleAction.putValue( Action.SHORT_DESCRIPTION,
46             "A Sample Action" );
47
48          // set Action mnemonic key
49          sampleAction.putValue( Action.MNEMONIC_KEY,
50             new Integer( 'S' ) );
51
52          // create AbstractAction subclass for exitAction
53          exitAction = new AbstractAction() {
54
55             public void actionPerformed( ActionEvent event )
56             {
57                // display message indicating exitAction invoked
58                JOptionPane.showMessageDialog( ActionSample.this,
59                   "The exitAction was invoked" );
60                System.exit( 0 );
61             }
62          };
63
64          // set Action name
65          exitAction.putValue( Action.NAME, "Exit" );
66
67          // set Action icon
68          exitAction.putValue( Action.SMALL_ICON, new ImageIcon(
69             getClass().getResource( "images/EXIT.gif" ) ) );
70
```

Repeatedly invoke method **putValue** of interface **Action** to configure **sampleAction** properties.

design pattern with Swing

Create an anonymous inner cl **A** as re

Swing event mechanism invokes method **actionPerformed** when the user activates a GUI component associated with

Display a **JOptionPane** message dialog to inform the user that **exitAction** was invoked.

Repeatedly invoke method **putValue** of interface **Action** to configure **exitAction** properties.

```
71          // set Action short description (tooltip text)
72          exitAction.putValue( Action.SHORT_DESCRIPTION,
73             "Exit Application" );
74
75          // set Action mnemonic key
76          exitAction.putValue( Action.MNEMONIC_KEY,
77             new Integer( 'x' ) );
78
79          // disable exitAction and associated GUI
80          exitAction.setEnabled( false );
81
82          // create File menu
83          JMenu fileMenu = new JMenu( "File" );
84
85          // add sampleAction and exitAction to File
86          // create a JMenuItem for each Action
87          fileMenu.add( sampleAction );
88          fileMenu.add( exitAction );
89
90          fileMenu.setMnemonic( 'F' );
91
92          // create JMenuBar and add File menu
93          JMenuBar menuBar = new JMenuBar();
94          menuBar.add( fileMenu );
95          setJMenuBar( menuBar );
96
97          // create JToolBar
98          JToolBar toolBar = new JToolBar();
99
100         // add sampleAction and exitAction to JToolB
101         // JButtons for each Action
102         toolBar.add( sampleAction );
103         toolBar.add( exitAction );
104
```

Repeatedly invoke method **putValue** of interface **Action** to configure **exitAction** properties.

Disables the **exitAction** and its associated GUI components.

Create **fileMenu** **Jmenu**, add **sampleAction** and **exitAction** to the **fileMenu**, and set menu mnemonic key.

Add the **fileMenu** to a **JMenuBar** and invoke method **setJMenuBar** of class **JFrame** to add the **JMenuBar** to the application.

Creates **toolBar** **JToolBar** and add **sampleAction** and **exitAction** to the **toolBar**.

```
105         // create JButton and set its Action to sampleActio
106         JButton sampleButton = new JButton();
107         sampleButton.setAction( sampleAction );
108
109         // create JButton and set its Action to exitAction
110         JButton exitButton = new JButton( exitAction );
111
112         // lay out JButtons in JPanel
113         JPanel buttonPanel = new JPanel();
114         buttonPanel.add( sampleButton );
115         buttonPanel.add( exitButton );
116
117         // add toolBar and buttonPanel to JFrame's content pa
118         Container container = getContentPane();
119         container.add( toolBar, BorderLayout.NORTH );
120         container.add( buttonPanel, BorderLayout.CENTER );
121      }
122
123      // execute application
124      public static void main( String args[] )
125      {
126         ActionSample sample = new ActionSample();
127         sample.setDefaultCloseOperation( EXIT_ON_CLOSE );
128         sample.pack();
129         sample.setVisible( true );
130      }
131   }
```

Create **JButton** **sampleButton** and **exitButton**, and associate **sampleAction** and **exitAction** to them.

Add **JButtons** to a **JPanel**.

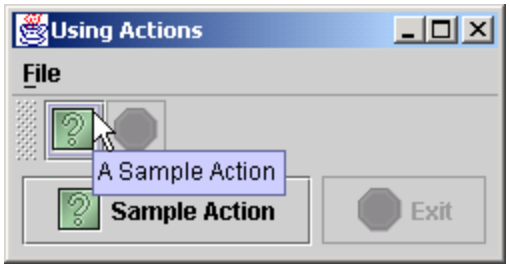
Lay out the **JToolBar** and **JPanel** in the **JFrame**'s content pane.

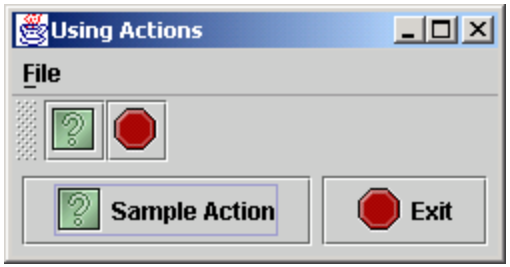
application demonstrating ommand design pattern

110

Lines 113-115

Lines 118-120





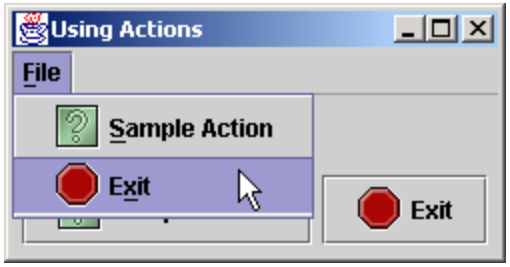


Fig. 2.5 ActionSample application demonstrating the Command design pattern with Swing Actions.

program output

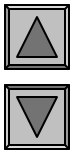
2.3 Swing Actions (Cont.)

Name	Description
NAME	Name to be used for GUI-component labels.
SHORT_DESCRIPTION	Descriptive text for use in tooltips.
SMALL_ICON	**`Icon`** for displaying in GUI-component labels.
MNEMONIC_KEY	Mnemonic key for keyboard access (e.g., for accessing menus and menu items using the keyboard).
ACCELERATOR_KEY	Accelerator key for keyboard access (e.g., using the *Ctrl* key).
ACTION_COMMAND_KEY	Key for retrieving command string to be used in **`ActionEvent`**s.
LONG_DESCRIPTION	Descriptive text, e.g., for application help.

Fig. 2.6 **`Action`** class **`static`** keys for **`Action`** properties.

2.4 JSplitPane and JTabbedPane

- Container components
  - Display information in a small area
- JSplitPane
  - Divide two components with a divider
- JTabbedPane
  - Separate components with file-folder-style tabs



Outline

Fig. 2.7 FavoritesWebBrowser application for displaying two Web pages side-by-side using JSplitPane.

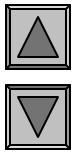
Lines 31-32

Line 35

```
1  // FavoritesWebBrowser.java
2  // FavoritesWebBrowser is an application for browsing Web sites
3  // using a WebToolBar and WebBrowserPane and displaying an HTML
4  // page containing links to favorite Web sites.
5  package com.deitel.advjhtp1.gui.splitpane;
6
7  // Java core packages
8  import java.awt.*;
9  import java.awt.event.*;
10 import java.net.*;
11
12 // Java extension packages
13 import javax.swing.*;
14 import javax.swing.event.*;
15
16 // Deitel packages
17 import com.deitel.advjhtp1.gui.webbrowser.*;
18
19 public class FavoritesWebBrowser extends JFrame {
20
21    private WebToolBar toolBar;
22    private WebBrowserPane browserPane;
23    private WebBrowserPane favoritesBrowserPane;
24
25    // WebBrowser constructor
26    public FavoritesWebBrowser()
27    {
28       super( "Deitel Web Browser" );
29
30       // create WebBrowserPane and WebToolBar for na
31       browserPane = new WebBrowserPane();
32       toolBar = new WebToolBar( browserPane );
33
34       // create WebBrowserPane for displaying favorite
35       favoritesBrowserPane = new WebBrowserPane();
```

Create a **WebBrowserPane** for displaying Web pages and a **We** g this **We**

Creates an additional **WebBrowser** Pane to display favorites.html.



Outline

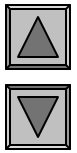
```
36
37          // add WebToolBar as listener for HyperlinkEvents
38          // in favoritesBrowserPane
39          favoritesBrowserPane.addHyperlinkListener( toolBar );
40
41          // display favorites.html in favoritesBrowserPane
42          favoritesBrowserPane.goToURL(
43             getClass().getResource( "favorites.html" ) );
44
45          // create JSplitPane with horizontal split (side-by-sid
46          // and add WebBrowserPanes with JScrollPanes
47          JSplitPane splitPane = new JSplitPane(
48             JSplitPane.HORIZONTAL_SPLIT,
49             new JScrollPane( favoritesBrowserPane ),
50             new JScrollPane( browserPane ) );
51
52          // position divider between WebBrowserPanes
53          splitPane.setDividerLocation( 210 );
54
55          // add buttons for expanding/contracting div
56          splitPane.setOneTouchExpandable( true );
57
58          // lay out WebBrowser components
59          Container contentPane = getContentPane();
60          contentPane.add( toolBar, BorderLayout.NORTH );
61          contentPane.add( splitPane, BorderLayout.CENTER );
62       }
63
```

Invoke method **goToURL** of class **WebBrowserPane** to load **favorites.html** in **favoritesBrowserPane**.

Place each **WebBrowserPane** in a **JScrollPane** and create

Invokes method **setDividerLocation** of cla ex **fa** an

Invokes method **setOneTouchExpandable** of class **JSplitPane** to add two buttons to the divider that enable the user to expand or collapse the divider to one side or the other with a single click.

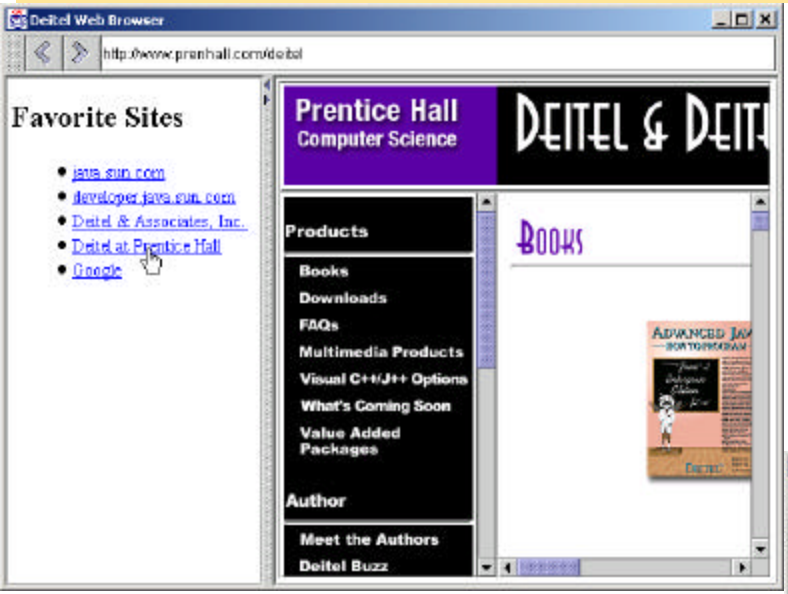


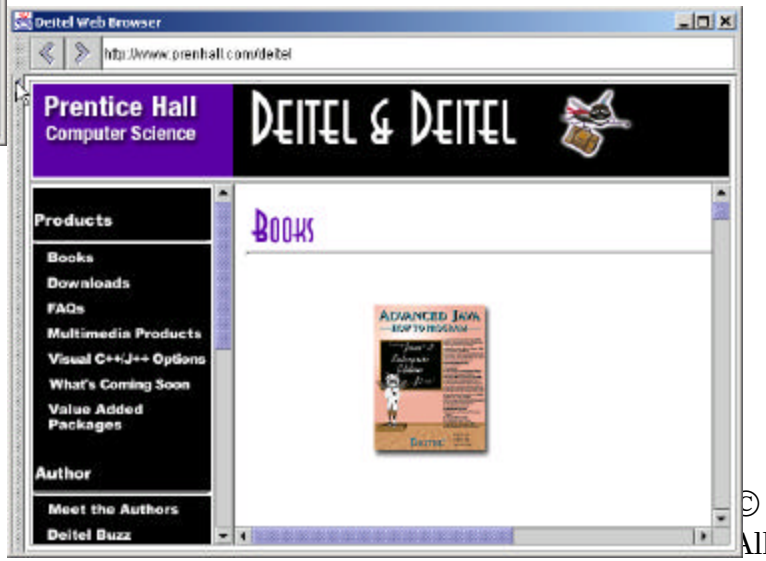
Outline

```
64       // execute application
65       public static void main( String args[] )
66       {
67          FavoritesWebBrowser browser = new FavoritesWebBrowser();
68          browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
69          browser.setSize( 640, 480 );
70          browser.setVisible( true );
71       }
72    }
```

Fig. 2.7 FavoritesWebBrowser application for displaying two Web pages side-by-side using JSplitPane.

Program output





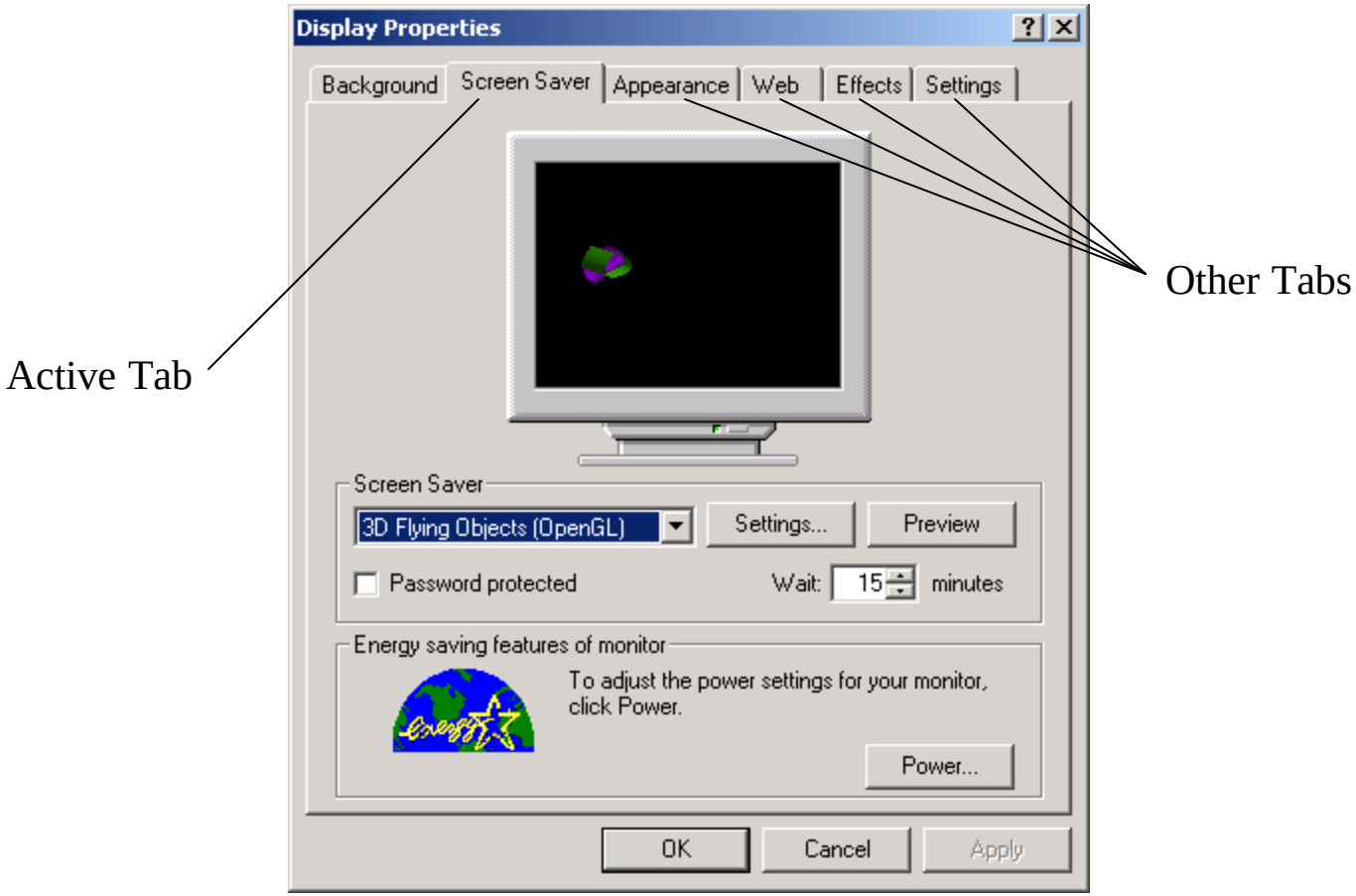
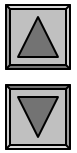


Fig. 2.8 Tabbed interface of Display Properties dialog box in Windows 2000.

Fig. 2.8 Tabbed interface of Display Properties dialog box in Window 2000.



Outline

Fig. 2.9 TabbedPaneWebBrowser application using JTabbedPane to browse multiple Web sites concurrently.

```
1  // TabbedPaneWebBrowser.java
2  // TabbedPaneWebBrowser is an application that uses a
3  // JTabbedPane to display multiple Web browsers.
4  package com.deitel.advjhtp1.gui.tabbedpane;
5
6  // Java core packages
7  import java.awt.*;
8  import java.awt.event.*;
9
10 // Java extension packages
11 import javax.swing.*;
12
13 // Deitel packages
14 import com.deitel.advjhtp1.gui.webbrowser.*;
15
16 public class TabbedPaneWebBrowser extends JFrame {
17
18    // JTabbedPane for displaying multiple browser tabs
19    private JTabbedPane tabbedPane = new JTabbedPane();
20
21    // TabbedPaneWebBrowser constructor
22    public TabbedPaneWebBrowser()
23    {
24       super( "JTabbedPane Web Browser" );
25
26       // create first browser tab
27       createNewTab();
28
29       // add JTabbedPane to contentPa
30       getContentPane().add( tabbedPane );
31
```

Line 19

Creates a new **JTabbedPane** to add **WebBrowserPanes**.

Line 30

Invokes method **createNewTab** of class **TabbedPaneWebBrowser** to create the

Adds the **JTabbedPane** to the **TabbedPaneWebBrowser**'s content pane.

Outline

```
eWebBro
ication
using
JTabbedPane to
browse multiple
b sites
ncurrently.
```

```
32          // create File JMenu for creating new browser tabs and
33          // exiting application
34          JMenu fileMenu = new JMenu( "File" );
35          fileMenu.add( new NewTabAction() );
36          fileMenu.addSeparator();
37          fileMenu.add( new ExitAction() );
38          fileMenu.setMnemonic( 'F' );
39
40          JMenuBar menuBar = new JMenuBar();
41          menuBar.add( fileMenu );
42          setJMenuBar( menuBar );
43
44       } // end TabbedPaneWebBrowser const
45
46       // create new browser tab
47       private void createNewTab()
48       {
49          // create JPanel to contain WebBrowserPane and WebToolBa
50          JPanel panel = new JPanel( new BorderLayout() );
51
52          // create WebBrowserPane and WebToolBar
53          WebBrowserPane browserPane = new WebBrowserPane();
54          WebToolBar toolBar = new WebToolBar( browserPane );
55
56          // add WebBrowserPane and WebToolBar to JPanel
57          panel.add( toolBar, BorderLayout.NORTH );
58          panel.add( new JScrollPane( browserPane ),
59             BorderLayout.CENTER );
60
61          // add JPanel to JTabbedPane
62          tabbedPane.addTab( "Browser " + tabbedPane.getTabCount(),
63             panel );
64       }
65
```

Create the **File** menu, and add an **Action** for creating new **WebBrowserPanes** and an **Action** for exiting the application.

Method **createNewTab** creates a new **WebBrowserPane** and adds it to the **JTabbedPane**.

Creates a **JPanel** for laying out the **WebBrowserPane** and its **WebToolBar**.

Line 50

Create a **WebBrowserPane** and a **WebToolBar**, and add t

Invoke method **addTab** of class **JTabbedPane** to add the **JPanel** containing the **WebBrowserPane** and **WebToolBar** to the application's **JTabbedPane**.

```
66       // Action for creating new browser tabs
67       private class NewTabAction extends AbstractAction {
68
69          // NewTabAction constructor
70          public NewTabAction()
71          {
72             // set name, description and mnemonic key
73             putValue( Action.NAME, "New Browser Tab" );
74             putValue( Action.SHORT_DESCRIPTION,
75                "Create New Web Browser Tab" );
76             putValue( Action.MNEMONIC_KEY, new Integer( 'N' ) );
77          }
78
79          // when Action invoked, create new browser tab
80          public void actionPerformed( A
81          {
82             createNewTab();
83          }
84       }
85
86       // Action for exiting application
87       private class ExitAction extends AbstractAction {
88
89          // ExitAction constructor
90          public ExitAction()
91          {
92             // set name, description and mnemonic key
93             putValue( Action.NAME, "Exit" );
94             putValue( Action.SHORT_DESCRIPTION, "Exit Application" );
95             putValue( Action.MNEMONIC_KEY, new Integer( 'x' ) );
96          }
97
```

Define inner class **NewTabAction**, which extends **AbstractAction**.

Define method **actionPerformed** and invoke method **createNewTab** to create a new tab in the **JTabbedPane** containing a **WebBrowserPane** and **WebToolBar**.

Define inner class **ExitAction**, which extends **AbstractAction**.

Outline

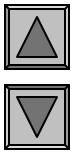
TabbedPaneWebBrowser application using JTabbedPane to browse multiple Web sites concurrently.

Lines 67-84

Lines 80-83

Lines 87-103

```
98          // when Action invoked, exit application
99          public void actionPerformed( ActionEvent event )
100         {
101            System.exit( 0 );
102         }
103      }
104
105      // execute application
106      public static void main( String args[] )
107      {
108         TabbedPaneWebBrowser browser = new TabbedPaneWebBrowser();
109         browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
110         browser.setSize( 640, 480 );
111         browser.setVisible( true );
112      }
113   }
```

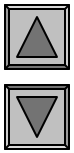


Outline

Fig. 2.9 TabbedPaneWebBrowser application using JTabbedPane to browse multiple Web sites concurrently.

2.5 Multiple-Document Interfaces

- Multiple-Document Interfaces
  - View multiple documents in a single application
- Swing Implementations
  - **`JDesktopPane`** class
  - **`JInternalFrame`** class



Outline

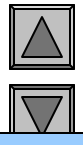
Fig. 2.10 MDIWebBrowser application using JDesktopPane and JInternalFrames to browse multiple Web sites concurrently.

Line 32

```
1    // MDIWebBrowser.java
2    // MDIWebBrowser is an application that uses JDesktopPane
3    // and JInternalFrames to create a multiple-document-interface
4    // application for Web browsing.
5    package com.deitel.advjhtp1.gui.mdi;
6
7    // Java core packages
8    import java.awt.*;
9    import java.awt.event.*;
10
11   // Java extension packages
12   import javax.swing.*;
13
14   // Deitel packages
15   import com.deitel.advjhtp1.gui.webbrowser.*;
16
17   public class MDIWebBrowser extends JFrame {
18
19      // JDesktopPane for multiple document interface
20      JDesktopPane desktopPane = new JDesktopPane();
21
22      // MDIWebBrowser constructor
23      public MDIWebBrowser()
24      {
25         super( "MDI Web Browser" );
26
27         // create first browser window
28         createNewWindow();
29
30         // add JDesktopPane to contentPane
31         Container contentPane = getContentPane();
32         contentPane.add( desktopPane );
33
```

Creates a **JDesktopPane**, which is a container for **JInternalFrame**s.

Adds the **JDesktopPane** to the **Jframe**'s content page.



Outline

```
34          // create File JMenu for creating new windows and
35          // exiting application
36          JMenu fileMenu = new JMenu( "File" );
37          fileMenu.add( new NewWindowAction() );
38          fileMenu.addSeparator();
39          fileMenu.add( new ExitAction() );
40          fileMenu.setMnemonic( 'F' );
41
42          JMenuBar menuBar = new JMenuBar();
43          menuBar.add( fileMenu );
44          setJMenuBar( menuBar );
45       }
46
47       // create new browser window
48       private void createNewWindow()
49       {
50          // create new JInternalFrame that is resizable, closable,
51          // maximizable and iconifiable
52          JInternalFrame frame = new JIntern
53             "Browser", // title
54             true,      // resizable
55             true,      // closable
56             true,      // maximizable
57             true );    // iconifiable
58
59          // create WebBrowserPane and WebToolBar
60          WebBrowserPane browserPane = new WebBrowserPane(
61          WebToolBar toolBar = new WebToolBar( browserPane );
62
63          // add WebBrowserPane and WebToolBar to JInternalFram
64          Container contentPane = frame.getContentPane();
65          contentPane.add( toolBar, BorderLayout.NORTH );
66          contentPane.add( new JScrollPane( browserPane ),
67             BorderLayout.CENTER );
68
```

Construct the application menu. The **File** menu includes an **Action** for creating new browser window and an **Action** for exiting the application.

Method **createNewWindow** creates a new **JInternalFrame** in response to the user invoking **NewWindowAction**.

Create a new **JInternalFrame** with title “Browser” and configure it as resizable, colsable, maximizable and iconifiable.

Create a **WebBrowserPane** and **WebToolBar** for displaying and navigating Web pages.

Get the **JInternalFrame**’s content pane and lay out the **WebToolBar** and **WebBrowserPane** in the content pane.

```
69          // make JInternalFrame opaque a
70          frame.setSize( 320, 240 );
71
72          // move JInternalFrame to prevent it from
73          int offset = 30 * desktopPane.getAllFrame
74          frame.setLocation( offset, offset );
75
76          // add JInternalFrame to JDesk
77          desktopPane.add( frame );
78
79          // make JInternalFrame visibl
80          frame.setVisible( true );
81       }
82
83       // Action for creating new browser windows
84       private class NewWindowAction extends AbstractAction {
85
86          // NewWindowAction constructor
87          public NewWindowAction()
88          {
89             // set name, description and mnemonic key
90             putValue( Action.NAME, "New Window" );
91             putValue( Action.SHORT_DESCRIPTION,
92                "Create New Web Browser Window" );
93             putValue( Action.MNEMONIC_KEY, new Integer( 'N' ) );
94          }
95
96          // when Action invoked, create new browser window
97          public void actionPerformed( ActionEvent event )
98          {
99             createNewWindow();
100         }
101      }
102
```

Invokes method **setSize** of class **JInternalFrame** to size the

Invoke method **setLocation** of class **JInternalFrame** to position the new **JInternalFrame** at an offset from the previously created **JInternalFrame**.

Invokes method **add** of class **JDesktopPane** to add the **JInternalFrame** to the display.

Invokes method **setVisible** of class **JInternalFrame** to make the **JInternalFrame** visible.

Declare inner class **NewWindowAction**.

Pane and

sites concurrently.

Line 70

Lines 73-74

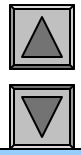
Line 77

Line 80

Lines 84-101

```
103     // Action for exiting application
104     private class ExitAction extends AbstractAction {
105
106        // ExitAction constructor
107        public ExitAction()
108        {
109           // set name, description and mnemonic key
110           putValue( Action.NAME, "Exit" );
111           putValue( Action.SHORT_DESCRIPTION, "Exit Application" );
112           putValue( Action.MNEMONIC_KEY, new Integer( 'x' ) );
113        }
114
115        // when Action invoked, exit application
116        public void actionPerformed( ActionEvent event )
117        {
118           System.exit( 0 );
119        }
120     }
121
122     // execute application
123     public static void main( String args[] )
124     {
125        MDIWebBrowser browser = new MDIWebBrowser();
126        browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
127        browser.setSize( 640, 480 );
128        browser.setVisible( true );
129     }
130  }
```

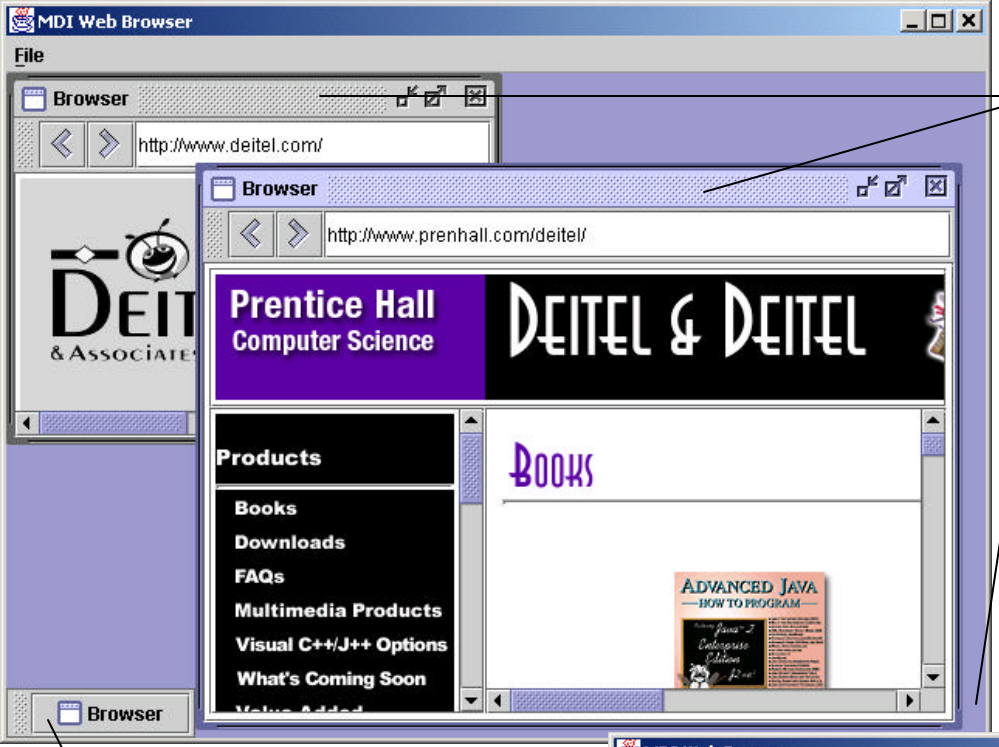
Declare inner class **ExitAction**.



Outline

2.10 MDIWebBrowser application using JDesktopPane and JInternalFrames to browse multiple Web sites concurrently.

Lines 104-120



JInternalFrames

Position the mouse over any corner of a child window to resize the window (if resizing is allowed).

Iconify

Maximize

Close

Iconified **JInternalFrame**

Maximized **JInternalFrame**

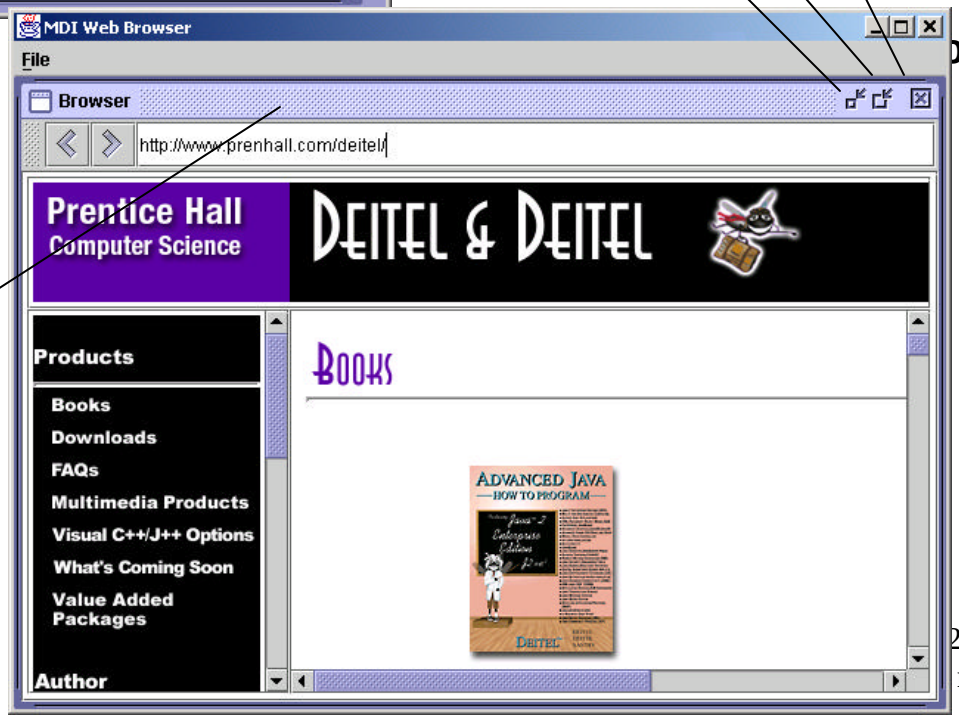
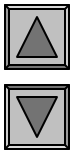


Fig. 2.10 MDIWebBrowser application using JDesktopPane and JInternalFrames to browse multiple Web sites concurrently.

ogram output

2.6 Drag and Drop

- Drag and Drop
  - Move items around the desktop
  - Move and copy data among applications using mouse gestures
- Java APIs
  - Data transfer API
  - Drag-and-drop API



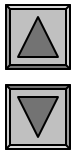
Outline

```
1    // DnDWebBrowser.java
2    // DnDWebBrowser is an application for viewing Web pages using
3    // drag and drop.
4    package com.deitel.advjhtp1.gui.dnd;
5
6    // Java core packages
7    import java.awt.*;
8    import java.awt.dnd.*;
9    import java.awt.datatransfer.*;
10   import java.util.*;
11   import java.io.*;
12   import java.net.*;
13
14   // Java extension packages
15   import javax.swing.*;
16   import javax.swing.event.*;
17
18   // Deitel packages
19   import com.deitel.advjhtp1.gui.webbrowser.*;
20
21   public class DnDWebBrowser extends JFrame {
22
23      private WebToolBar toolBar;
24      private WebBrowserPane browserPane;
25
26      // DnDWebBrowser constructor
27      public DnDWebBrowser()
28      {
29         super( "Drag-and-Drop Web Browser" );
30
31         // create WebBrowserPane and WebToolBar for navigation
32         browserPane = new WebBrowserPane();
33         toolBar = new WebToolBar( browserPane );
34
```

Create a **WebBrowserPane** component for viewing Web pages and a **WebToolBar** to provide navigation controls.

Fig. 2.11 DnDWebBrowser application for browsing Web sites that also accepts drag-and-drop operations for viewing HTML pages.

Lines 32-33



Outline

```
35          // enable WebBrowserPane to accept drop operations, using
36          // DropTargetHandler as the DropTargetListener
37          browserPane.setDropTarget( new DropTarget( browserPane,
38             DnDConstants.ACTION_COPY, new DropTargetHandler() ) );
39
40          // lay out WebBrowser components
41          Container contentPane = getContentPane();
42          contentPane.add( toolBar, BorderLayout.NORTH
43          contentPane.add( new JScrollPane( browserPane
44             BorderLayout.CENTER );
45       }
46
47       // inner class to handle DropTargetEvents
48       private class DropTargetHandler implements DropTarget
49
50          // handle drop operation
51          public void drop( DropTargetDropEvent event )
52          {
53             // get dropped Transferable object
54             Transferable transferable = event.getTransfer
55
56             // if Transferable is a List of Files, accept
57             if ( transferable.isDataFlavorSupported(
58                DataFlavor.javaFileListFlavor ) ) {
59
60                // accept the drop operation to copy the object
61                event.acceptDrop( DnDConstants.ACTION_COPY );
62
```

Invoke method **setDropTarget** of class **WebBrowserPane** and

Class **DropTargetHandler** implements **interfaceDropTargetListener** to listen

Method drop of interface **DropTargetListener** was

Invokes method **getTransferable** of class **DropTargetDropEvent** to

Invoke method **isDataFlavorSupported** of interface **Transferable** to

Invokes method **acceptDrop** of class **DropTargetDropEvent** to indicate that the drop operation is allowed for this **DropTarget**.

Lines 57-58

Line 61

```
63             // process list of files and di
64             try {
65
66                // get List of Files
67                java.util.List fileList =
68                   ( java.util.List ) transferable.getTransferData(
69                      DataFlavor.javaFileListFlavo
70
71                Iterator iterator = fileList.itera
72
73                while ( iterator.hasNext() ) {
74                   File file = ( File ) iterator.next();
75
76                   // display File in browser an
77                   browserPane.goToURL( file.toU
78                }
79
80                // indicate successful drop
81                event.dropComplete( true );
82             }
83
84             // handle exception if DataFlavor not supported
85             catch ( UnsupportedFlavorException flavorException ) {
86                flavorException.printStackTrace();
87                event.dropComplete( false );
88             }
89
90             // handle exception reading Transferable data
91             catch ( IOException ioException ) {
92                ioException.printStackTrace();
93                event.dropComplete( false );
94             }
95          }
96
```

Retrieve the List of **File**s from the **Transferable** object by invoking method **getTransfeData** of interface **Transferable**.

Iterate the List of Files, displaying each by invoking method **goToURL** of class **WebBrowserPane**.

Invokes method **dropComplete** of class **DropTargetDropEvent** with a true argument to indicate that the drag-and-drop operation was successful.

Fig. 2.11
DnDWebBrowser
n for
eb
also
accepts drag-

Lines 67-69

Lines 73-78

Line 80

```
97             // if dropped object is not file list, reject dro
98             else
99                event.rejectDrop();
100         }
101
102         // handle drag operation entering DropTarget
103         public void dragEnter( DropTargetDragEvent even
104         {
105            // if data is javaFileListFlavor, accept dra
106            if ( event.isDataFlavorSupported(
107               DataFlavor.javaFileListFla
108
109               event.acceptDrag( DnDConst
110
111            // reject all other DataFlavo
112            else
113               event.rejectDrag();
114         }
115
116         // invoked when drag operation exits DropTarget
117         public void dragExit( DropTargetEvent event ) {}
118
119         // invoked when drag operation occurs over DropTarget
120         public void dragOver( DropTargetDragEvent event ) {}
121
122         // invoked if dropAction changes (e.g., from COPY to LINK)
123         public void dropActionChanged( DropTargetDragEvent event )
124         {}
125
126      } // end class DropTargetHandler
127
```

Method **dragEnter** of interface **DropTargetListener** is invoked when a drag-and-drop operation enters a **DropTarget**.

Invokes method **acceptDrag** of class **DropTargetDragEvent** to indicate that this **DropTarget**

Invokes method **rejectDrag** of class **DropTargetDragEvent** to indicate that the **DropTarget** does not allow this drag-and-drop operation.

pages.

Lines 103-114

Lines 106-107

Line 109

Line 113

```
128     // execute application
129     public static void main( String args[] )
130     {
131        DnDWebBrowser browser = new DnDWebBrowser();
132        browser.setDefaultCloseOperation( EXIT_ON_CLOSE );
133        browser.setSize( 640, 480 );
134        browser.setVisible( true );
135     }
136  }
```

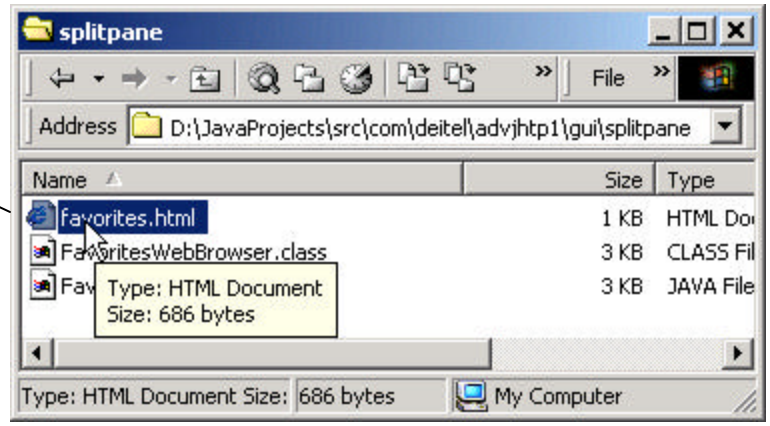
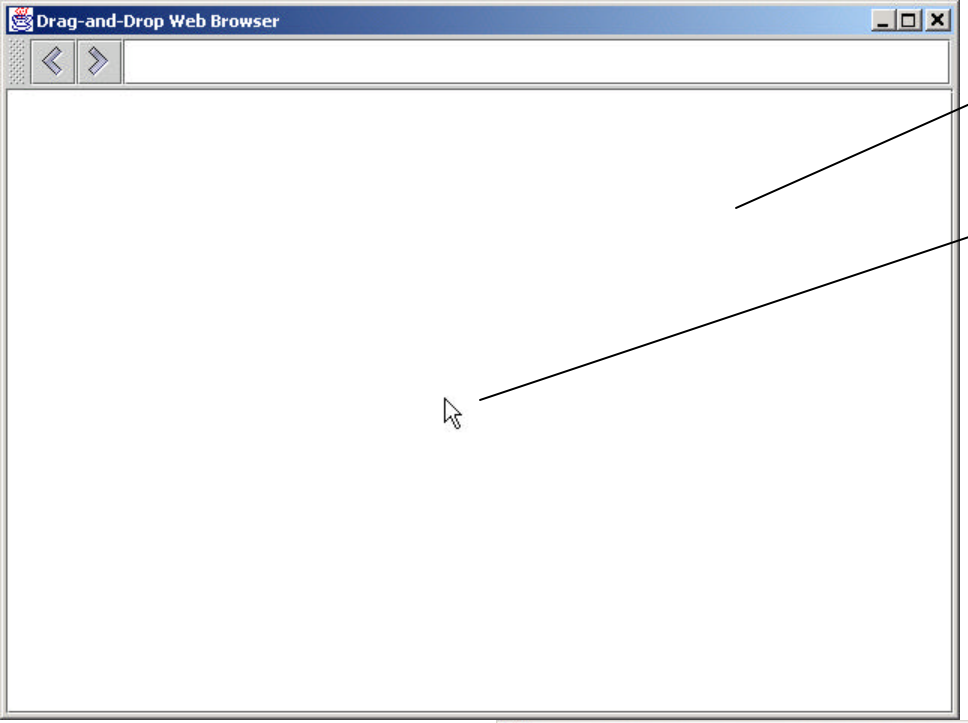


Fig. 2.11 DnDWebBrowser application for browsing Web sites that also accepts drag-and-drop operations for viewing HTML pages.

Program output



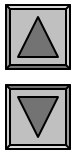
WebBrowserPane displaying **favorites.html**

Fig. 2.11 DnDWebBrowser application for browsing Web sites that also accepts drag-and-drop operations for viewing HTML pages.

Program output

2.7 Internationalization

- Locale
  - Identify entities that present in a particular country or region
- Internationalization
  - Using **`ResourceBoundle`** to code strings
  - Using local-sensitive classes to format data
    - **`NumberFormat`** and **`DateFormat`**
- Unicode



Outline

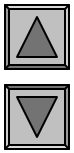
Fig. 2.12 WebToolBar that uses Resourcebundles for internationalization.

Lines 30-31

```
1    // WebToolBar.java
2    // Internationalized WebToolBar with components for navigating
3    // a WebBrowserPane.
4    package com.deitel.advjhtp1.gui.i18n;
5
6    // Java core packages
7    import java.awt.*;
8    import java.awt.event.*;
9    import java.net.*;
10   import java.util.*;
11
12   // Java extension packages
13   import javax.swing.*;
14   import javax.swing.event.*;
15
16   // Deitel packages
17   import com.deitel.advjhtp1.gui.webbrowser.WebBrowserPane;
18   import com.deitel.advjhtp1.gui.actions.MyAbstractActio
19
20   public class WebToolBar extends JToolBar
21      implements HyperlinkListener {
22
23      private WebBrowserPane webBrowserPane;
24      private JTextField urlTextField;
25
26      // WebToolBar constructor
27      public WebToolBar( WebBrowserPane browser, Loca
28      {
29         // get resource bundle for internationalized
30         ResourceBundle resources = ResourceBundle.ge
31            "StringsAndLabels", locale );
32
33         setName( resources.getString( "toolBarTitle" ) );
34
```

Load the **ResourceBundle** named **StringsAndLabels** for the given Locale by invoking method **getBundle**

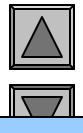
Invokes method **getString** of class **ResourceBundle** to retrieve the **toolBarTitle** string from the **ResourceBundle** and invokes method **setName** of class **JToolBar** to set the **JToolBar**'s name to the retrieved value.



Outline

```
35          // register for HyperlinkEvents
36          webBrowserPane = browser;
37          webBrowserPane.addHyperlinkListener( this );
38
39          // create JTextField for entering URLs
40          urlTextField = new JTextField( 25 );
41          urlTextField.addActionListener(
42             new ActionListener() {
43
44                // navigate webBrowser to user-entered URL
45                public void actionPerformed( ActionEvent event )
46                {
47                   // attempt to load URL in webBrowserPane
48                   try {
49                      URL url = new URL( urlTextField.getText() );
50                      webBrowserPane.goToURL( url );
51                   }
52
53                   // handle invalid URL
54                   catch ( MalformedURLException urlException ) {
55                      urlException.printStackTrace();
56                   }
57                }
58             }
59          );
60
```

Fig. 2.12 WebToolBar that uses Resourcebundles for internationalization.



Outline

```
61          // create backAction and configure its properties
62          MyAbstractAction backAction = new MyAbstractAction() {
63
64             public void actionPerformed( ActionEvent event )
65             {
66                // navigate to previous URL
67                URL url = webBrowserPane.back();
68
69                // display URL in urlTextField
70                urlTextField.setText( url.toString() );
71             }
72          };
73
74          backAction.setSmallIcon( new ImageIcon(
75             getClass().getResource( "images/back.gif
76
77          backAction.setShortDescription(
78             resources.getString( "backToolTip" ) );
79
80          // create forwardAction and configure its properties
81          MyAbstractAction forwardAction = new MyAbstractA
82
83             public void actionPerformed( ActionEvent event
84             {
85                // navigate to next URL
86                URL url = webBrowserPane.forward();
87
88                // display new URL in urlTextField
89                urlTextField.setText( url.toString() );
90             }
91          };
92
```

Create an instance of class **MyAbstractAction** for the **WebToolBar**'s **backAction**.

Load the **Icon** for **backAction**.

Retrieve the internationalized tooltip text for **backAction** from the **ResourceBundle**.

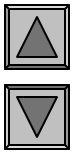
Create an instance of class **MyAbstractAction** for the **WebToolBar**'s **forwardAction**.

uses Resourcebundles for liza tion.

2-72

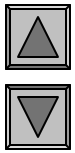
Lines 74-75

```
93          forwardAction.setSmallIcon( new ImageIcon(
94             getClass().getResource( "images/forward.gif" ) ) );
95
96          forwardAction.setShortDescription(
97             resources.getString( "forwardToolTip" ) );
98
99          // add JButtons and JTextField to WebToolBar
100         add( backAction );
101         add( forwardAction );
102         add( urlTextField );
103
104      } // end WebToolBar constructor
105
106      // listen for HyperlinkEvents in WebBrowserPane
107      public void hyperlinkUpdate( HyperlinkEvent event )
108      {
109         // if hyperlink was activated, go to hyperlink's URL
110         if ( event.getEventType() ==
111            HyperlinkEvent.EventType.ACTIVATED ) {
112
113            // get URL from HyperlinkEvent
114            URL url = event.getURL();
115
116            // navigate to URL and display URL in urlTextField
117            webBrowserPane.goToURL( event.getURL() );
118            urlTextField.setText( url.toString() );
119         }
120      }
121   }
```



Outline

Fig. 2.12 WebToolBar that uses Resourcebundles for internationalization.



Outline

Fig. 2.13 MyAbstractAction AbstractAction subclass that provides set methods for Action

```
1   // MyAbstractAction.java
2   // MyAbstractAction is an AbstractAction subclass that provides
3   // set methods for Action properties (e.g., name, icon, etc.).
4   package com.deitel.advjhtp1.gui.actions;
5
6   // Java core packages
7   import java.awt.event.*;
8
9   // Java extension packages
10  import javax.swing.*;
11
12  public abstract class MyAbstractAction extends AbstractAction {
13
14     // no-argument constructor
15     public MyAbstractAction() {}
16
17     // construct MyAbstractAction with given name, icon
18     // description and mnemonic key
19     public MyAbstractAction( String name, Icon icon,
20        String description, Integer mnemonic
21     {
22        // initialize properties
23        setName( name );
24        setSmallIcon( icon );
25        setShortDescription( description );
26        setMnemonic( mnemonic );
27     }
28
29     // set Action name
30     public void setName( String name )
31     {
32        putValue( Action.NAME, name );
33     }
34
```

Constructor takes as arguments the name, **Icon**, description and mnemonic key for the **Action**.

Invoke the appropriate set methods to configure the **Action** to the given values.

```
35      // set Action Icon
36      public void setSmallIcon( Icon icon )
37      {
38         putValue( Action.SMALL_ICON, icon );
39      }
40
41      // set Action short description
42      public void setShortDescription( String description )
43      {
44         putValue( Action.SHORT_DESCRIPTION, description );
45      }
46
47      // set Action mnemonic key
48      public void setMnemonic( Integer mnemonic )
49      {
50         putValue( Action.MNEMONIC_KEY, mnemonic );
51      }
52
53      // abstract actionPerformed method to be implemented
54      // by concrete subclasses
55      public abstract void actionPerformed( ActionEvent event );
56   }
```

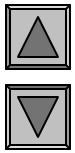
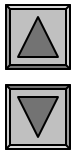


Fig. 2.13 MyAbstractAction AbstractAction subclass that provides set methods for Action properties.



Outline

Fig. 2.14 WebBrowser that uses ResourceBundles for internationalization.

Lines 26-42

Lines 28-29

Line 31

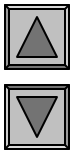
```
1   // WebBrowser.java
2   // WebBrowser is an application for browsing Web sites using
3   // a WebToolBar and WebBrowserPane.
4   package com.deitel.advjhtp1.gui.i18n;
5
6   // Java core packages
7   import java.awt.*;
8   import java.awt.event.*;
9   import java.net.*;
10  import java.util.*;
11
12  // Java extension packages
13  import javax.swing.*;
14  import javax.swing.event.*;
15
16  // Deitel packages
17  import com.deitel.advjhtp1.gui.webbrowser.WebBrowserPane;
18
19  public class WebBrowser extends JFrame {
20
21     private ResourceBundle resources;
22     private WebToolBar toolBar;
23     private WebBrowserPane browserPane;
24
25     // WebBrowser constructor
26     public WebBrowser( Locale locale )
27     {
28        resources = ResourceBundle.getBundle(
29           "StringsAndLabels", locale );
30
31        setTitle( resources.getString( "applicationTitle"
32
```

Constructor takes as an argument the **Lo** applic

Invoke static method **getBundle** of class **ResourceBundle** to load the

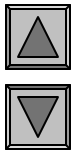
Invokes method **getString** of class **ResourceBundle** to retrieve the **applicationTitle** string.

```
33         // create WebBrowserPane and WebToolBar for navigation
34         browserPane = new WebBrowserPane();
35         toolBar = new WebToolBar( browserPane, locale );
36
37         // lay out WebBrowser components
38         Container contentPane = getContentPane();
39         contentPane.add( toolBar, BorderLayout.NORTH );
40         contentPane.add( new JScrollPane( browserPane ),
41            BorderLayout.CENTER );
42      }
43   }
```



Outline

Fig. 2.14 WebBrowser that uses ResourceBundles for internationalization.



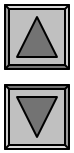
Outline

Fig. 2.15 BrowserLauncher application for selecting a Locale and launching an internationalized WebBrowser.

Lines 25-34

```
1  // BrowserLauncher.java
2  // BrowserLauncher provides a list of Locales and launches a new
3  // Internationalized WebBrowser for the selected Locale.
4  package com.deitel.advjhtp1.gui.i18n;
5
6  // Java core packages
7  import java.awt.*;
8  import java.awt.event.*;
9  import java.util.*;
10
11 // Java extension packages
12 import javax.swing.*;
13
14 public class BrowserLauncher extends JFrame {
15
16    // JComboBox for selecting Locale
17    private JComboBox localeComboBox;
18
19    // BrowserLauncher constructor
20    public BrowserLauncher()
21    {
22       super( "Browser Launcher" );
23
24       // create JComboBox and add Locales
25       localeComboBox = new JComboBox();
26
27       // United States, English
28       localeComboBox.addItem( Locale.US );
29
30       // France, French
31       localeComboBox.addItem( Locale.FRANCE );
32
33       // Russia, Russian
34       localeComboBox.addItem( new Locale( "ru", "RU" ) );
35
```

Create a **JComboBox** and add sample Locales to the **JComboBox**.



Outline

```
36         // launch new WebBrowser when Locale selection changes
37         localeComboBox.addItemListener(
38            new ItemListener() {
39
40               public void itemStateChanged( ItemEvent event )
41               {
42                  if ( event.getStateChange() == ItemEvent.SELECTED )
43                     launchBrowser( ( Locale )
44                        localeComboBox.getSelectedItem() );
45               }
46            }
47         );
48
49         // lay out components
50         Container contentPane = getContentPane();
51         contentPane.setLayout( new FlowLayout() );
52         contentPane.add( new JLabel( "Select Locale" ) );
53         contentPane.add( localeComboBox );
54      }
55
56      // launch new WebBrowser for given Locale
57      private void launchBrowser( Locale locale )
58      {
59         WebBrowser browser = new WebBrowser( locale );
60         browser.setDefaultCloseOperation( DISPOSE_ON_CLOSE );
61         browser.setSize( 640, 480 );
62         browser.setVisible( true );
63      }
64
```

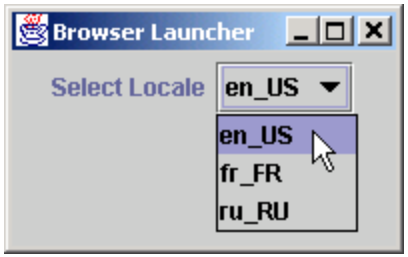
Fig. 2.15 BrowserLauncher application for

Invoke method **launchBrowser** of class **BrowserLauncher** to launch a new **WebBrowser** when the user selects a Locale from the **JComboBox**.

Lines 43-44

Method **launchBrowser** creates a new **WebBrowser** for the given **Locale**, set its size and displays it.

```
65      // execute application
66      public static void main( String args[] )
67      {
68         BrowserLauncher launcher = new BrowserLauncher();
69         launcher.setDefaultCloseOperation( EXIT_ON_CLOSE );
70         launcher.setSize( 200, 125 );
71         launcher.setVisible( true );
72      }
73   }
```



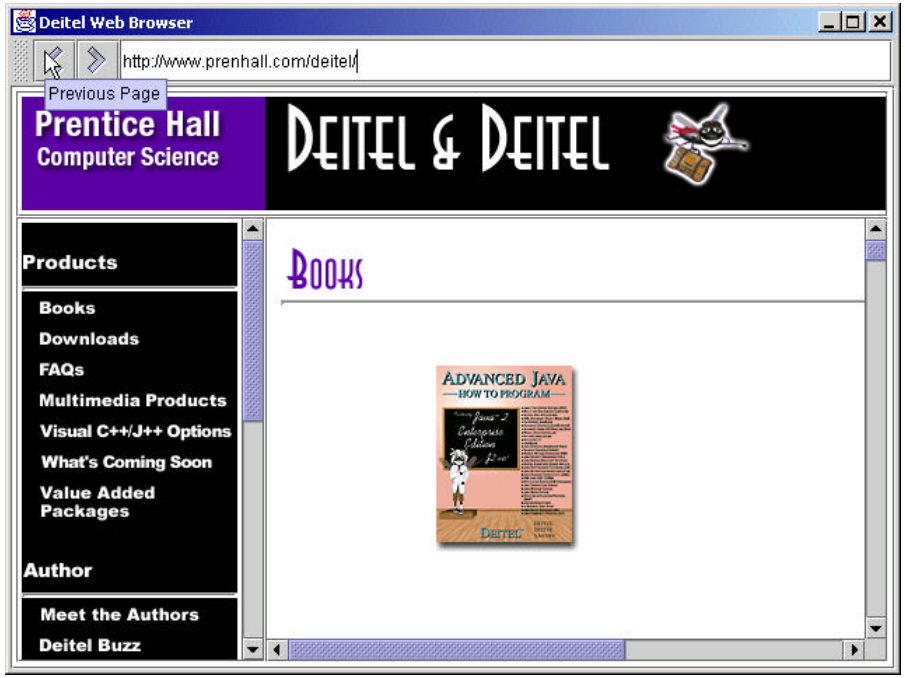
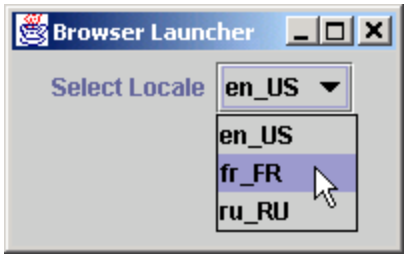


Fig. 2.15 BrowserLauncher application for selecting a Locale and launching an internationalized WebBrowser.

Program output



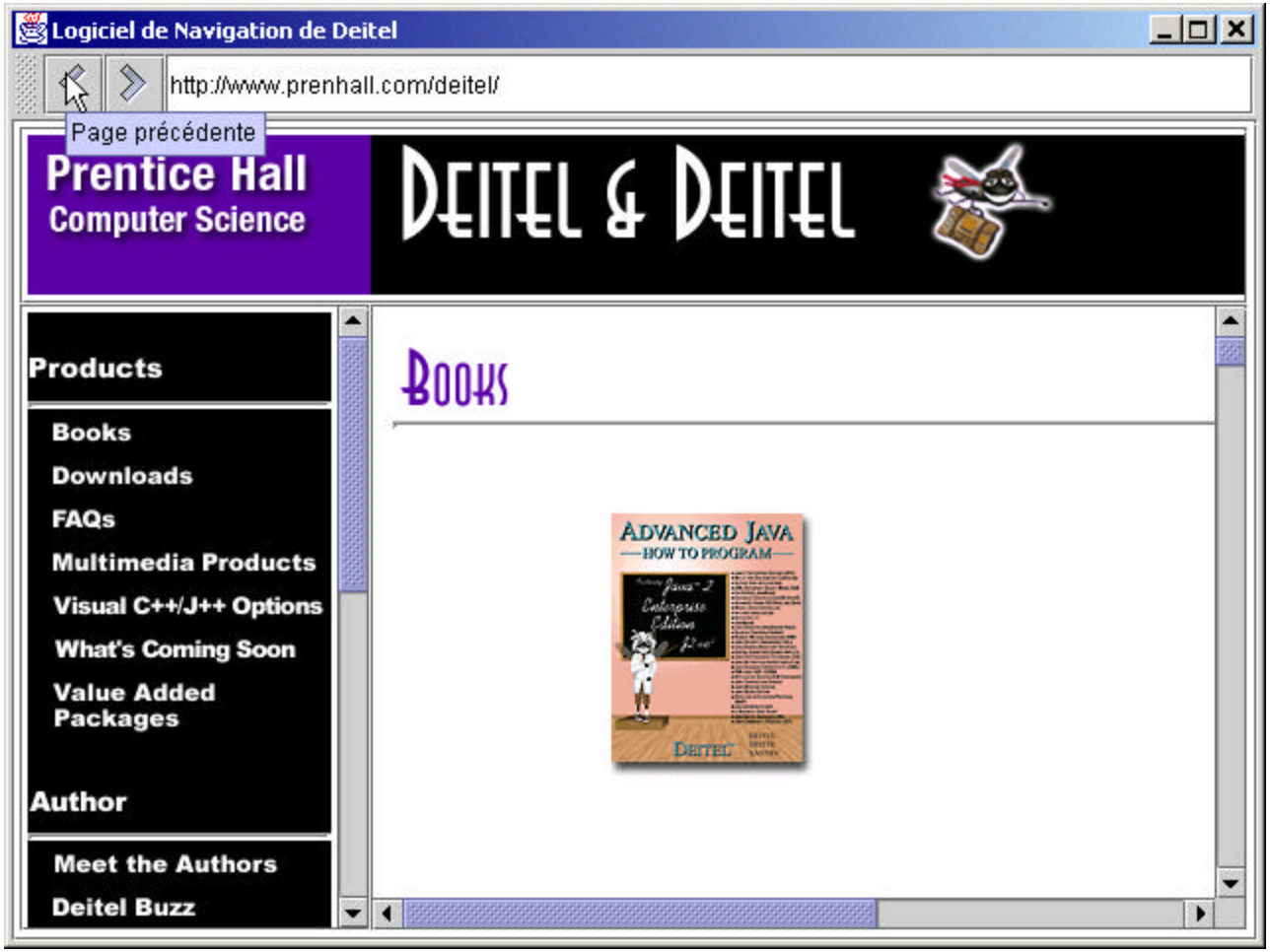
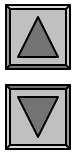


Fig. 2.15 BrowserLauncher application for selecting a Locale and launching an internationalized WebBrowser.

Program output

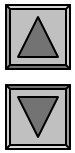
```
1    # English language strings for internationalized WebBrowser
2    # application title
3    applicationTitle = Deitel Web Browser
4
5    # title for WebToolBar
6    toolBarTitle = Web Navigation
7
8    # tooltip for forward toolbar button
9    forwardToolTip = Next Page
10
11   # tooltip for back button
12   backToolTip = Previous Page
```



Outline

Fig. 2.16 Properties file for default Locale (US English) – StringsAndLabels.properties.

```
1    # French language strings for internationalized WebBrowser
2    # tooltip for back button
3    backToolTip = Page pr\u00E9c\u00E9dente
4
5    # application title
6    applicationTitle = Logiciel de Navigation de Deitel
7
8    # title for WebToolBar
9    toolBarTitle = Navigation des Pages sur la Toile
10
11   # tooltip for forward toolbar button
12   forwardToolTip = Prochaine Page
```

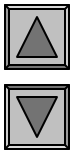


Outline

Fig. 2.17 Properties file for French Locale – StringsAndLabels_fr_FR.properties.

2.8 Accessibility

- Accessibility
  - Application accessible to people with disabilities
- Java Accessibility API
  - Tooltip text
  - Descriptive text
    - **`setAccessibleName`**
    - **`setAccessibleDescription`**



Outline

Fig. 2.18 ActionSamle2 demonstrates Accessibility package.

Lines 26-50

```
1   // ActionSample2.java
2   // ActionSample2 demonstrates the Accessibility features of
3   // Swing components.
4   package com.deitel.advjhtp1.gui.actions;
5
6   // Java core packages
7   import java.awt.*;
8   import java.awt.event.*;
9
10  // Java extension packages
11  import javax.accessibility.*;
12  import javax.swing.*;
13
14  public class ActionSample2 extends JFrame {
15
16     // Swing Actions
17     private Action sampleAction;
18     private Action exitAction;
19
20     // ActionSample2 constructor
21     public ActionSample2()
22     {
23        super( "Using Actions" );
24
25        // create AbstractAction subclass for sampleActio
26        sampleAction = new AbstractAction() {
27
28           public void actionPerformed( ActionEvent event )
29           {
30              // display message indicating sampleAction invoked
31              JOptionPane action = new JOptionPane(
32                 "The sampleAction was invoked." );
33
```

Action **sampleAction** contains accessible text in the dialog box that opens when **sampleAction** is fired.

```
34             // get AccessibleContext for action and
35             // and description
36             AccessibleContext actionContext =
37                action.getAccessibleContext();
38             actionContext.setAccessibleName( "sampleAction" );
39             actionContext.setAccessibleDescription(
40                "SampleAction opens a dialog box to demonstrate"
41                + " the Action class." );
42
43             // create and display dialog box
44             action.createDialog( ActionSample2.this,
45                "sampleAction" ).setVisible( true );
46
47             // enable exitAction and associated GUI components
48             exitAction.setEnabled( true );
49          }
50       };
51
52       // set Action name
53       sampleAction.putValue( Action.NAME, "Sample Action" );
54
55       // set Action Icon
56       sampleAction.putValue( Action.SMALL_ICON, new ImageIcon(
57          getClass().getResource( "images/Help24.gif" ) ) );
58
59       // set Action short description (tooltip text)
60       sampleAction.putValue( Action.SHORT_DESCRIPTION,
61          "A Sample Action" );
62
63       // set Action mnemonic key
64       sampleAction.putValue( Action.MNEMONIC_KEY,
65          new Integer( 'S' ) );
66
```

Declare an **AccessibleContext** object for the **JOptionPane** action by calling method **getAccessibleContext** on **action**.

Sets action's name in

ActionSamle2

t.

Call method **setAccessibleDescription** of class **AccessibleContext** to set **sampleAction**'s description.

Line 38

Lines 39-41

Lines 60-61

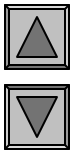
Specify a short description for **sampleAction**.

```
67          // create AbstractAction subclass for exitAction
68          exitAction = new AbstractAction() {
69
70             public void actionPerformed( ActionEvent event )
71             {
72                // display message indicating sampleAction invoked
73                JOptionPane exit = new JOptionPane(
74                   "The exitAction was invoked." );
75
76                // get AccessibleContext for exit and set name and
77                // description
78                AccessibleContext exitContext =
79                   exit.getAccessibleContext();
80                exitContext.setAccessibleName( "exitAction" );
81                exitContext.setAccessibleDescription( "ExitAction"
82                   + " opens a dialog box to demonstrate the"
83                   + " Action class and then exits the program." );
84
85                // create and display dialog box
86                exit.createDialog( ActionSample2.this,
87                   "exitAction" ).setVisible( true );
88
89                // exit program
90                System.exit( 0 );
91             }
92          };
93
94          // set Action name
95          exitAction.putValue( Action.NAME, "Exit" );
96
97          // set Action icon
98          exitAction.putValue( Action.SMALL_ICON, new ImageIcon(
99             getClass().getResource( "images/EXIT.gif" ) ) );
100
```

Action **exitAction** contains accessible text in the dialog box that opens when **exitAction** is fired.

demonstrates Accessibility package.

Lines 58-92



Outline

Fig. 2.18 ActionSamle2 demonstrates Accessibility package.

```
101        // set Action short description (tooltip text)
102        exitAction.putValue( Action.SHORT_DESCRIPTION,
103           "Exit Application" );
104
105        // set Action mnemonic key
106        exitAction.putValue( Action.MNEMONIC_KEY,
107           new Integer( 'x' ) );
108
109        // disable exitAction and associated GUI components
110        exitAction.setEnabled( false );
111
112        // create File menu
113        JMenu fileMenu = new JMenu( "File" );
114
115        // add sampleAction and exitAction to File menu
116        // create a JMenuItem for each Action
117        fileMenu.add( sampleAction );
118        fileMenu.add( exitAction );
119
120        fileMenu.setMnemonic( 'F' );
121
122        // create JMenuBar and add File menu
123        JMenuBar menuBar = new JMenuBar();
124        menuBar.add( fileMenu );
125        setJMenuBar( menuBar );
126
127        // create JToolBar
128        JToolBar toolBar = new JToolBar();
129
130        // add sampleAction and exitAction to JToolBar to create
131        // JButtons for each Action
132        toolBar.add( sampleAction );
133        toolBar.add( exitAction );
134
```

Create a **fileMenu** , add **sampleAction** and **exitAction** to the **fileMenu**, and set mnemonic key for the **fileMenu**.

```
135        // get AccessibleContext for toolBar and
136        // description
137        AccessibleContext toolContext =
138           toolBar.getAccessibleContext();
139        toolContext.setAccessibleName( "ToolBar" );
140        toolContext.setAccessibleDescription( "ToolBar cont
141           + " sampleAction button and exitAction button." );
142
143        // create JButton and set its Action to sampleA
144        JButton sampleButton = new JButton();
145        sampleButton.setAction( sampleAction );
146
147        // get AccessibleContext for sampleButton and s
148        // and description
149        AccessibleContext sampleContext =
150           sampleButton.getAccessibleContext();
151        sampleContext.setAccessibleName( "SampleButton" );
152        sampleContext.setAccessibleDescription( "SampleButton"
153           + " produces a sampleAction event." );
154
155        // create JButton and set its Action to exitAction
156        JButton exitButton = new JButton( exitAction );
157
158        // get AccessibleContext for exitButton and set name and
159        // description
160        AccessibleContext exitContext =
161           exitButton.getAccessibleContext();
162        exitContext.setAccessibleName( "ExitButton" );
163        exitContext.setAccessibleDescription( "ExitButton"
164           + " produces an exitAction event." );
165
166        // lay out JButtons in JPanel
167        JPanel buttonPanel = new JPanel();
168        buttonPanel.add( sampleButton );
169        buttonPanel.add( exitButton );
```

Obtain the **AccessibleContext** for **toolBar**.

Sets toolbar's name in **AccessibleContext**.

Call method **setAccessibleDescription** of class **AccessibleContext** to set toobar's description.

tline

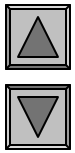
le2

demonstrates

Line 139

Lines 140-141

```
170
171        // add toolBar and buttonPanel to JFrame's content pane
172        Container container = getContentPane();
173        container.add( toolBar, BorderLayout.NORTH );
174        container.add( buttonPanel, BorderLayout.CENTER );
175
176     }
177
178     // execute application
179     public static void main( String args[] )
180     {
181        ActionSample2 sample = new ActionSample2();
182        sample.setDefaultCloseOperation( EXIT_ON_CLOSE );
183        sample.pack();
184        sample.setVisible( true );
185     }
186  }
```



Outline

Fig. 2.18 ActionSamle2 demonstrates Accessibility package.

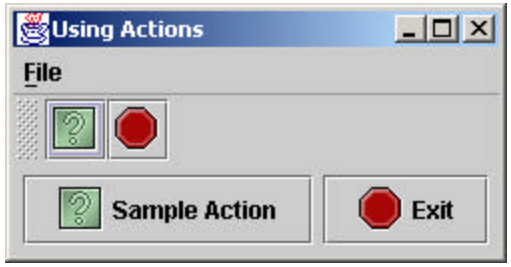


Fig. 2.19 Actions sampleAction and exitAction of ActionSample2.

Program output

Fig. 2.19 **Action**s **sampleAction** and **exitAction** of **ActionSample2**.

Fig. 2.20 AccessibleDescription of sampleButton.

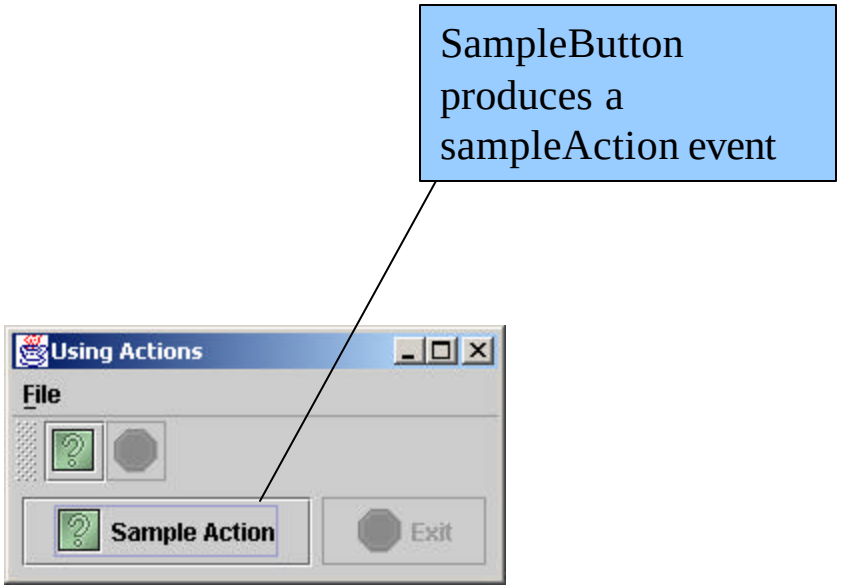


Fig. 2.20 **`AccessibleDescription`** of **`sampleButton`**.

Fig. 2.21 AccessibleDesceiption of exitButton.

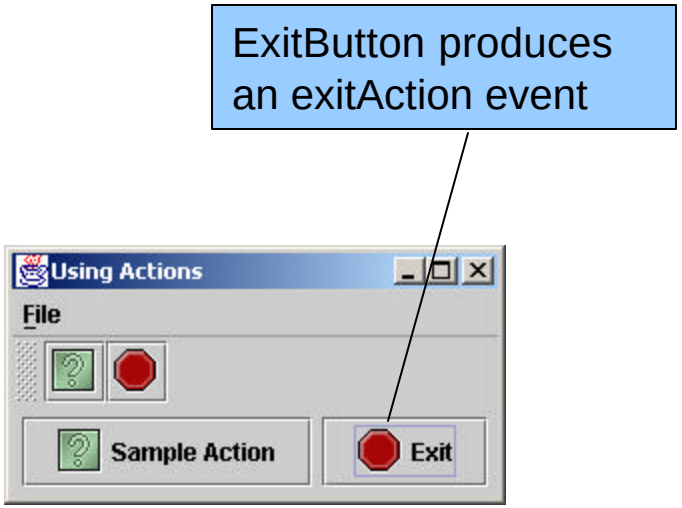


Fig. 2.21 **`AccessibleDescription`** of **`exitButton`**.

Fig. 2.22 Sample **`Action`** menu item description.

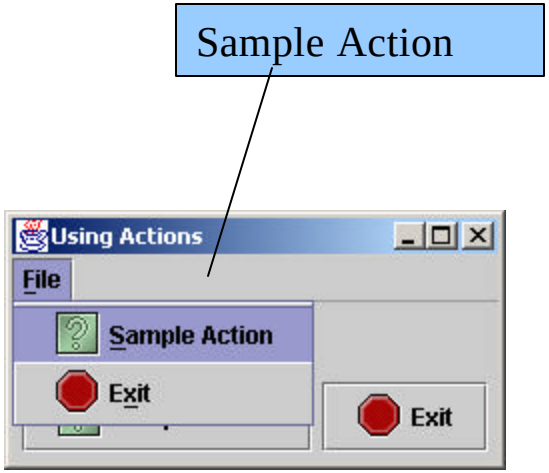


Fig. 2.22 Sample **`Action`** menu item description.

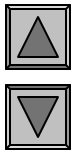


Fig. 2.23 Exit menu item description.

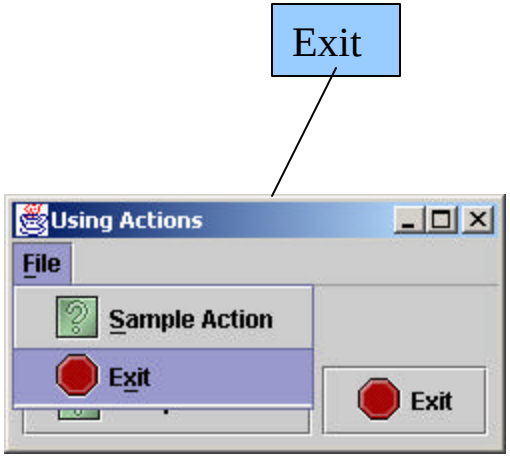


Fig. 2.23 Exit menu item description.