Appendix C

Swing Components

Objectives

At the end of this appendix, you should be able to:

- Describe the appearance of each Swing component.

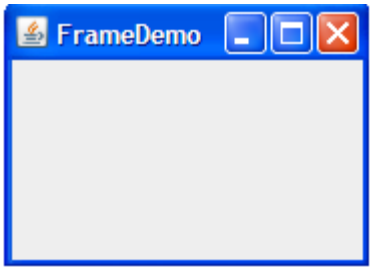
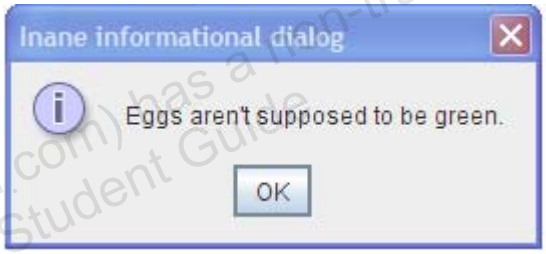
Swing Component Examples

This appendix shows examples of many of the Swing components along with a brief description of that component.

Top Level Containers

Table C-1 contains a brief description of each top-level container.

Table C-1 Top-level Containers

Container	**User Interface**
A Frame is the basic window used in most GUI applications. It has a border and a title. Other components can be added to the frame. You can also draw in the window. There is a provision to add menus to the frame. The `javax.swing.JFrame` class is used to create frames.	
`JDialog` is used to create a dialog window. The API provides several different versions of constructors for defining dialogs. Dialogs are dependent on the frames. Dialogs can be used to take input from the user and confirm any critical actions. Dialogs can be used to display warnings, errors, questions, and information to the user.	
The `JWindow` container is similar to `JFrame` but it does not have a border or title bar. There are no window management services.	
The `JApplet` container is used to create a UI that is run in a web browser. Usually, `JApplets` are embedded in a web page and can be used to run animations. Other components and menus can be added to this container. You can also draw in an applet.	

General-purpose Containers

General-purpose containers are intermediate containers, for use in several circumstances, for example: `JPanel`, `JScrollPane`, `JToolBar`, `JSplitPane`, and `JTabbedPane`. All of these components extend `JComponent`. Table C-2 contains a brief description of each general-purpose container.

Table C-2 General-purpose Containers

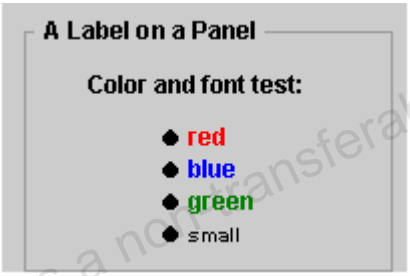
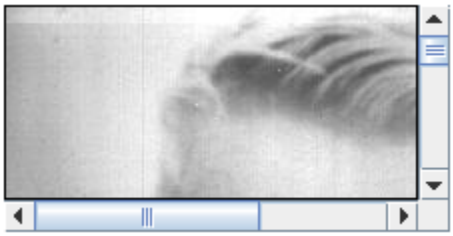
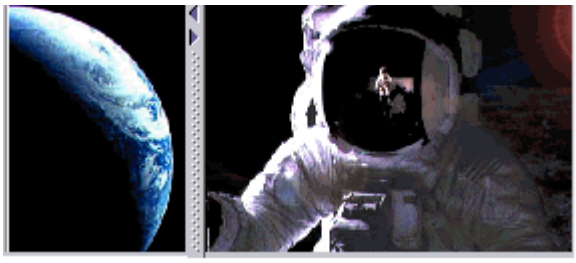
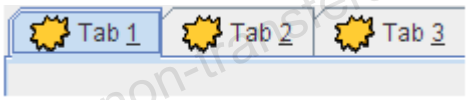
Container	User interface
Panels are containers into which several components can be added. They also provide a surface to draw on. Unlike `JFrame`, they are not top-level containers. Panels should be contained in top-level containers. The `javax.swing.JPanel` class is used to create panels. It extends `JComponent` and not `java.awt.Panel`.	
Scroll panes are very handy when the amount of space is limited. They are used to display large components or images. Scroll panes have two scroll bars, a row header, and a column header. The `javax.swing.JScrollpane` class is used to create scroll panes.	
Toolbars are a group of buttons with icons for easily accessing the frequently used functions. They can be considered as shortcuts to the actions in menus. The `javax.swing.JToolbar` class is used to create toolbars.	

Table C-2 General-purpose Containers (Continued)

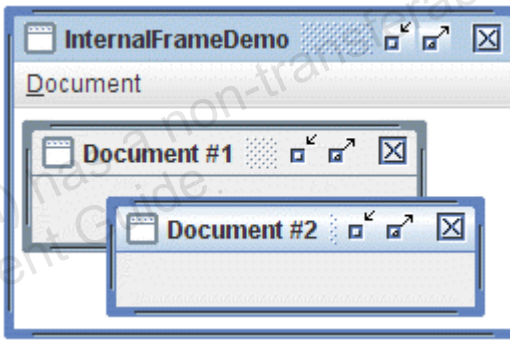
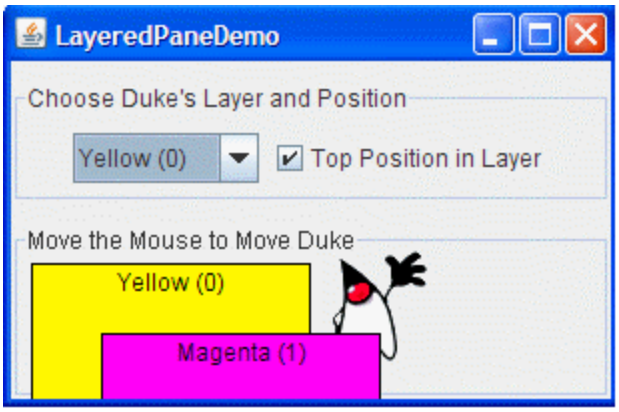
Container	User interface
Split panes display two or more components separated by a divider. Components can be displayed side by side or one over the other. By dragging the divider the amount of space for each component can be adjusted. The `javax.swing.JSplitPane` class is used to create split panes.	
Tabbed panes are also useful when the space is limited. Several tabs share the same space. At any one time only one tab is visible. To be displayed, a tab needs to be selected by the user. The `javax.swing.JTabbedPane` class is used to create tabbed panes.	

Special-Purpose Containers

`JInternalFrame` and `JLayeredPane` are examples of special-purpose containers. They play specific roles in a user interface. Internal frames are designed to work within desktop panes. They are not top-level, such as `JFrame`. The `JLayeredPane` container helps to specify the depth of the component, which is helpful for rendering the GUI when components overlap.

Table C-3 contains a brief description of the special-purpose containers.

Table C-3 Special-purpose Containers

Containers	**User Interface**
Internal frames are non top-level containers. They contain features similar to `JFrames` such as dragging, resizing, iconifying, and maximizing. Internal frames are created using the `javax.swing.JInternalFrame` class, which is added to `JDesktopPane` and is, in turn, added to a `JFrame`. As with regular frames, components can be added to a `JInternalFrame`.	
Layered panes allow the addition of components at required depths. The depth is specified as an integer value. The `javax.swing.JLayeredPane` class is used to create layered panes. For convenience, you can use the standard layers defined by this class: DEFAULT_LAYER, the bottommost layer, PALETTE_LAYER, MODAL_LAYER, POPUP_LAYER, DRAG_LAYER the topmost layer.	

When using container classes, you should keep the following rules in mind:

- GUI components are displayed only when they are in a containment hierarchy. A containment hierarchy is a tree of components that has a top-level container as its root.
- A GUI component object instance can appear only once in a containment tree. If a component in one container is added to another container, the component is moved to the latter container and removed from the former.
- Each top-level container has a content pane that, generally speaking, contains (directly or indirectly) the visible components in that top-level container's GUI.
- You can optionally add a menu bar to a top-level container. The menu bar is, by convention, positioned within the top-level container, but outside the content pane. Some look-and-feels, such as the Mac look-and-feel, give you the option of placing the menu bar in another place more appropriate for the look-and-feel, such as at the top of the screen.

Each of these four containers (including `JApplet`) implements a special interface called `RootPaneContainer`. An in-depth examination of the `RootPaneContainer` is outside the scope of this module.

`JFrame` Container Essentials

The `JFrame` container is the most commonly used top-level Swing container. The `JFrame` container permits you to set one of four reactions for the `Close Window` menu button. These reactions are:

- DO_NOTHING_ON_CLOSE
- HIDE_ON_CLOSE
- DISPOSE_ON_CLOSE
- EXIT_ON_CLOSE

You set the option by invoking the `setDefaultCloseOperation` method on the `JFrame` instance.

Buttons

Regular buttons, check boxes, and radio buttons are all considered to be in the button category. Creating graphical buttons is straight forward if you use an `Icon` object that defines the graphic you want displayed. The `JCheckBox` class provides support for check box buttons. The `JRadioButton` class behaves in such a way that turning *on* a radio button in a radio button group causes all the other radio buttons in that group to be turned *off*. Table C-4 describes each of these components and shows their user interface.

Table C-4 Buttons

Component	User Interface
`JButton` objects can be created with a simple `String` argument to the constructor, in which case they display that text as their label. Clicking on a `JButton` generates an `ActionEvent`. You might want to set the action command property of your button so that the `ActionEvent` carries a particular command string. If you create a `JButton` with text, the label text is used by default as the action command string. However, if you create a graphics-only button or if the default action command string (which is the text label of the button) is not what you need, you can define the action command explicitly using the `setActionCommand` method.	
Check boxes are similar to `JButtons` in that they can be initialized with a simple `String` argument to the constructor, in which case they display that text as their label. But their selection model is different, by convention. A check box has a boolean state value that can be in either *on* (true) or *off* (false). Clicking the check box toggles its state from *on* to *off*, or from *off* to *on*. The `javax.swing.JCheckBox` class is used to create check boxes.	

Table C-4 Buttons (Continued)

Component	User Interface
Individually, a `JRadioButton` simply toggles on and off each time it is selected, just like a `JCheckBox`. To obtain the mutual exclusion behavior of a radio button, add the buttons to a `ButtonGroup` instance. A button group is a manager that ensures that only one button is selected at one time. Use the `ButtonGroup` class to create a button group.	

Text Components

Swing text components can be broadly divided into three categories.

- Text controls – `JTextField`, `JPasswordField` (for user input)
- Plain text areas – `JTextArea` (displays text in plain text, also for multi-line user input
- Styled text areas – `JEditorPane`, `JTextPane` (displays formatted text)

Table C-5 describes each of these components and shows their user interface.

Table C-5 Text Components

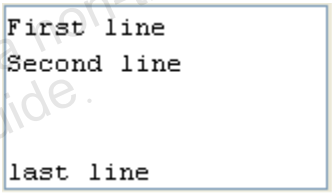
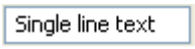
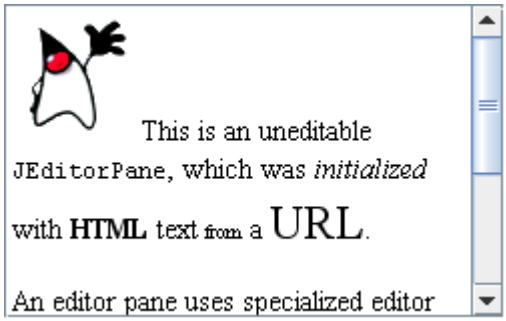
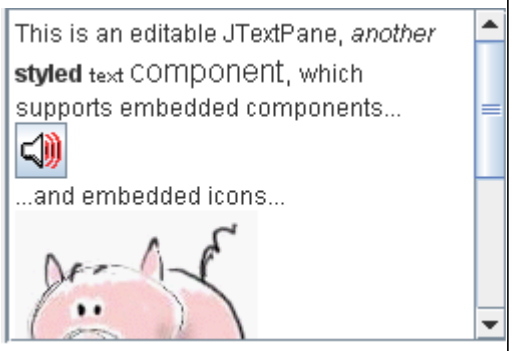
Component	User Interface
Text areas are commonly used to collect more than one line of information from the user. The `javax.swing.JTextArea` class is used to create text areas. When creating a text area, you can specify the number of rows, columns, and initial content. Text areas display plain text only.	
Text fields are also used to collect input from the user. These are similar to text areas but they are restricted to accept only one line of text. The `javax.swing.JTextField` class is used to create text fields.	
Editor panes are styled text components. In addition to plain text, editor panes can display and edit text in RTF and HTML format. Editor panes are commonly used to display help in HTML format. The `javax.swing.JEditorPane` class is used to create editor panes.	

Table C-5 Text Components (Continued)

Component	User Interface
The `javax.swing.JTextPane` class inherits from `javax.swing.JEditorPane`. In addition to providing all the features of the `JEditorPane`, the `JTextPane` class also allows for embedding components.	
`javax.swing.JPasswordField` is a text input field specialized for password entry. For security, a password field displays a character such as an asterisk '*'. A password field's value is stored as an array of characters, instead of a string. Like any other text field object, a password field sends an action event when you press the Enter key.	

Uneditable Information Display Components

Uneditable information display components are used to display more information about the components. These components can be used only as display components. Table C-6 describes some of these components.

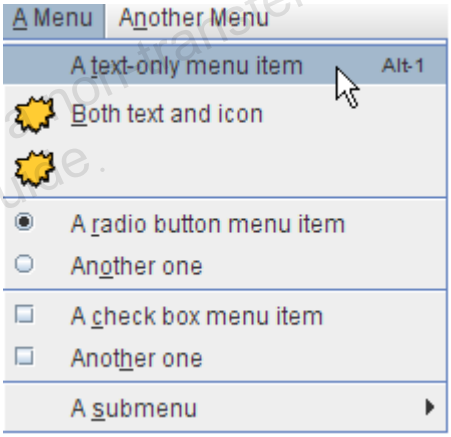
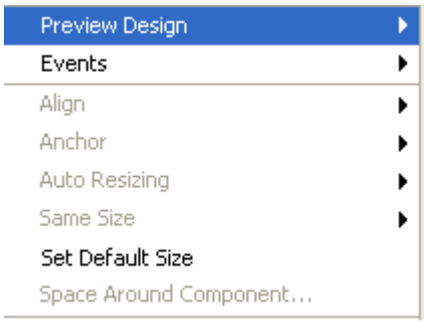
Table C-6 Uneditable Display Components

Component	User Interface
Labels are used to display text on the screen. They are uneditable components. The `javax.swing.JLabel` class is used to create labels. Labels can also be used to display images.	
The `javax.swing.JToolTip` class is helpful to display information about the components. `JComponent` provides a method called the `setToolTipText`, which can be used by all the components to set the string that should be displayed.	
The `javax.swing.JProgressBar` class is very helpful in indicating the progress of any long-running tasks.	

Menus

Menus behave similar to lists except, by convention, a menu typically is displayed either in a menu bar or as a popup menu. A menu bar can contain one or more menus (called pull-down menus) and has an operating system-dependent location, typically, at the top of each window. A popup menu is displayed when the user triggers a platform-specific mouse button or keyboard sequence, for example, pressing the right mouse button, or rolling the mouse cursor over a popup-enabled component. Table C-7 describes each of these components and shows their user interface.

Table C-7 Menu Components

Component	**User interface**
A menu bar contains the names of one or more pull down menus. Clicking these names opens menu items and submenus. The `javax.swing.JMenu`, `javax.swing.JMenuItem` classes are used to create menus. Menu items can be selected by using keyboard mnemonics and accelerators. Menu items could be of type check boxes and radio buttons.	
A popup menu is a menu that is not attached to the menu bar. These menus are some times referred to as context menus. The javax.swing.`JPopupMenu` class is used to create popup menus.	

Formatted Display Components

Formatted display components are among the most complex components found in Swing. Tables, trees, color chooser, and file chooser are a few examples in this category. Table C-8 describes these components and shows their user interface.

Table C-8 Formatted Display Component

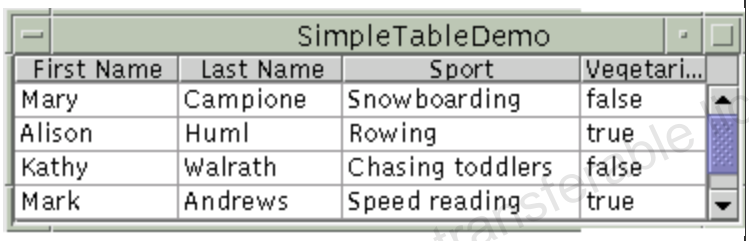
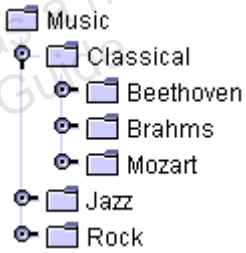
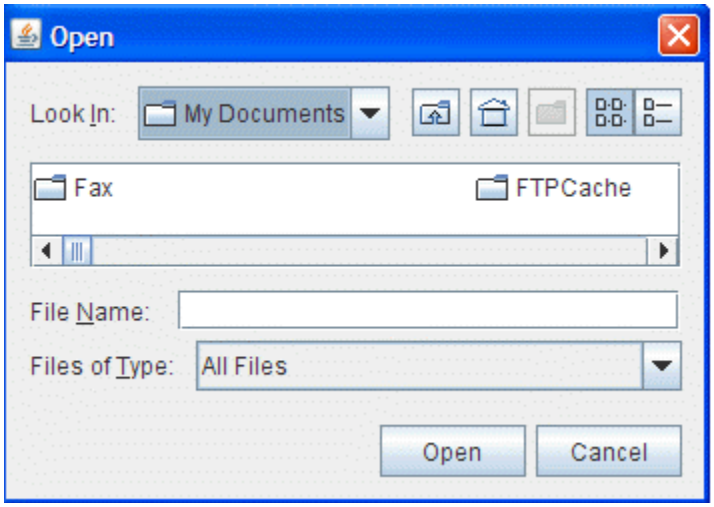
Component	User Interface
Tables are used to display and edit information in the form of a grid. The `javax.swing.JTable` class is used to create tables. `JTable` does not store the data, but only displays the data from the table model.	
`JTree` is used to display hierarchical information. The data for constructing the display is provided through a `TreeModel` instance. The data can also be provided as a collection of elements such as a `Hashtable` or a `Vector`. A JTree does not actually store the data, but rather presents the data in the `TreeModel` instance.	
The `JFileChooser` class allows users to navigate to the file system and choose a file. The file chooser contains filter methods you can use to filter the types of files displayed and methods you can use to customize the view presented by the file chooser.	

Table C-8 Formatted Display Component (Continued)

Component	User Interface
`JColorChooser` allows the user to manipulate and select a color. A color can be selected in three different ways. Swatches, Hue, Saturation, and Brightness (HSB) or Red, Green, and Blue (RGB). The `JColorChooser` class provides several constructors for creating the color-chooser pane. The default constructor creates a pane with initial color as white. Another constructor takes the initial color as a parameter. You can also specify the color selection model in the constructor.	

Other Basic Controls

This section describes other Swing components, such as combo boxes, lists, sliders, and spinners that are often used in GUIs. A `JComboBox` lets the user choose one of several choices. The `JComboBox` class has a convenient constructor that takes an array of objects to use as the initial choices. You can add and remove choices using the `addItem` and `removeItem` methods, respectively. A `JList` presents items in one or more columns. You choose one or more items from the display by clicking or navigating with keyboard commands. With `JSlider`, you use a mouse click and drag to enter a numeric value. A spinner is a possible alternative to a slider when screen space is limited.

Table C-9 describes each of these components and shows their user interface.

Table C-9 Other Basic Controls

Component	User Interface
`JComboBox` has two forms: Editable and UnEditable. The default form, uneditable, shows a button and a drop-down list of values. The editable form presents a text edit field with a selection button. You can type a value in the text field or use the button to display a drop-down list of choices.	
Lists can feature scroll bars if the number of items is too large to display in the allotted screen area. Lists can also be made resizable. The `JList` class is simple to use in simple circumstances, and can create its own `ListModel` if needed. To do this, you put the data items into either a `Vector` or an array of `Objects` and invoke the `JList` constructor using the data as an argument.	

Table C-9 Other Basic Controls (Continued)

Component	User Interface
Spinners also allow you to type in a value. `JSpinner` has three subcomponents: an up arrow, a down arrow, and an editor. The editor can be any `JComponent`, but, by default, it is implemented as a panel with formatted text field	
Slider values have a finite range, that is, a minimum and maximum value. If the ability to specify precise numbers is important, a slider can be coupled with a formatted text field.