Chapter 16

User Interface Design and Programming

Topics

- User interface design guidelines
- User interface components
  - containers
  - menus
  - controls
- User interface event handling
Clients

Programmable client
- program resides and executes on the client and it has access to client storage resources
- called also a thick client or rich client

Browser client
- needs a server to download the requested data and to obtain instructions for rendering the data in a web-based UI
- except for simple validations of user input, a browser client does not have processing capability on its own
- the data is presented as a web page formatted with HyperText Markup Language (HTML)
- called also a thin client, Web client, or HTML client

An application can be deployed:
- locally with a programmable client
- on a Web server with a browser client and accessed by servlets and Java Server Pages (JSP)
- on an application server, such as an Enterprise JavaBeans (EJB) server, with a programmable or browser client

UI design guidelines

user in control
- no mothering

interface consistency
- no creative urges
- interface aesthetics

interface forgiveness
- resilient interface
- easy to explore

interface adaptability
- features can be changed depending on user’s familiarity
- locale-specific information
- people with disabilities
UI components

- **Java™ Foundation Classes (JFC)**
- **Swing component kit**
  - enables delivery of applications with pluggable look-and-feel
  - lightweight (peerless) components
  - some components are heavyweight
- **Crude classification:**
  - containers
    - e.g. JInternalFrame, JTabbedPane
  - menus
    - e.g. JPopupMenu, JRadioButtonMenuItem
  - controls
    - e.g. JRadioButton, JScrollBar

Containers

- rectangular areas on a GUI desktop that contain other components, including other containers, menus, and controls
- called windows, dialogs, panes, panels, and similar terms
- determine the prime look-and-feel of an application
- **Heavyweight:**
  - JWindow, JFrame, JDialog, and JApplet
- **Lightweight:**
  - JInternalFrame, JDesktopPane, JOptionPane, JPanel, JTabbedPane, JScrollPane, JSplitPane, JTextPane, and JTable
Primary and secondary windows

```java
public class Frame1_AboutBoxPanel1 extends JPanel

private JToolBar toolBar = new JToolBar();
private JMenuBar menuBar = new JMenuBar();
private JButton buttonHelp = new JButton();
private JMenuItem menuHelpAbout = new JMenuItem();
```

Primary window in Swing

- A top-level container of a program is an instance of a **heavyweight** container
- **Lightweight** container classes need heavyweight components for screen painting and event handling
- JWindow is a top-level container with no “decorations” - no borders, title, menu bar or scrollbar
  - a subclass of JWindow, such as JFrame or JPanel, is normally used to implement a popup window
- JFrame object has “decorations”
- To add lightweight components to a heavyweight container object, the container must use a special method `getContentPane()`

```java
aFrame.getContentPane().add(toolBar, BorderLayout.NORTH);
```
Secondary window in Swing

- **JDialog** is a heavyweight component for creating dialog windows (dialog boxes)
- **JOptionPane** provides a number of standard dialog windows
  - All standard dialogs are **modal**
  - An application can request a dialog window by invoking appropriate method in **JOptionPane**

```java
void helpAbout_ActionPerformed(ActionEvent e) {
    JOptionPane.showMessageDialog(this,
                                   new Framel_AboutBoxPanell(),"About",
                                   JOptionPane.PLAIN_MESSAGE);
}
```

Tabbed pane

- A **JTabbedPane** object results in a window with many "tabbed pages/panels"
- Useful in applet programming, which discourages the use of pop-up dialogs
- Adding a JPanel object to the JTabbedPane object creates each tabbed pane
Table

- JTable delivers a table of rows and columns
  - Scrolling provided by a JScrollPane object
  - JScrollPane uses a JViewport object to provide a "viewport" onto a data source

```java
private JTable tableMovie1View1 = new JTable();
private JScrollPane scroller = new JScrollPane();
scroller.getViewport().add(tableMovie1View1, null);
```

<table>
<thead>
<tr>
<th>MovieCode</th>
<th>MovieTitle</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Interview with the Vampire</td>
<td>Neil Jordan</td>
</tr>
<tr>
<td>11</td>
<td>The Birdcage</td>
<td>Mike Nichols</td>
</tr>
<tr>
<td>12</td>
<td>The Pianist</td>
<td>Roman Polanski</td>
</tr>
<tr>
<td>13</td>
<td>Solaris</td>
<td>Steven Soderbergh</td>
</tr>
<tr>
<td>14</td>
<td>Quiet American</td>
<td>Philip Noyce</td>
</tr>
<tr>
<td>15</td>
<td>Frida</td>
<td>Julie Taymor</td>
</tr>
</tbody>
</table>

Layout management

- Swing applies a "layout manager" to place components within a container
- A method `setLayout()` allows choosing desired layout manager, e.g.
  ```java
aFrame.getContentPane().setLayout(new GridLayout(6,5));
```

- Swing layouts include BorderLayout, FlowLayout, GridLayout, BoxLayout, and GridBagLayout.
  - BorderLayout is a default layout scheme for most containers, except JPanel (which uses FlowLayout by default).

- Manual positioning of graphical components within containers can be programmed with a layout manager set to null:
  ```java
  aContainer.getContentPane().setLayout(null);
  ```
Layout management

null

FlowLayout

BorderLayout

FlowLayout

GridLayout

null

BorderLayout

GridLayout

Layouts on tabbed page

BorderLayout

FlowLayout

null

BorderLayout

FlowLayout
**Layering management**

- **JLayeredPane**
  - provides methods to “layer” components
  - most components go to a standard (default) layer (components in a **default layer** overlap properly based on user’s selections of these components)
  - allows declaring special predefined characteristics of layers
    - **palette layer** that floats above the default layer (e.g. floating toolbar)
    - **modal layer** that appears on top of all other active windows, toolbars and palettes in the application and does not allow switching to these other windows unless it itself is dismissed
    - **pop-up layer** that displays temporarily in its own layer above other layers (e.g. a combo box, tooltip)
    - **drag layer** that makes the component visible when it is dragged before it is dropped on a destination layer

---

**Menus**

- `JMenuBar`
- `JMenu`
- `MenuElement`
- `JMenuItem`
- `JPopupMenu`
- `JRadioButtonMenuItem`
- `AbstractButton`
- `JCheckBoxMenuItem`
Menus

- JMenu
- JMenuItem
- Cascading popup menu
- Right-click popup menu
- JCheckBoxMenuItem
- Accelerator key

Toolbars

- Toolbar
- Floatable toolbar
Controls

Buttons and other controls
Swing event handling

Interfaces from Swing library:
- ActionListener
- WindowListener
- MouseListener
- etc.

Can be a distinct class, but frequently an inner class or an anonymous inner class within a publisher class.

Swing Listener

Publisher1

Subscriber1

<<event source>>

Event objects from Swing library

<<event listener>>

Publisher2

Subscriber2

<<event source>>

ActionEvent

<<event listener>>

Subscriber3

<<event source>>

WindowEvent

<<event listener>>

cancelButton.addActionListener(instanceofSubscriber3);

public void actionPerformed(java.awt.event.ActionEvent evt) {
    cancelButtonActionPerformed(evt);
}

Summary

- User Interface (UI) design must consider two main kinds of clients: a programmable client and a browser client.
- The main UI design guidelines are: (1) user in control, (2) interface consistency, (3) interface forgiveness, and (4) interface adaptability.
- The Swing components can be grouped into containers, menus, and controls.
  - Containers determine the prime look-and-feel of an application. Swing applies a “layout manager” to place components within a container.
  - Windows and some other components can hold menus. Check boxes and radio buttons can be considered as kinds of menu items. Toolbars can also classify as menus.
  - Controls represent the UI event model. They divide into actions buttons and other controls. One of the most versatile and useful controls is a tree view of containers and items.
- The Swing event model is derived from the MVC framework. The UI event handling should conform to the Observer pattern.