Outline

- Event handling example
- Model-view-controller
- JTable
- Homework #3
Event handling example
Model-view-controller

- Controller
  - manipulate-able interface modality
- View
  - observable interface display
- Model
  - underlying data structure
### JTable

#### Image Browser

<table>
<thead>
<tr>
<th>File name</th>
<th>Size (K)</th>
<th>Height</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>kathyCosmo.gif</td>
<td>29</td>
<td>200</td>
<td>288</td>
</tr>
<tr>
<td>lainesTongue.gif</td>
<td>63</td>
<td>237</td>
<td>369</td>
</tr>
<tr>
<td>smallbabies.jpg</td>
<td>34</td>
<td>252</td>
<td>369</td>
</tr>
<tr>
<td>sophie.gif</td>
<td>140</td>
<td>400</td>
<td>600</td>
</tr>
<tr>
<td>stickerface.gif</td>
<td>26</td>
<td>238</td>
<td>230</td>
</tr>
<tr>
<td>youngdad.jpg</td>
<td>19</td>
<td>480</td>
<td>320</td>
</tr>
</tbody>
</table>
MVC in JTable
More Typical
TableModel

// Mandatory
int getColumnCount()
int getRowCount()
Object getValueAt(int rowIndex, int columnIndex)

// For display
String getColumnName(int columnIndex)

// For editing
boolean isCellEditable(int rowIndex, int columnIndex)
void setValueAt(Object aValue, int rowIndex, int columnIndex)

// For rendering
Class getColumnClass(int columnIndex)

// For event handling
void addTableModelListener(TableModelListener l)
void removeTableModelListener(TableModelListener l)
DefaultTableModel

- Vector of Vectors
- To create
  - DefaultTableModel(Object[][] data, Object[] columnNames)
  - DefaultTableModel(Vector data, Vector columnNames)
- You can use the default table model implicitly
  - new JTable(Object[] [] data, Object[] columnNames)
  - new JTable(Vector data, Vector columnNames)
AbstractTableModel

- Supplies default methods for most of the methods
- Specializers need implement only
  - int getColumnCount()
  - int getRowCount()
  - Object getValueAt(int rowIndex, int columnIndex)
Why a separate class?

- **Efficiency 1**
  - no need to convert table to a different form

- **Efficiency 2**
  - only generate those items that the user wants to see

- **Modularity**
  - Isolate the form of data display from the data itself
Handling Change

- The JTable needs to know when the model has changed
  - TableModelListener
- If you have your own setValueAt method
  - you must call fireTableCellUpdated (row, col)
  - the JTable is a listener
Other Models

- **TableColumnModel**
  - Separate from the table data
  - Columns have their own display characteristics
    - visible / hidden
    - width
    - ordering

- **ListSelectionModel**
  - Selection is a set of intervals
  - Issue of “what is selected” can be complex
    - TreeSelectionModel
Editing and Rendering

- Return a class with getColumnClass
- Some built-in
  - Boolean = check-box
  - ImageIcon = centered label containing the image
- Otherwise
  - Register an editor and/or renderer for that class
  - `table.setCellEditor (Class, CellEditor)`
  - `table.setDefaultRenderer (Class, Renderer)`
Example

- FlightTableModel
JTree

- Root
  - Letters
  - Numbers
    - One
    - Two
    - Three
  - The Dictionary
TreeModel

// essential
Object getRoot()
Object getChild(Object parent, int index)
int getChildCount(Object parent)
int getIndexOfChild(Object parent, Object child)
boolean isLeaf(Object node)

// editability
void valueForPathChanged(TreePath path, Object newValue)

// event handling
void addTreeModelListener(TreeModelListener l)
void removeTreeModelListener(TreeModelListener l)
TreeNode

// move down
Enumeration children()
TreeNode getChildAt(int childIndex)
int getIndex(TreeNode node)
int getChildCount()

// move up
TreeNode getParent()

// status
boolean getAllowsChildren()
boolean isLeaf()
Events

- Tree expand/collapse
  - TreeExpansionListener
- Node selection
  - TreeSelectionListener
- Node/tree modification
  - TreeModelListener
Homework #3

- FFGUIMode
- Differences
  - package structure
  - dynamic command loading
  - automatic path detection
  - graphical interface
  - in-memory database
  - must use JDK 1.4
db package

- Class related to data storage and retrieval
- FrequentFlyerDB
  - main class for interacting with the stored data
  - also MemberDB and FlightDB
- Member
  - holds individual data
  - has miles value
- Flight
  - holds flight records
- MultiMap
  - a utility class for a Map with multiple values for one key
- DBUtilities
  - home of various utility functions
core package

- CommandMap
  - maps names to commands
  - loads commands at start up

- Command
  - SimpleCommand
    - ExitCommand
    - HelpCommand
  - MDatabaseCommand
    - “M” indicates “memory” database
cmd package

- Adding/deleting commands
  - MAddMemberCommand
  - MAddFlightCommand
  - MDeleteFlightCommand

- Reporting commands
  - MShowMembersCommand
  - MShowFlightsCommand

- Finding Commands
  - MFindByByNameCommand
  - MFindByLevelCommand
  - MFindFlightsByMemberCommand

- UnknownCommand
ui package

- FFTextMode
- FFGUIMode
  - FFFrame
    - menus
  - CommandListener
    - external listener class for commands
- PromptDialog
  - default dialog for getting parameters for commands
- ResultDialog
  - default dialog for displaying results
ui package continued

- FlightDisplayDialog
  - JTable
  - FlightTableModel
- EnterLevelDialog
  - Combo box for benefit level
- EnterIdDialog
Assignment

- Input dialog for “add member”
  - use appropriate components

- Output dialog with a table of members
  - to be used by “show members”, “find by name”, “find by level”
  - must use a table model

- Help dialog
  - But the set of commands is not fixed!
Command Class
New I/O Methods

- **getParameters**
  - detects mode
  - calls `getParametersText` or `getParametersGUI`
  - default implementations provided
  - subclasses must override `getParametersGUI` to display their own dialogs

- **displayResults**
  - detects mode
  - calls `displayResultsText` or `displayResultsGUI`
  - default / must override
Examine

- FFFrame
- CommandListener
- MShowFlightsCommand
- MDeleteFlightCommand
- EnterIdDialog
Advice

- Start early!
- Draw your dialogs before implementing
- Use existing classes as models