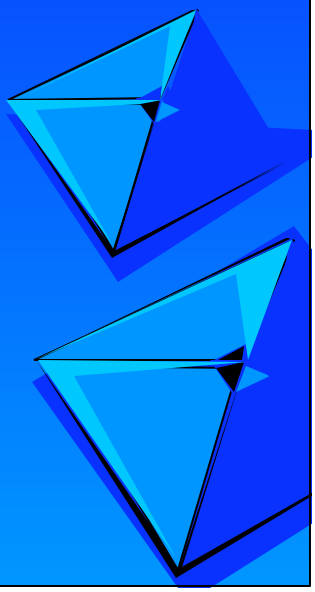
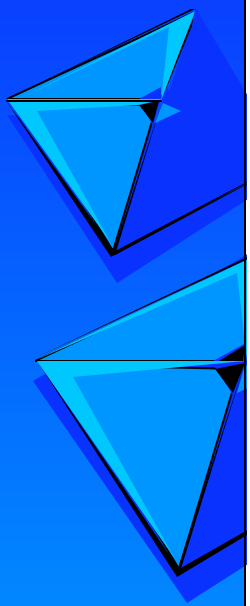
Java Swing Programming

Kyle Brown
IBM WebSphere Services



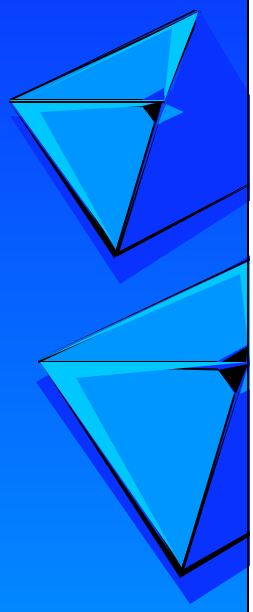
Overview

- Swing's Architecture & background
- Basic Swing Programming
- Swing lists
- Swing tables
- Swing trees



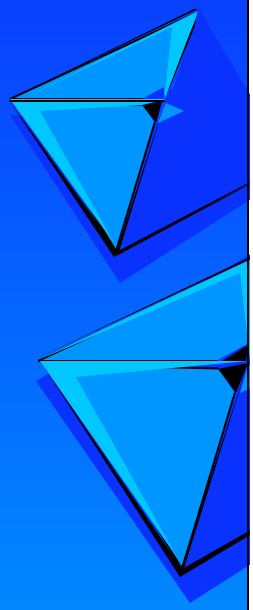
What's Swing?

- Swing is Sun's alternate Windowing Framework
- Part of the JFC (Java Foundation Classes)
  - Standard part of Java 2
- A full replacement/alternative to AWT
- Found in packages starting with javax.swing



Key Features of Swing

- Not based on windowing system
  - all Swing components are lightweight
- Pluggable Look and Feel (PLAF)
  - Richer component set than AWT
  - Doesn't take the "least common denominator" approach
  - includes components not found in *any* windowing system
  - adds functionality to components that already have counterparts in the AWT

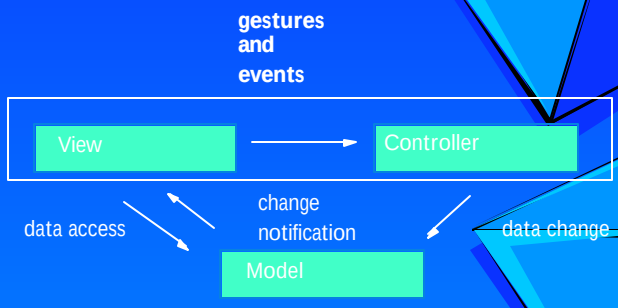


Using Swing

- Swing should not be combined with AWT
  - Each Window should be fully Swing or fully AWT
- Since Swing matches the AWT's functionality, that's not hard
  - Most Swing components are upwards-compatible with their AWT counterparts
  - easy to change code from AWT to Swing

Swing Architecture

- Swing follows the MVC paradigm for building user interfaces
  - Each UI Component has a model
- We will often customize certain Swing models

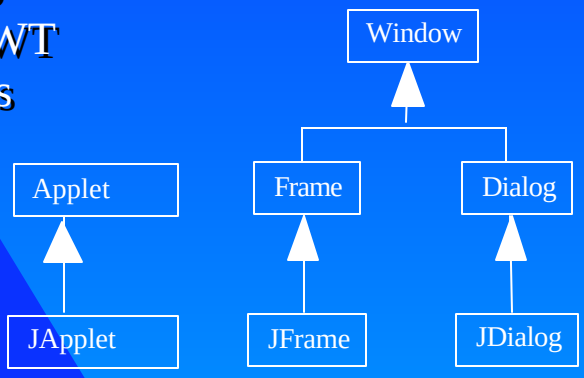


Basic Swing Programming

- As noted previously, Swing adds an entirely new set of components to Java
  - Upward-compatible with their AWT counterparts
  - We won't deeply cover those similar to their AWT counterparts
- We will start at the bottom
  - JFrame
  - Japplet

Top-level hierarchy

- Each top-level class adds new behavior to existing AWT components



Comparing Swing & AWT

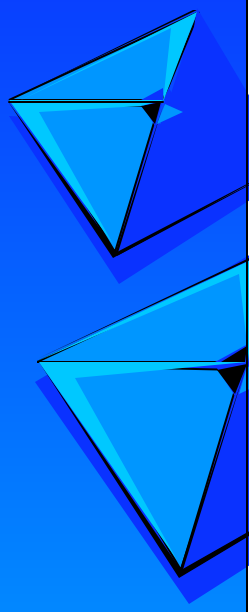
```
import java.awt.*;

public class ExampleAWTFrame extends java.awt.Frame {

public ExampleAWTFrame ( ) {
      super();
      Panel aPanel = new Panel();
      aPanel.setLayout(new BorderLayout());
      add(aPanel);
      Label hello = new Label("Hello World");
      aPanel.add(hello, BorderLayout.NORTH);
}

public static void main(String args[]) {
      ExampleAWTFrame aFrame = new ExampleAWTFrame();
      aFrame.setVisible(true);
}}
```

Comparing Swing & AWT

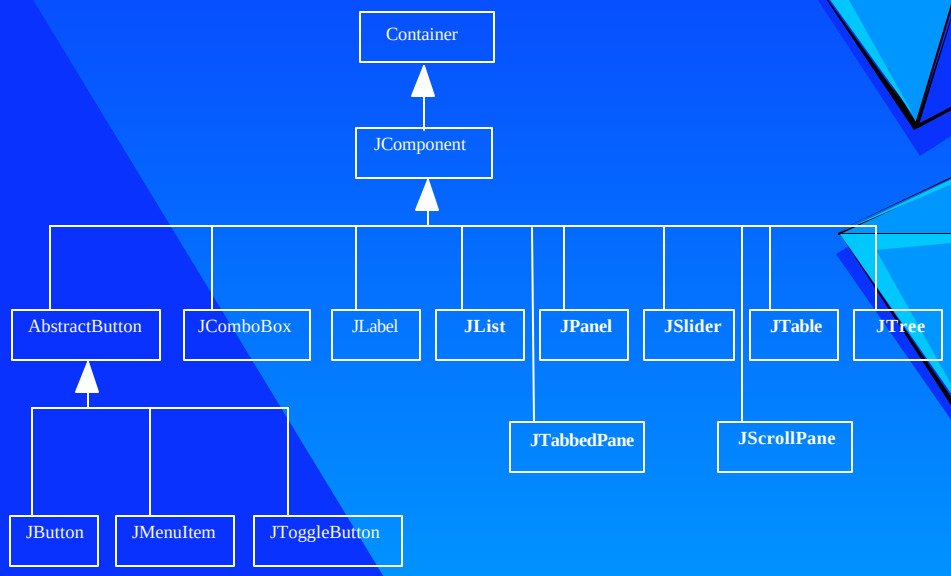
```
import java.awt.*;
import com.sun.java.swing.*;

public class ExampleSwingJFrame extends com.sun.java.swing.JFrame {

public ExampleSwingJFrame() {
      super();
      JPanel aPanel = new JPanel();
      aPanel.setLayout(new BorderLayout());
      getContentPane().add(aPanel);
      JLabel hello = new JLabel("Hello World");
      aPanel.add(hello, BorderLayout.NORTH);
}

public static void main(String args[]) {
      ExampleSwingJFrame aFrame = new ExampleSwingJFrame();
      aFrame.setVisible(true);
}}
```

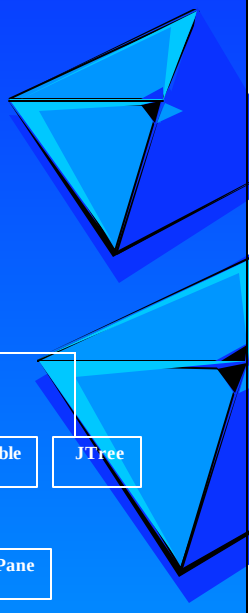
JFrames

- A JFrame acts like an AWT Frame
  - Except it handles Swing component nesting
- A JFrame is really two “panes”
  - A “LayeredPane” that has an optional menu bar and a content pane
  - A “GlassPane” that sits transparently in front of the LayeredPane



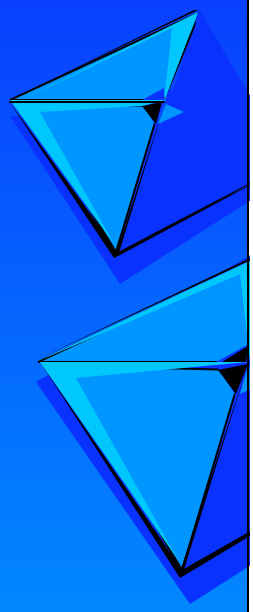
Partial Swing Hierarchy





Borders

- Any JComponent can have a Border placed on it
  - Usually only used on JPanels
- Specify the border with
  - aJComponent.setBorder(Border aBorder)
- Common Borders include BevelBorder, TitledBorder
  - Normally built with a BorderFactory

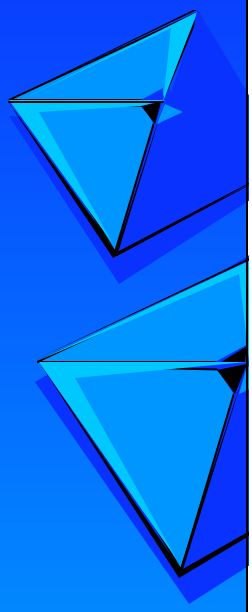


JTitledBorder Example

```
public ExampleWithGroupBox ( ) {
  super();
  JPanel aPanel = new JPanel();
  aPanel.setLayout(new BorderLayout());
  TitledBorder border =
    BorderFactory.createTitledBorder(BorderFactory.createBevelBorder(2), "Group");
  aPanel.setBorder(border);
  getContentPane().add(aPanel);
  JLabel hello = new JLabel("Hello World");
  aPanel.add(hello, BorderLayout.CENTER);
}
```

JTabbedPane

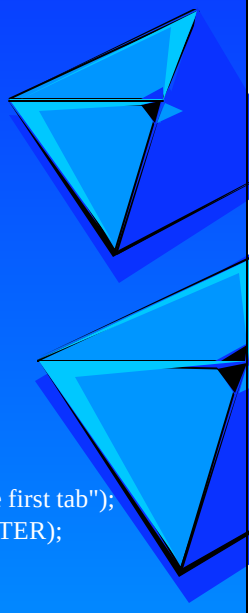
- A JTabbedPane represents a “notebook”
  - Any JComponent can be a page in a JTabbedPane
  - JPanels are usually used
- Tabs can be added, inserted at runtime, deleted and selected



Creating Tabs

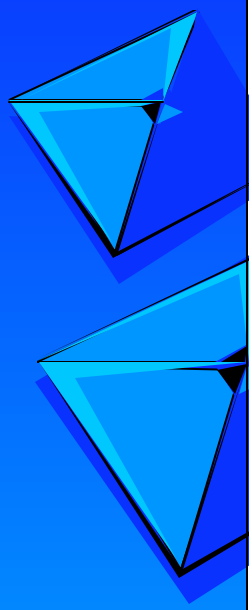
```
public ExampleWithTabbedPane ( ) {
    super();
    JTabbedPane tabs = new JTabbedPane();
    tabs.addTab("Page1", buildTabOne());
    tabs.addTab("Page2", buildTabTwo());
    tabs.addTab("Page3", buildTabThree());
    getContentPane().add(tabs);
}
```

```
public JPanel buildTabOne() {
    JPanel aPanel = new JPanel();
    JLabel first = new JLabel("This is the first tab");
    aPanel.add(first, BorderLayout.CENTER);
    return aPanel;
}
```



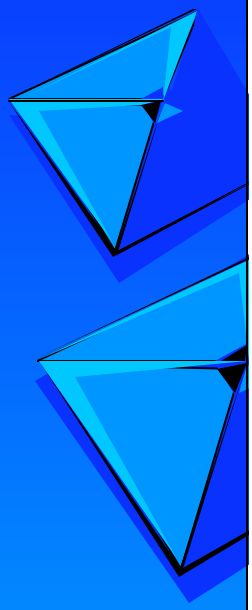
JTabbedPane Details

- Tabs are indexed from 0
  - JTabbedPane.getTabCount();
- Can remove tabs with
  - JTabbedPane.removeTabAt(index);
- Can select a tab with
  - JTabbedPane.setSelectedIndex(index)



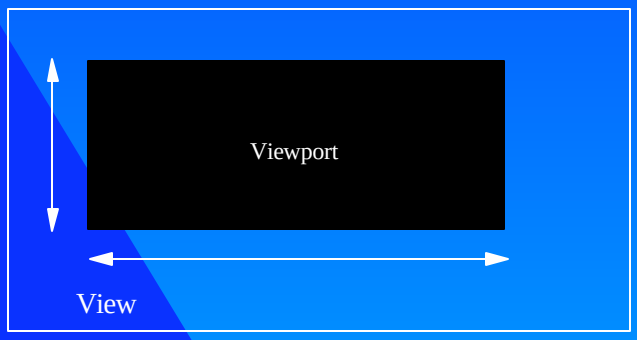
JTabbedPane Events

- The JTabbedPane supports the ChangedEvent notification
- Clients must implement the ChangeListener interface
  - void stateChanged(ChangeEvent e)
- No state information is passed in
  - query the source for the new state



JScrollPane

- A JScrollPane manages scrolling over a larger view
  - It manages a viewport on the view

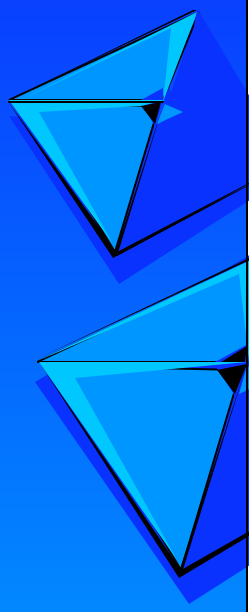


JScrollPane example

```
/*
* Add a Scroll pane to the Jframe.  Put a JLabel having
* the numbers 0 - 49 into the scrollPane's viewport
*
*/
public ExampleScrolling ( ) {
      super();
      JScrollPane scroller = new JScrollPane();
      getContentPane().add(scroller);
      StringBuffer bigBuffer = new StringBuffer();
      for (int i=0; i<50; i++) {
            bigBuffer.append(Integer.toString(i));
            bigBuffer.append(' ');
      }
       JLabel longLabel = new JLabel(bigBuffer.toString());
      scroller.getViewport().add(longLabel);
}
```

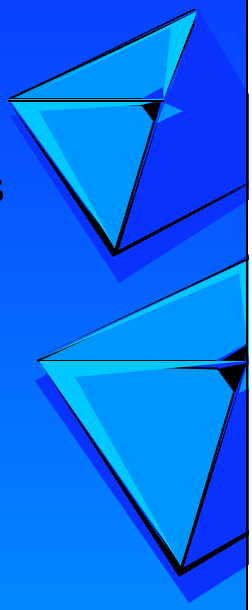
Scrollable Interface

- Components that will be scrolled by a JScrollPane should implement the Interface Scrollable
  - Most Swing components implement this already
- This Interface defines methods to
  - return the preferred size of the viewport
  - get the increment in pixels to scroll by unit and block



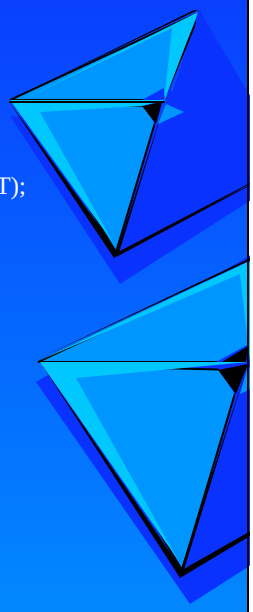
JSplitPane

- A JSplitPane is a “splitter” pane that allows the user to resize two components dynamically
  - Can split horizontally or vertically
- Use the constant JSplitPane.VERTICAL_SPLIT or JSplitPane.HORIZONTAL_SPLIT in the constructor



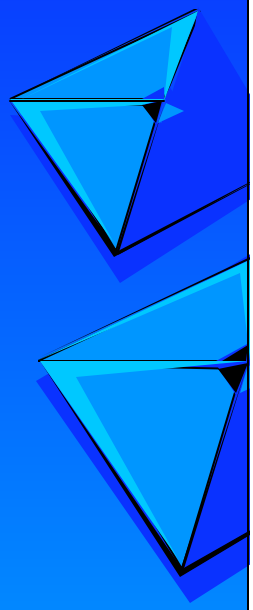
JSplitPane Example

```
public ExampleWithSplitPane ( ) {
      JSplitPane splitter = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
      splitter.setLeftComponent(new JTextArea());
      splitter.setRightComponent(new JTextArea());
      getContentPane().add(splitter);
}


public static void main(String args[]) {
    ExampleWithSplitPane example = new ExampleWithSplitPane();
    example.pack();
    example.setVisible(true);
}
```



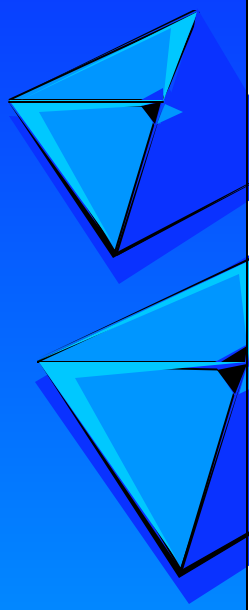
JProgressBar

- A JProgressBar is a standard Windows-like progress indicator
  - LED-like display like a Stereo system
- It is usually used in its own Frame or dialog
  - Often combined with one or more labels



JSlider

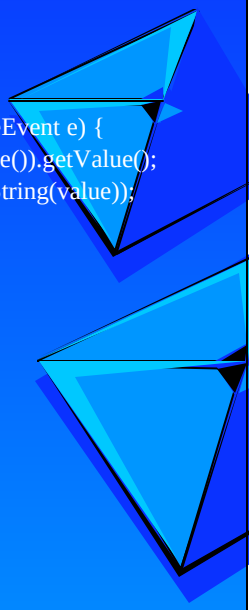
- A JSlider is a volume-control slider component
  - Familiar in many Windows applications
- Jsliders have a variety of attributes
  - Major and Minor Ticks
  - Labels
  - Minimum & Maximum values
- Clients watch for the ChangeEvent notification
  - Similar to JTabbedPane



JSlider Example

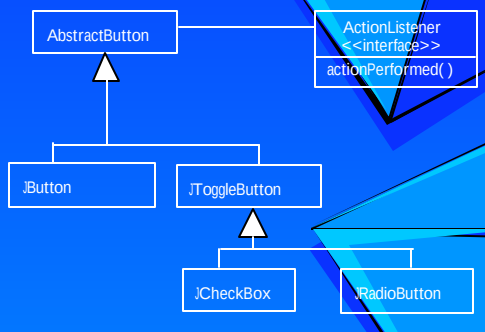
```
public ExampleWithSlider ( ) {
  JPanel aPanel = new JPanel();
  aPanel.setLayout(new BorderLayout());
  getContentPane().add(aPanel);

  counterLabel = new JLabel("0");
  aPanel.add(counterLabel, BorderLayout.SOUTH);

  JSlider slider = new
    JSlider(SwingConstants.HORIZONTAL, 0, 100, 0);
  slider.setMajorTickSpacing(20);
  slider.setMinorTickSpacing(10);
  slider.setPaintTicks(true);
  slider.setPaintLabels(true);
  slider.addChangeListener(this);
  aPanel.add(slider, BorderLayout.NORTH);
}
```

```
public void stateChanged(ChangeEvent e) {
  int value  = ((JSlider) e.getSource()).getValue();
  counterLabel.setText(Integer.toString(value));
}
```



AbstractButton Hierarchy

- Swing has a number of button components
  - push buttons, radio buttons, check boxes
- Usually each will use an ActionListener to hook into UI actions

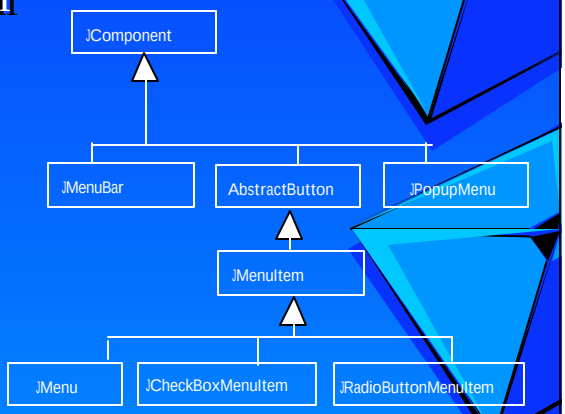


JButton Example

```
public ExampleJButton() {
    super();
    JPanel aPanel = new JPanel();
    aPanel.setLayout(new BorderLayout());
    getContentPane().add(aPanel);

    JButton push = new JButton("Push Me");
    push.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent evt) {
            buttonPushed();
        }
    });

    aPanel.add(push, BorderLayout.NORTH);

    this.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
}

public void buttonPushed() {
    System.out.println("The button was pushed");
}
```

JMenu Hierarchy

- Swing Menus are similar to Buttons
  - JMenuItems also have ActionListeners
  - This is because they are AbstractButtons

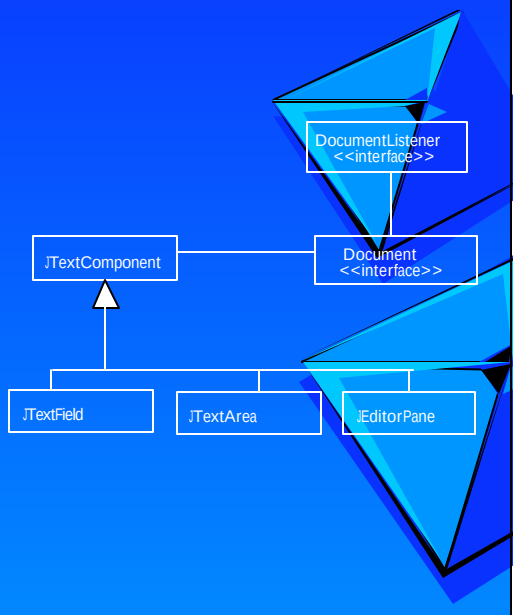


JMenu Example

```
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);  // method of JFrame
JMenu testMenu = new JMenu("Test");
JMenuItem menuItem = new JMenuItem("Select Me");
menuItem.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
            menuSelected();
      }
});

testMenu.add(menuItem);
menuBar.add(testMenu);
```

Text Handling

- Swing has text handling capabilities similar to AWT
  - JTextField, JPasswordField, JTextArea
- However, it also has sophisticated support for viewing HTML and RTF
  - JEditorPane

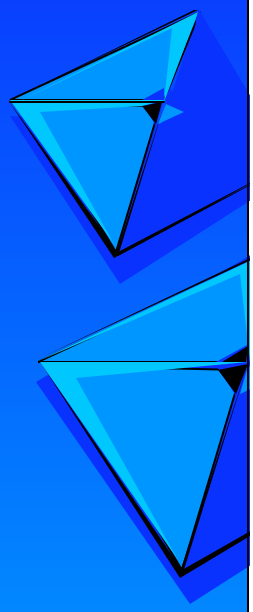


JList

- JList is the primary Swing list class
- It's like the AWT List widget except:
  - The widget doesn't support adding & removing elements directly
  - Adds customized data models
  - Adds custom element rendering

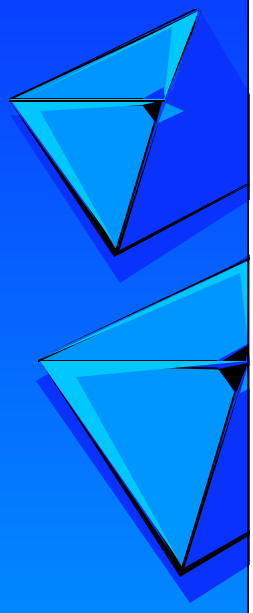
Simple JList Example

```
/*
* You must place a Jlist inside a JScrollPane to get scroll bars!
*/

public ExampleSimpleList ( ) {
   String[] strings = {"Bob", "Carol", "Ted", "Alice", "Jane", "Fred", "Sue"};
   JScrollPane scroller = new JScrollPane();
   JList aList = new JList(strings);
   scroller.getViewport().add(aList);
   getContentPane().add(scroller);
}
```



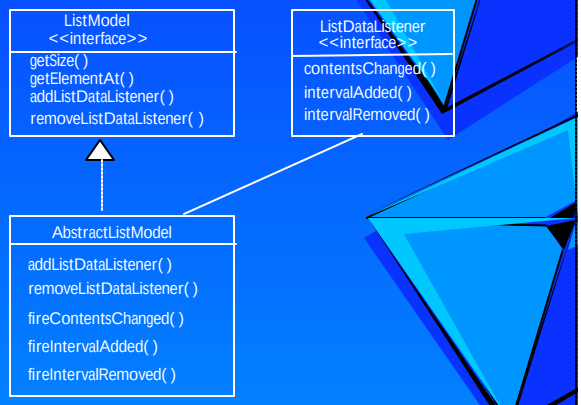
JList Constructors

- Jlist has four constructors
  - JList()
  - JList(Object[])
  - JList(Vector)
  - JList(ListModel)
- Use the array and Vector versions only for simple, static lists
  - For more complex lists, use a ListModel



ListModel

- A ListModel represents a list of elements to a JList
- Subclass AbstractListModel to override getSize() and getElementAt()

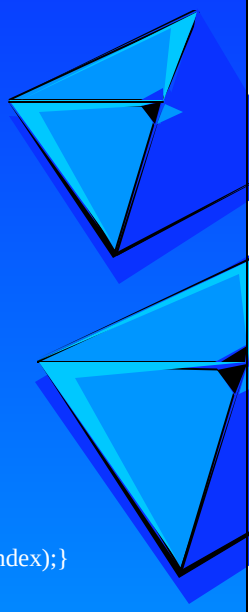


ListModels

- There would be several reasons you might make a ListModel
  - Loading database information as it is requested
  - “Synthetic” lists of calculated items
- As our example we’ll store information in a hashtable, and display the keys as they were added to the hashtable

Example ListModel

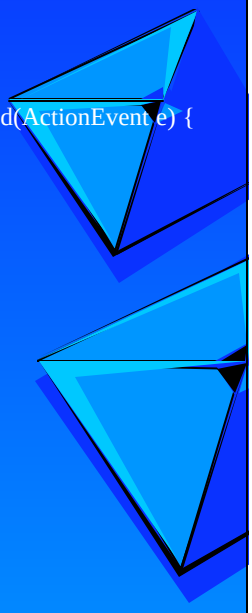
```
public class CustomListModel extends AbstractListModel {

Hashtable data = new Hashtable();
Vector orderedKeys = new Vector();

public void put(Object key, Object value) {
      data.put(key, value);
      if (!orderedKeys.contains(key))
            orderedKeys.addElement(key);
      fireContentsChanged(this, -1, -1);
}

public Object get(Object key) { return data.get(key);}

public Object getElementAt(int index) { return orderedKeys.elementAt(index);}

public int getSize() { return orderedKeys.size();}

}
```



ListModel Example

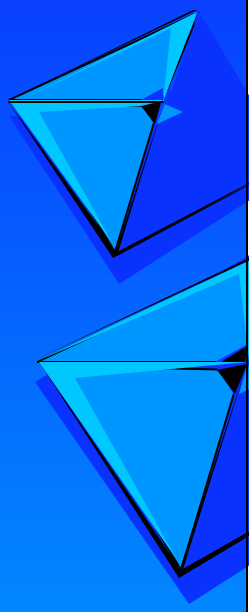
```
public ExampleCustomListModelList ( ) {
      JPanel outerPanel = new JPanel();
      outerPanel.setLayout(new BorderLayout());
      JScrollPane scroller = new JScrollPane();

      JList aList = new JList(buildCustomListModel());
      scroller.getViewport().add(aList);

      JButton button = new JButton("Add");
      button.addActionListener(this);
      outerPanel.add(scroller, BorderLayout.NORTH);
      outerPanel.add(button, BorderLayout.SOUTH);
      getContentPane().add(outerPanel);
}
```

```
public void actionPerformed(ActionEvent e) {
      model.put("As", "If");
}
```



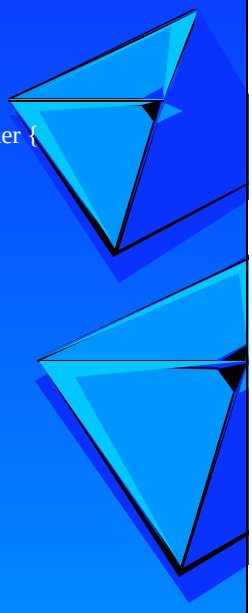
JList Events

- JLists support the ListSelection event notification
- Clients implement the ListSelectionListener interface
  - void valueChanged(ListSelectionEvent e)
- A ListSelectionEvent knows several things
  - first selected index
  - last selected index
  - getValueIsAdjusting()



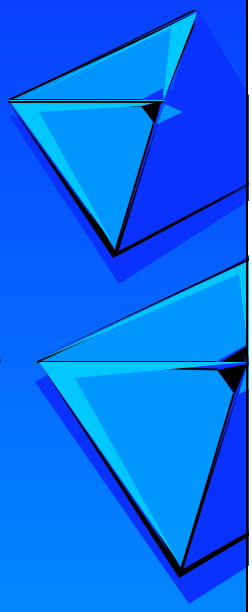
JList Event Example

```
public class ExampleListListening extends JFrame implements ListSelectionListener {
…
ExampleListListening() {
…
JList aList = new JList(someModel);
aList.addListSelectionListener(this);
…

public void valueChanged(ListSelectionEvent e) {
      if (!event.getValueIsAdjusting()) {
            String value = event.getSource().getSelectedValue();
            System.out.println("Selection is " + value);
      }
}

}
```



List Cell Rendering

- JList gives you the option to display not just Strings, but graphical icons as well
- You need to create a new cell renderer
  - implement the ListCellRenderer interface
  - set the custom text and icon (image) in this class

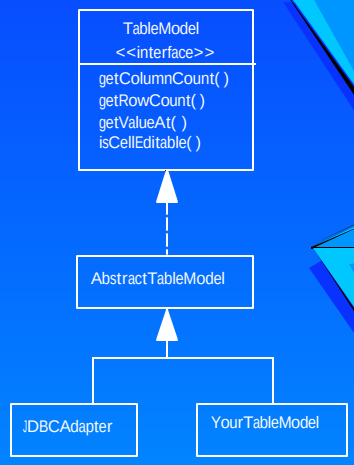


JTable

- JTable is a Tabular (row-column) view with read-only or editable cells
- Can create a JTable on:
  - Two Arrays
    - 2d array of data, 1d array of column headings
  - Two Vectors
    - (data of length row X columns), column headings
  - TableModel, (optionally) ListSelectionModel and TableColumnModel

Table Models

- TableModel is an interface that defines the row/column behavior
  - AbstractTableModel implements most of the behavior
  - you implement getColumnCount(), getRowCount(), getValueAt()



Example Table Model

```
public ExampleTableCustomModel() {
      super();
      JTable table = new JTable(new CustomDataModel());
      createColumnHeadings(table);
      getContentPane().add(table.createScrollPaneForTable(table));
}


public void createColumnHeadings(JTable table) {
      String headers[] = {"Zero", "One", "Two", "Three", "Four"};
      table.setAutoCreateColumnsFromModel(false);
      for (int i=0; i<5; i++) {
            TableColumn column = new TableColumn(i);
            column.setHeaderValue(headers[i]);
            table.addColumn(column);
      }
}
```

Example Custom Model

```
public class CustomDataModel extends AbstractTableModel {

/**
 * getColumnCount returns 0 since we will create columns ourselves
 */
public int getColumnCount() { return 0;}

public int getRowCount() {return 5;}

/**
 * getValueAt is semi-bogus; it returns a string of row * column.
 */
public Object getValueAt(int row, int col) {
      return Integer.toString(row * col);
}

}
```

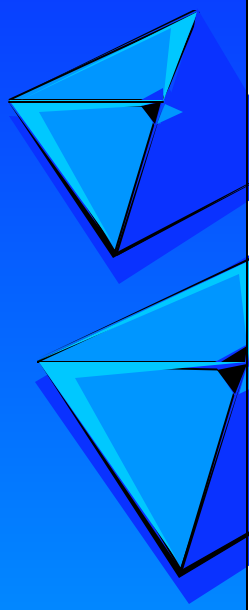
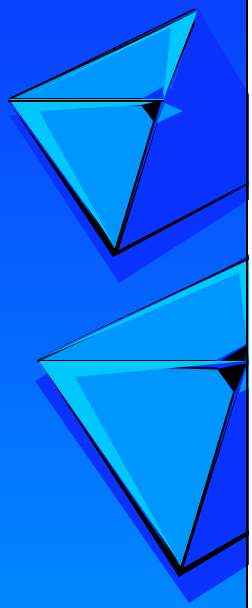


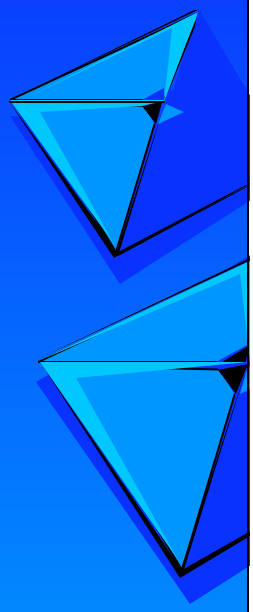
Table Selection

- There are several selection attributes of JTable you can set
  - setRowSelectionAllowed(boolean value)
  - setColumnSelectionAllowed(boolean value)
  - setSelectionForeground(Color value)
  - setSelectionBackground(Color value)
- You can also turn horizontal and vertical lines on and off



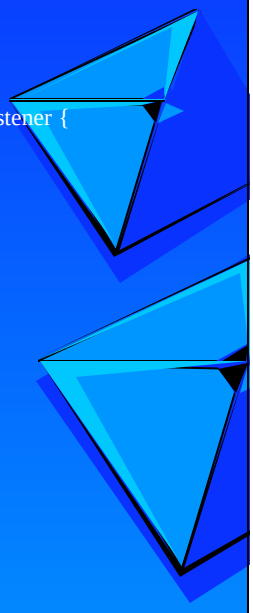
List Selection

- JTables support ListSelection notification
  - Clients must implement the ListSelectionListener interface
  - Just like JLists in that respect
- The Event doesn't carry the necessary selection information
  - You need to query the table for selection information
  - Use getSelectedRow(), getSelectedColumn(), getValueAt()



TableSelectionListener Example

```
public class ExampleTableSelection extends JFrame implements ListSelectionListener {
…
ExampleTableSelection() {
…
        JTable table = new JTable(someModel);
        ListSelectionModel selectionModel = table.getSelectionModel();
        selectionModel.addListSelectionListener(this);
…
}
public void valueChanged(ListSelectionEvent e) {
        JTable table = (JTable) e.getSource();
        DefaultTableModel model = (DefaultTableModel) table.getModel();
        int row = table.getSelectedRow();
        int column = table.getSelectedColumn();
        String value = (String) model.getValueAt(row, column);
        System.out.println("Selected value is " + value);
}
…
}
```

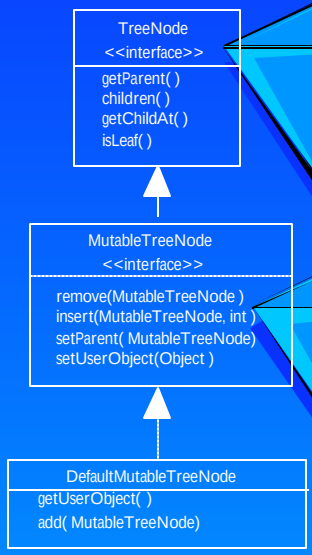


Swing Trees

- JTree is a hierarchical display component
  - allows expansion, contraction, editing of nodes
- JTree displays a TreeModel
  - TreeModel is built from TreeNodes
  - MutableTreeNodes hold arbitrary objects

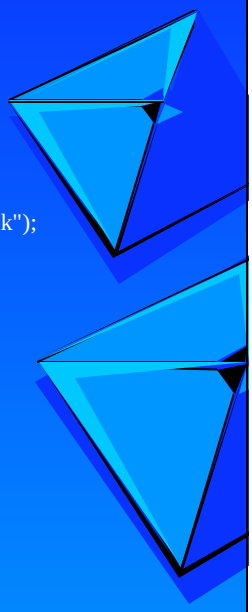
TreeModel Hierarchy

- TreeNode is an interface that describes node behavior
- MutableTreeNode is an interface that allows addition/removal of children
- DefaultMutableTreeNode implements MutableTreeNode



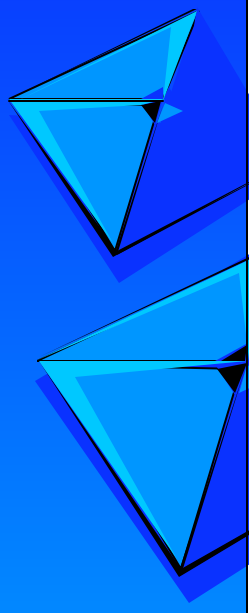
TreeNode Example

```
public DefaultMutableTreeNode buildTree() {
      DefaultMutableTreeNode root = new DefaultMutableTreeNode("Classes");
      DefaultMutableTreeNode level1a = new DefaultMutableTreeNode("Java");
      DefaultMutableTreeNode level1b = new DefaultMutableTreeNode("Smalltalk");
      root.add(level1a);
      root.add(level1b);
      level1a.add(new DefaultMutableTreeNode("Introduction to Java"));
      level1a.add(new DefaultMutableTreeNode("Advanced Java"));
      level1a.add(new DefaultMutableTreeNode("Enterprise Java Programming"));
      return root;
}

public ExampleSimpleTree() {
      JScrollPane scroller = new JScrollPane();
      JTree tree = new JTree(buildTree());
      scroller.getViewport().add(tree);
      getContentPane().add(scroller);
}
```



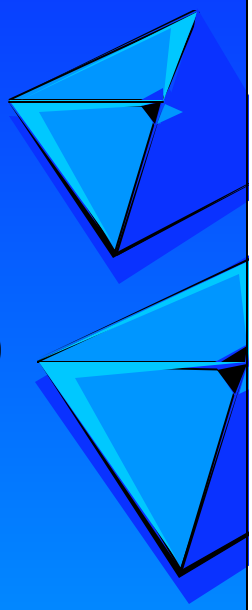
Tree Selection

- JTrees support the TreeSelectionEvent notification
  - Clients must implement the TreeSelectionListener interface
  - void valueChanged(TreeSelectionEvent e)
- Use the JTree method getSelectionPath() to get a TreePath that gives the selection(s)



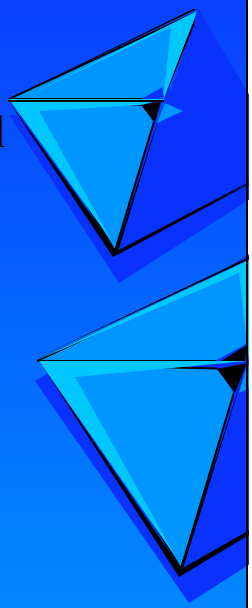
Tree Expansion

- JTrees also support the TreeExpansionEvent notification
  - Clients implement TreeExpansionListener
  - void treeExpanded(TreeExpansionEvent e)
  - void treeCollapsed(TreeExpansionEvent e)
- You can ask the TreeExpansionEvent to getPath() and return the affected path



Summary

- Swing is a comprehensive expansion and replacement for AWT
- You've seen some basic and advanced concepts in Swing Programming



More Information

- For more information, see the Swing tutorials on the Sun Java Developer's Connection website (http://developer.java.sun.com)
  - requires registration (free)
- Steven Gutz, "Up to Speed With Swing, 2nd Edition", Manning, 2000