Swing 2

Widget JComponent Example

setString(string)

Change what we display, and possibly change our preferred size.

Repaint

Demonstrates the repaint-in-setter idea from last time

Revalidate

Tell the layout manager to re-do the layout computation. We do it here because we have changed our preferred size. It's somewhat rare to need to call revalidate().

Code

```
import java.awt.*;
import javax.swing.*;

// The minimal JComponent subclass
// Now with setString() behavior which may or may not
// change our preferred size.
public class Widget extends JComponent {
   private String string;
   private boolean doesResize = true;

   public Widget(String string) {
      super();
      setString(string);
   }

   public Widget() {
      this("");
      if (!doesResize) setPreferredSize(new Dimension(100, 20));
   }

   public void setString(String newString) {
      string = newString;
      repaint();
      if (doesResize) {
         setPreferredSize(new Dimension(string.length() * 8, 20));
         revalidate();
      }

   }

   public String getString() {
      return string;
```

```
   }

   public void paint(Graphics g) {
      g.setColor(Color.red);
      g.drawRect(0, 0, getWidth()-1, getHeight()-1);

      g.drawString(string, 4, 14);

      //where();
   }


   public void setDoesResize(boolean doesResize) {
      this.doesResize = doesResize;
      // Is naming the parameter the same as the instance var a good idea?
   }

   // Used for debugging
   public void where() {
      System.out.println("x:" + getX() + " y:" + getY() + " width:" + getWidth() + "
height:" + getHeight());
   }

}
```

Old Action Model

Override processActionEvent()
send enableEvents(*mask*) to "turn on" events
Bad: overriding it to heavyweight -- what if I just want to use something off the shelf?
Bad: too many notifications that receivers don't care about

New Action Model: Listeners

Register

A Listener registers that it is interested in a type of event with the component where the event happens
When the event happens, all the listeners are notified (the order in which the listeners are told is unspecified which can be a source or problems)
There are subclasses WindowListener, ItemListener, ... specialized for the appropriate sort of notification.

Window Examples

WindowListener

All 7 "notifications" of window type events

WindowAdapater

Subclass with empty {} definitions, so you don't need to override all 7 notifications.

Button Example

ActionListener

Get a notification when the button is clicked

Inner Class issues

Have a pointer to the enclosing class, so can see its ivars
Can see local (stack) vars if they are final

ComboBox

Item Listener

Hear about what is clicked in the combo box

Code

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

import java.awt.event.*;
import java.util.*;

public class Listeners {

   // Build a window with a WindowListener
   public static void closable() {
      JFrame frame = new JFrame("WindowListener");

      frame.addWindowListener(
         // WindowListener prototypes the 7 notifictions for frames
         new WindowListener() {
            public void windowClosing(WindowEvent e) {
               System.out.println("closing");
               // this is a common operation for WindowListeners
               //System.exit(0);
            }
            public void windowClosed(WindowEvent e)
               { System.out.println("closed"); }
            public void windowOpened(WindowEvent e)
               { System.out.println("opened"); }
            public void windowActivated(WindowEvent e)
               { System.out.println("activated"); }
            public void windowDeactivated(WindowEvent e)
               { System.out.println("deactivated"); }
            public void windowIconified(WindowEvent e)
               { System.out.println("iconified"); }
            public void windowDeiconified(WindowEvent e)
               { System.out.println("deiconified"); }
         }
      );

      // Set the window close-box behavior (hide is the default)
      frame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
```

```
         // or DISPOSE_ON_CLOSE or EXIT_ON_CLOSE (new Java 2 option)

      frame.setVisible(true);
   }

   public static void closable2() {
      JFrame frame = new JFrame("Closable2");

      frame.addWindowListener(
         // The "adapter" version provides empty {} definitions for all the
         // notifications, so you can just override the one you care about
         // (make sure you get the prototype exactly right).
         new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
               System.exit(0);
            }
         }
      );

      frame.setVisible(true);
   }

   public static void main(String args[]) {
      closable();
      new ButtonFrame("Buttons");
      new ComboFrame("Combo");

   }
}

/*
 Demonstrates putting an action listener on a button.
 Notice how the inner class can see the ivars of its
 enclosure.
 It can also see stack vars, but only if they are final.
*/
class ButtonFrame extends JFrame {

   private Widget widget;
   private JButton button;
   private JCheckBox checkbox;

   public ButtonFrame(String title) {
      super(title);

      Container container = getContentPane();
      container.setLayout(new FlowLayout());

      widget = new Widget("Hello");
      container.add(widget);

      button = new JButton("Add '!'");
      container.add(button);

      final char suffix = '!';   // example stack var for inner class

      // Classic action listener
      // Translates the incoming action to a local operation
```

```
      // Note how it can access widget and suffix
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               String string = widget.getString();
               widget.setString(string + suffix);
            }
         }
      );


      checkbox = new JCheckBox("Dynamic Sizing", true);
      container.add(checkbox);
      checkbox.addItemListener(
         new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
               // Move the boolean from the checkbox to the widget
               widget.setDoesResize(e.getStateChange() == ItemEvent.SELECTED);
            }
         }
      );

      setVisible(true);
   }
}

// A frame with a combo box connected to a widget
class ComboFrame extends JFrame {
   private Widget widget;

   public ComboFrame(String title) {
      super(title);

      Container container = getContentPane();
      container.setLayout(new FlowLayout());

      widget = new Widget("Hello");
      container.add(widget);

      buildCombo();

      setVisible(true);
   }

   // A helper function to build the combo box
   private void buildCombo() {

      String choices[] = {"Clattu", "Barrada", "Nickto", "Resistance is futile",
         "We come in peace", "Tried have you?"};
      JComboBox combo = new JComboBox(choices);

      // Connect selection changes to the widget
      // (The abstraction supports "multiple selection", but we're
      // not using that feature here.)
      combo.addItemListener(new ItemListener() {
         public void itemStateChanged(ItemEvent e) {
            if (e.getStateChange() == ItemEvent.SELECTED) {
               widget.setString(
```

```
                  //(String)e.getItemSelectable().getSelectedObjects()[0]
                  (String)e.getItem()
               );
            }
         }
      });

      // combo.setSelectedIndex(0); // actually, it comes up this way anyway
      widget.setString((String)combo.getSelectedItem());

      Container container = getContentPane();
      container.add(combo);
   }

}
```

Model-View-Controller

View

Display

Model

Storage

Controller

Control, notification, coordination

Advantages

Multiple views

Modularity

e.g. Undo, file save are pure model side operations

3 Layers

ListModel

AbstractListModel

Keeps list of listeners, supports fireXXX events

DefaultListModel

Code

```
import java.awt.*;
```

```
import javax.swing.*;
import javax.swing.border.*;

import java.awt.event.*;
import java.util.*;  // for Vector

import javax.swing.event.*;

import java.io.*; // for File

/*
 An example of implemention a ListModel.
 In this case, we just use Vector.
 AbstractListModel keeps track of the listeners for us, but
 we still need to trigger the notifications.
*/
class MyListModel extends AbstractListModel {

   private Vector data = new  Vector();

   // Must override these two from ListModel
   public int getSize() {
      return(data.size());
   }

   public Object getElementAt(int index) {
      return(data.elementAt(index));   // could sanity check index
   }


   // My methods so clients can add and remove elements
   // on the data model (clients can also use the standard
   // getElementAt() for accessing).
   public int addRow(String string) {
      data.addElement(string);

      // Must send the following
      // (AbstractListModel provides the listener support for us)
      fireIntervalAdded(this, data.size()-1, data.size()-1);
      return(data.size()-1);
   }

   public void deleteRow(int row) {
      // ??? could error check the row int
      data.removeElementAt(row);
      fireIntervalRemoved(this, row, row);
   }

   // If we had an operation that CHANGED the contents of a row, we would
   // fireContentsChanged(this, row, row)
}

public class TrivialApplication {

   static int count = 0;

   // Demonstrate MyListModel
   public static void list() {
```

```
   final JFrame frame = new JFrame("List");
   final Container container = frame.getContentPane();
   container.setLayout(new FlowLayout());

   final MyListModel listModel = new MyListModel();
   final JList list = new JList(listModel);
   JScrollPane scrollpane = new JScrollPane(list);

   // could do this, or use the medium default size
   // scrollpane.setPreferredSize(new Dimension(200,120));

   // Button to add a row to the model
   JButton button = new JButton("Add Row");
   container.add(button);
   button.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            count++;
            int newRow = listModel.addRow(Integer.toString(count));
            list.setSelectedIndex(newRow);
         }
      }
   );


   // Delete the currently selected row
   final JButton button2 = new JButton("Delete Row");
   container.add(button2);
   button2.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            // ??? can return -1
            listModel.deleteRow(list.getSelectedIndex());
            list.clearSelection();
         }
      }
   );


   // The correct way to screen out the above -1 case with the correct HI
   // is to disable the button when there is no selection --
   // a good HI reflects the current state visually.
   // But it doesn't work -- Swing bug or at least a design flaw
   // These notifications appear to only be set when a selection happens
   // but not when it goes away because of a deletion.
   list.getSelectionModel().addListSelectionListener(
      new ListSelectionListener()  {
            public void valueChanged(ListSelectionEvent e) {
               System.out.println("Button :" + list.getSelectedIndex());
               button2.setEnabled(list.getSelectedIndex() != -1);
            }
         }
      );

   container.add(scrollpane);
   frame.setVisible(true);

}
```

```
// DefaultComboBoxModel has basic storage built in
// -- has getSize() and addElement()
static class SharedModel extends DefaultComboBoxModel {
   private final static String[] strings = {"Nick", "Julia", "Mitch"};

   public SharedModel() {
      super();

      for(int i=0; i<strings.length; i++) {
         addElement(strings[i]);
      }
   }
}

public static void shared() {

   final JFrame frame = new JFrame("Shared");
   final Container container = frame.getContentPane();
   container.setLayout(new FlowLayout());

   final SharedModel model = new SharedModel();
   final JList list = new JList(model);
   JScrollPane scrollpane = new JScrollPane(list);
   scrollpane.setPreferredSize(new Dimension(100,100));
   container.add(scrollpane);
   list.getSelectionModel().
      setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

   JComboBox combo = new JComboBox(model);
   container.add(combo);


   // Add a row the model
   JButton button = new JButton("Add Row");
   container.add(button);
   button.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            count++;
            model.addElement(Integer.toString(count));

         }
      }
   );

   // Read the lines out of a file and add them
   JButton button2 = new JButton("Load File");
   container.add(button2);
   button2.addActionListener(
      new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JFileChooser chooser = new JFileChooser(".");
            int status = chooser.showOpenDialog(frame);
            if (status == JFileChooser.APPROVE_OPTION) {
               File file = chooser.getSelectedFile();

               try {
```

```
                    FileReader fileReader = new FileReader(file);

                    // The buffered layer is optional by recommended
                       Reader bufferedReader = new BufferedReader(fileReader);

                       StreamTokenizer tokenizer = new
StreamTokenizer(bufferedReader);

                    // Try to set the tokenizer to reader "words" line by line
                    //tokenizer.resetSyntax();
                    //tokenizer.whitespaceChars( '\n', '\r');    // no

                    //tokenizer.ordinaryChar(' ');   // no
                    //tokenizer.ordinaryChar('\t');

                    //tokenizer.wordChar(' '); // no such method
                    tokenizer.wordChars(' ', ' ');   // now each line counts as a
token
                    tokenizer.wordChars('\t', '\t');
                    int tok;

                    // The standard loop to get all the tokens in a file
                    while ((tok = tokenizer.nextToken()) != StreamTokenizer.TT_EOF)
{
                       model.addElement(tokenizer.sval);
                    }
                 }
                 catch (IOException ignored) {
                 }
              }
           }
        }
     );
     frame.setVisible(true);
  }

  public static void main(String args[]) {
     list();
     shared();
  }
}
```