```
Loading vc-cvs...
package caltool.schedule_ui;

import caltool.schedule.*;
import caltool.caltool_ui.*;
import mvp.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/****
 *
 * Class ScheduleEventDialog provides a view of Event as an input to the
 * scheduleEvent method.  Hence, the dialog is a view of both an Event object
 * as well as the scheduleEvent method.  The data-entry components of the
 * dialog constitute the Event view.  The 'OK' button is the view of the
 * scheduleEvent method.
 *                                                                        <p>
 * The data components consist of JLabels, JTextFields, and a JComboBox.  The
 * 'OK', 'Clear', and 'Cancel' buttons are JButtons.  The description of the <a
 * href= "#compose()"> compose </a> method has details of how the components
 * are laid out in the dialog window.
 *                                                                        <p>
 * The companion model for ScheduleEventDialog is the <a href= "Schedule.html">
 * Schedule </a> class, since Schedule has the method that is invoked from the
 * 'OK' button action listener.  See class <a href=
 * "OKScheduleEventButtonListener"> OKScheduleEventButtonListener.html </a> for
 * details of how the Schedule.scheduleEvent method is invoked.
 *
 * @author Gene Fisher (gfisher@calpoly.edu)
 * @version 6feb04
 *
 */
public class ScheduleEventDialog extends CalendarToolWindow {

    /**
     * Construct this with the given Schedule as companion model.
     */
    public ScheduleEventDialog(Screen screen, Schedule schedule,
            CalendarToolUI calToolUI) {

        /*
         * Call the parent constructor.
         */
        super(screen, schedule, calToolUI);

        /*
         * Set the maximum component height and width.  These the height value
         * was empiricaly derived, i.e., I fiddled around with it while looking
         * at the display.  The width value is presumably as big as the biggest
         * screen we'll come across.  If not, deal with it.
         */
        maxComponentHeight = 1.9;
        maxComponentWidth = 2000;
    }

    /**
     * Compose this as a vertical Box of four rows.  Each row is a horizontal
     * Box.  The first row contains a labeled text field for the event title.
     * The second row has labeled text fields for the start and end dates.  The
     * third row has a labeled combo box for the category selection and a
     * labeled combol box for the event security.  Finally, the fourth row has
     * the 'OK', 'Clear', and 'Cancel' buttons.
     *                                                                    <p>
     * Vertical and horizontal struts are used for spacing among all of the
     * components.
     */
    public Component compose() {

        /*
         * Add a JPanel to this' window, which was created in the parent class'
         * constructor.  JPanel is the standard background container for
         * holding Swing components.
         */
        panel = new JPanel();
        window.add(panel);

        /*
         * Set the layout style of the panel to be a vertical box.
         */
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        /*
         * Compose each of the rows and add to the vertically laid out panel.
         * Put some spacing between each row, in the form of a vertical strut.
         */
        panel.add(Box.createVerticalStrut(15));
        panel.add(composeTitleRow());
        panel.add(Box.createVerticalStrut(15));
        panel.add(composeStartAndEndDateRow());
        panel.add(Box.createVerticalStrut(15));
        panel.add(composeCategoryAndSecurityRow());
        panel.add(Box.createVerticalStrut(25));
        panel.add(composeButtonRow());
        panel.add(Box.createVerticalStrut(15));

        /*
         * Set the window titlebar.
         */
        window.setTitle("Schedule an Event");

        /*
         * Call JFrame.pack to have Java size up the window properly.
         */
        window.pack();

        /*
         * Return the window to the caller.
         */
        return window;
    }

```

```
112     /*-*
113      * Data collection methods.
114      */
115
116     /**
117      * Return the title as a string.
118      */
119     public String getTitle() {
120         return titleTextField.getText();
121     }
122
123     /**
124      * Return the start date as a Date value.  Note that the Date model class
125      * does the parsing.
126      */
127     public Date getStartDate() {
128         return new Date(startDateTextField.getText());
129     }
130
131     /**
132      * Return the end date as a Date value.  Note that the Date model class
133      * does the parsing.
134      */
135     public Date getEndDate() {
136         return new Date(startDateTextField.getText());
137     }
138
139     /**
140      * Return selected category as a Category value.
141      * does the parsing.
142      */
143     public Category getCategory() {
144         return new Category((String) categoryComboBox.getSelectedItem());
145     }
146
147     /**
148      * Return the security as a simple security value.
149      */
150     public SimpleSecurity getSecurity() {
151         return SimpleSecurity.valueOf(
152             (String) securityComboBox.getSelectedItem());
153     }
154
155     /**
156      * Display the error messages in the given exception object in a modal
157      * pop-up window.
158      */
159     public void displayErrors(ScheduleEventPrecondViolation errors) {
160
161         int i;                  // Error array loop index
162
163         /*
164          * Stubbed out implementation to std err instead of pop-up window.
165          */
166         for (i = 0; i < errors.numberOfErrors(); i++) {
167             System.err.println(errors.getErrors()[i]);
168         }
169
170     }
171
172     /**
173      * Compose the title row using a JLabel and JTextField.
174      */
175     protected Box composeTitleRow() {
176
177         Box hbox = Box.createHorizontalBox();
178
179         /*
180          * Construct the label and text field.
181          */
182         JLabel label = new JLabel("Title:  ");
183         titleTextField = new JTextField();
184
185         /*
186          * Set the label font color to black, since I don't care for the
187          * default "Java blue".
188          */
189         label.setForeground(Color.black);
190
191         /*
192          * Set the max height of the text field to keep it from resizing
193          * vertically in the vertical box layout of the panel.
194          */
195         titleTextField.setMaximumSize(
196             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
197                 titleTextField.getFont().getSize())));
198
199         /*
200          * Add the label and text field to the hbox and return it.  Use
201          * horizontal struts for spacing.
202          */
203         hbox.add(Box.createHorizontalStrut(15));
204         hbox.add(label);
205         hbox.add(titleTextField);
206         titleTextField.setAlignmentX(Component.RIGHT_ALIGNMENT);
207         hbox.add(Box.createHorizontalStrut(15));
208         return hbox;
209
210     }
211
212     /**
213      * Compose the start/end date row using two pairs of JLabel and JTextField.
214      */
215     protected Box composeStartAndEndDateRow() {
216
217         Box hbox = Box.createHorizontalBox();
218
219         /*
220          * Construct the labels and text fields.  See internal comments in the
221          * composeTitle method for further explanatory details.
222          */
223         JLabel startLabel = new JLabel("Start Date:  ");
```

```
224         startLabel.setForeground(Color.black);
225         startDateTextField = new JTextField(15);
226         startDateTextField.setMaximumSize(
227             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
228                 startDateTextField.getFont().getSize())));
229         JLabel endLabel = new JLabel("End Date:  ");
230         endLabel.setForeground(Color.black);
231         endDateTextField = new JTextField(15);
232         endDateTextField.setMaximumSize(
233             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
234                 startDateTextField.getFont().getSize())));
235 
236         /*
237          * Add them to the hbox and return it.
238          */
239         hbox.add(Box.createHorizontalStrut(15));
240         hbox.add(startLabel);
241         hbox.add(startDateTextField);
242         hbox.add(Box.createHorizontalStrut(10));
243         hbox.add(endLabel);
244         hbox.add(endDateTextField);
245         hbox.add(Box.createHorizontalStrut(15));
246         return hbox;
247 
248     }
249 
250     /**
251      * Compose the category/security row using a JComboBox and JTextField, with
252      * JLabels next to each.
253      */
254     protected Box composeCategoryAndSecurityRow() {
255 
256         Box hbox = Box.createHorizontalBox();
257 
258         /*
259          * Construct the labels and text fields.  See internal comments in the
260          * composeTitle method for further explanatory details.
261          */
262         JLabel categoryLabel = new JLabel("Category:  ");
263         categoryLabel.setForeground(Color.black);
264         categoryComboBox = new JComboBox();
265         categoryComboBox.addItem("none");
266 
267         JLabel securityLabel = new JLabel("Security:  ");
268         securityLabel.setForeground(Color.black);
269         String[] selections = {"public", "private"};
270         securityComboBox = new JComboBox(selections);
271 
272         /*
273          * Add them to the hbox and return it.
274          */
275         hbox.add(Box.createHorizontalStrut(15));
276         hbox.add(categoryLabel);
277         hbox.add(categoryComboBox);
278         hbox.add(Box.createHorizontalStrut(10));
279         hbox.add(securityLabel);
280         hbox.add(securityComboBox);
281         hbox.add(Box.createHorizontalStrut(15));
282         return hbox;
283 
284     }
285 
286     /**
287      * Compose the buttons row with three JButtons.  The action listeners for
288      * Clear and Cancel buttons are straightforward.  The action listener for
289      * the OK button is responsible for communication with the Schedule model.
290      * See the description of <a href= "OKScheduleEventButtonListener.html>"
291      * OKScheduleEventButtonListener </a> for explanatory details.
292      *
293      */
294     protected Box composeButtonRow() {
295 
296         Box hbox = Box.createHorizontalBox();
297 
298         /*
299          * Construct the three buttons.
300          */
301         JButton okButton = new JButton("OK");
302         JButton clearButton = new JButton("Clear");
303         JButton cancelButton = new JButton("Cancel");
304 
305         /*
306          * Attach the appropriate action listeners to each button.
307          */
308         okButton.addActionListener(
309             new OKScheduleEventButtonListener((Schedule) model, this));
310 
311         clearButton.addActionListener(
312             new ActionListener() {
313                 public void actionPerformed(ActionEvent e) {
314                     clear();
315                 }
316             }
317         );
318 
319         cancelButton.addActionListener(
320             new ActionListener() {
321                 public void actionPerformed(ActionEvent e) {
322                     hide();
323                 }
324             }
325         );
326 
327         /*
328          * Add them to the hbox and return it.
329          */
330         hbox.add(okButton);
331         hbox.add(Box.createHorizontalStrut(30));
332         hbox.add(clearButton);
333         hbox.add(Box.createHorizontalStrut(30));
334         hbox.add(cancelButton);
335         return hbox;
```

```
336
337     }
338
339     /**
340      * Clear each of the text fields of this to empty.  Reset the combo box to
341      * no selection.  NOTE: This method needs to be refined to use default
342      * values for clearing, once options and defaults functionality is
343      * implemented.
344      */
345     protected void clear() {
346         titleTextField.setText("");
347         startDateTextField.setText("");
348         endDateTextField.setText("");
349         categoryComboBox.setSelectedIndex(0);
350         securityComboBox.setSelectedIndex(0);
351     }
352
353     /** The background panel of this */
354     protected JPanel panel;
355
356     /** The title text field */
357     protected JTextField titleTextField;
358
359     /** The start date text field */
360     protected JTextField startDateTextField;
361
362     /** The end date text field */
363     protected JTextField endDateTextField;
364
365     /** The Categories combo box */
366     protected JComboBox categoryComboBox;
367
368     /** The security text field */
369     protected JComboBox securityComboBox;
370
371     /** The max height of a text field or combobox; this is necessary since
372         these components stretch when the outer frame is resized, and look very
373         funky when they do. */
374     protected final double maxComponentHeight;
375
376     /** The max width of any component; this is only necessary because the max
377         height cannot be set separately, so we must pick some max width. */
378     protected final int maxComponentWidth;
379
380 }
```