```
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 ScheduleAppointmentDialog provides a view of Appointment as an input
 * to the scheduleAppointment method.  Hence, the dialog is a view of both an
 * Appointment object as well as the scheduleAppointment method.  The
 * data-entry components of the dialog constitute the Appointment view.  The
 * 'OK' button is the view of the ScheduleAppointment method.
 *                                                                        <p>
 * The data components consist of JLabels, JTextFields, JComboBoxes,
 * JCheckBoxes, and a JTextArea.  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>
 * For organizational clarity, two of the rows in the ScheduleAppointment
 * Dialog are defined in separate classes.  These are the <a href=
 * RecurringInfoSubdialog.html> RecurringInfoSubdialog </a> and <a href=
 * RemindInfoSubdialog.html> RemindInfoSubdialog </a> classes.
 *                                                                        <p>
 * The companion model for ScheduleAppointmentDialog is the <a href=
 * "../schedule/Schedule.html"> Schedule </a> class, since Schedule has the
 * method that is invoked from the 'OK' button action listener.  See class <a
 * href= "OKScheduleAppointmentButtonListener.html">
 * OKScheduleAppointmentButtonListener </a> for details of how the
 * Schedule.scheduleAppointment method is invoked.
 *
 */
public class ScheduleAppointmentDialog extends CalendarToolWindow {

    /**
     * Construct this with the given Schedule as companion model.  Construct
     * the two subviews for recurring info and reminder info.
     */
    public ScheduleAppointmentDialog(Screen screen, Schedule schedule,
            CalendarToolUI calToolUI) {
        super(screen, schedule, calToolUI);

        recurringInfo = new RecurringInfoSubdialog(screen, this);
        remindInfo = new RemindInfoSubdialog(screen, this);

        /*
         * Set the maximum component height and width.  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 = 3000;
    }

    /**
     * Compose this in six parts: (1) a top part consisting of the title, date,
     * end date, start time, and duration components; (2) a part consisting of
     * recurring info components; (3) a middle part with category, location,
     * security, and priority; (4) reminder info components; (5) details
     * components; (6) the bottom row consisting of the 'OK', 'Clear', and
     * 'Cancel' buttons.
     */
    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 the content rows.
         */
        composeRows();

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

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

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

    /**
     * Compose each of the rows and add to the vertically laid out panel.
     * Put some around spacing between each row, in the form of a vertical
     * strut.
     */
    protected void composeRows() {
        panel.add(Box.createVerticalStrut(15));
        panel.add(composeTopPart());
        panel.add(Box.createVerticalStrut(15));
```

```
112         panel.add(composeRecurringInfo());
113         panel.add(Box.createVerticalStrut(15));
114         panel.add(composeMiddlePart());
115         panel.add(Box.createVerticalStrut(15));
116         panel.add(composeRemindInfo());
117         panel.add(Box.createVerticalStrut(15));
118         panel.add(composeDetails());
119         panel.add(Box.createVerticalStrut(15));
120         panel.add(composeButtonRow());
121         panel.add(Box.createVerticalStrut(15));
122     }
123
124     /**
125      * Compose the top part of the dialog, consisting of the title, date, start
126      * time, end date, and duration.  The components are laid out in a
127      * three-row vertical box.  The title row is on top; it is composed as a
128      * horizontal box containing a JLabel and JTextField.  The second row of
129      * top-part components is a horizontal box, composed in turn of two
130      * horizontal boxes containing JLabel/JTextField pairs for the date and
131      * start time.  The third row consists of a JLabel/JTextField pairs for the
132      * end date and the duration.  The duration is in turn a horizontal box
133      * consisting of a JLabel, and two vertical box JLabel/JTextField pairs for
134      * the hour and minute components of the duration.
135      */
136     protected Box composeTopPart() {
137         Box vbox = Box.createVerticalBox();
138
139         vbox.add(composeTitleRow());
140         vbox.add(Box.createVerticalStrut(15));
141         vbox.add(composeStartDateRow());
142         vbox.add(Box.createVerticalStrut(15));
143         vbox.add(composeEndDateRow());
144
145         return vbox;
146     }
147
148     /**
149      * Compose the title row as an hbox with label and text field.
150      */
151     protected Box composeTitleRow() {
152
153         Box hbox = Box.createHorizontalBox();
154
155         /*
156          * Construct the label and text field.
157          */
158         JLabel label = new JLabel("Title:  ");
159         titleTextField = new JTextField();
160
161         /*
162          * Set the label font color to black, since I don't care for the
163          * default "Java blue".
164          */
165         label.setForeground(Color.black);
166
167         /*
168          * Set the max height of the text field to keep it from resizing
169          * vertically in the vertical box layout of the panel.
170          */
171         titleTextField.setMaximumSize(
172             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
173                 titleTextField.getFont().getSize())));
174
175         /*
176          * Add the label and text field to the hbox and return it.  Use
177          * horizontal struts for spacing.
178          */
179         hbox.add(Box.createHorizontalStrut(15));
180         hbox.add(label);
181         hbox.add(titleTextField);
182         hbox.add(Box.createHorizontalStrut(15));
183         return hbox;
184
185     }
186
187     /**
188      * Compose the start date row using two pairs of labels and text fields.
189      * Cf. <a href= "ScheduleEventDialg#composeStartAndEndDateRow()">
190      * ScheduleEventDialg.composeStartAndEndDateRow </a>.
191      */
192     protected Box composeStartDateRow() {
193
194         Box hbox = Box.createHorizontalBox();
195
196         /*
197          * Construct the labels and text fields.  See internal comments in the
198          * composeTitle method for further explanatory details.
199          */
200         startDateLabel = new JLabel("Date:  ");
201         startDateLabel.setForeground(Color.black);
202         startDateTextField = new JTextField(15);
203         startDateTextField.setMaximumSize(
204             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
205                 startDateTextField.getFont().getSize())));
206         JLabel startTimeLabel = new JLabel("Start Time:  ");
207         startTimeLabel.setForeground(Color.black);
208         startTimeTextField = new JTextField(15);
209         startTimeTextField.setMaximumSize(
210             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
211                 startTimeTextField.getFont().getSize())));
212
213         /*
214          * Add them to the hbox and return it.
215          */
216         hbox.add(Box.createHorizontalStrut(15));
217         hbox.add(startDateLabel);
218         hbox.add(startDateTextField);
219         hbox.add(Box.createHorizontalStrut(10));
220         hbox.add(startTimeLabel);
221         hbox.add(startTimeTextField);
222         hbox.add(Box.createHorizontalStrut(15));
223         return hbox;
```

```
224
225     }
226
227     /**
228      * Compose the end date row using two pairs of labels and text fields.
229      */
230     protected Box composeEndDateRow() {
231
232         Box hbox = Box.createHorizontalBox();
233
234         /*
235          * Construct the labels and text fields.  See internal comments in the
236          * composeTitle method for further explanatory details.
237          */
238         endDateLabel = new JLabel("End Date:  ");
239         endDateLabel.setForeground(Color.black);
240         endDateLabel.setEnabled(false);
241
242         endDateTextField = new JTextField(15);
243         endDateTextField.setMaximumSize(
244             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
245                 endDateTextField.getFont().getSize())));
246         endDateTextField.setEnabled(false);
247
248         JLabel durationLabel = new JLabel("Duration:  ");
249         durationLabel.setForeground(Color.black);
250         durationTextField = new JTextField(15);
251         durationTextField.setMaximumSize(
252             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
253                 durationTextField.getFont().getSize())));
254
255         /*
256          * Add them to the hbox and return it.
257          */
258         hbox.add(Box.createHorizontalStrut(15));
259         hbox.add(endDateLabel);
260         hbox.add(endDateTextField);
261         hbox.add(Box.createHorizontalStrut(10));
262         hbox.add(durationLabel);
263         hbox.add(durationTextField);
264         hbox.add(Box.createHorizontalStrut(15));
265         return hbox;
266
267     }
268
269     /**
270      * Have the recurring info subdialog compose itself.
271      */
272     protected Component composeRecurringInfo() {
273         return recurringInfo.compose();
274     }
275
276     /**
277      * Compose the middle part of the dialog, consisting of the category,
278      * location, security, and priority.  The category and security are combo
279      * boxes, laid out in a horizontal box.  Location and priority are also
280      * combo boxes in a horizontal box.  Location is editable.  See the
281      * description of <a href= "ScheduleEventDialog#composeTopPart()">
282      * composeTopPart for a more detailed description of component layout.
283      */
284     protected Box composeMiddlePart() {
285         Box vbox = Box.createVerticalBox();
286
287         vbox.add(composeCategorySecurityRow());
288         vbox.add(Box.createVerticalStrut(15));
289         vbox.add(composeLocationPriorityRow());
290
291         return vbox;
292     }
293
294     /**
295      * Compose the category/seurity row using two pairs of labels and text
296      * fields.
297      */
298     protected Box composeCategorySecurityRow() {
299         Box hbox = Box.createHorizontalBox();
300
301         JLabel categoryLabel = new JLabel("Category:  ");
302         categoryLabel.setForeground(Color.black);
303         categoryComboBox = new JComboBox();
304         categoryComboBox.addItem("none");
305         categoryComboBox.addItem("");
306         categoryComboBox.addItem("Edit ...");
307         categoryComboBox.setMaximumSize(
308             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
309                 categoryComboBox.getFont().getSize())));
310
311         JLabel securityLabel = new JLabel("Security:  ");
312         securityLabel.setForeground(Color.black);
313         String[] selections2 =
314             {"public", "title only", "confidential", "private"};
315         securityComboBox = new JComboBox(selections2);
316         securityComboBox.setMaximumSize(
317             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
318                 securityComboBox.getFont().getSize())));
319
320         hbox.add(Box.createHorizontalStrut(15));
321         hbox.add(categoryLabel);
322         hbox.add(categoryComboBox);
323         hbox.add(Box.createHorizontalStrut(10));
324         hbox.add(securityLabel);
325         hbox.add(securityComboBox);
326         hbox.add(Box.createHorizontalStrut(15));
327
328         return hbox;
329
330     }
331
332     /**
333      * Compose the location/priority row using two pairs of labels and text
334      * fields.
335      */
```

```
336     protected Box composeLocationPriorityRow() {
337         Box hbox = Box.createHorizontalBox();
338 
339         JLabel locationLabel = new JLabel("Location:  ");
340         locationLabel.setForeground(Color.black);
341         locationComboBox = new JComboBox();
342         locationComboBox.setEditable(true);
343         locationComboBox.setMaximumSize(
344             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
345                 locationComboBox.getFont().getSize())));
346 
347         JLabel priorityLabel = new JLabel("Priority:  ");
348         priorityLabel.setForeground(Color.black);
349         String[] selections = {"must", "optional"};
350         priorityComboBox = new JComboBox(selections);
351         priorityComboBox.setMaximumSize(
352             new Dimension(maxComponentWidth, (int)(maxComponentHeight *
353                 priorityComboBox.getFont().getSize())));
354 
355         hbox.add(Box.createHorizontalStrut(15));
356         hbox.add(locationLabel);
357         hbox.add(locationComboBox);
358         hbox.add(Box.createHorizontalStrut(10));
359         hbox.add(priorityLabel);
360         hbox.add(priorityComboBox);
361         hbox.add(Box.createHorizontalStrut(15));
362 
363         return hbox;
364 
365     }
366 
367     /**
368      * Have the remind info subdialog compose itself.
369      */
370     protected Component composeRemindInfo() {
371         return remindInfo.compose();
372     }
373 
374     /**
375      * Compose the details area as a labeled, scrolling text area.  The label
376      * and the text area are in a left-aligned vbox.  The vbox is in a hbox
377      * with horizontal spacing on each side
378      */
379     protected Box composeDetails() {
380         Box hbox = Box.createHorizontalBox();
381         Box vbox = Box.createVerticalBox();
382 
383         JLabel label = new JLabel("Details:");
384         label.setForeground(Color.black);
385         label.setAlignmentX(Box.LEFT_ALIGNMENT);
386         detailsTextArea = new JTextArea(6, 40);
387         JScrollPane scrollPane = new JScrollPane(detailsTextArea);
388         scrollPane.setAlignmentX(Box.LEFT_ALIGNMENT);
389 
390         vbox.add(label);
391         vbox.add(scrollPane);
392 
393         hbox.add(Box.createHorizontalStrut(15));
394         hbox.add(vbox);
395         hbox.add(Box.createHorizontalStrut(15));
396 
397         return hbox;
398     }
399 
400     /**
401      * Compose the buttons row with three JButtons.  The action listeners for
402      * Clear and Cancel buttons are straightforward.  The action listener for
403      * the OK button is responsible for communication with the Schedule model.
404      * See the description of <a href=
405      * "OKScheduleAppointmentButtonListener.html">
406      * OKScheduleAppointmentButtonListener </a> for explanatory details.
407      */
408     protected Box composeButtonRow() {
409 
410         Box hbox = Box.createHorizontalBox();
411 
412         /*
413          * Construct the three buttons.
414          */
415         JButton okButton = new JButton("OK");
416         JButton clearButton = new JButton("Clear");
417         JButton cancelButton = new JButton("Cancel");
418 
419         /*
420          * Attach the appropriate action listeners to each button.
421          */
422         okButton.addActionListener(
423             new OKScheduleAppointmentButtonListener((Schedule) model, this));
424 
425         clearButton.addActionListener(
426             new ActionListener() {
427                 public void actionPerformed(ActionEvent e) {
428                     clear();
429                 }
430             }
431         );
432 
433         cancelButton.addActionListener(
434             new ActionListener() {
435                 public void actionPerformed(ActionEvent e) {
436                     hide();
437                 }
438             }
439         );
440 
441         /*
442          * Add them to the hbox and return it.
443          */
444         hbox.add(okButton);
445         hbox.add(Box.createHorizontalStrut(30));
446         hbox.add(clearButton);
447         hbox.add(Box.createHorizontalStrut(30));
```

```
448         hbox.add(cancelButton);
449         return hbox;
450
451     }
452
453     /**
454      * Clear each of the text fields of this to empty.  Reset the combo box to
455      * no selection.  NOTE: This method needs to be refined to use default
456      * values for clearing, once options and defaults functionality is
457      * implemented.  It also needs to be refined to clear the recurring and
458      * remind check boxes and associated components.
459      */
460     protected void clear() {
461         titleTextField.setText("");
462         startDateTextField.setText("");
463         endDateTextField.setText("");
464         startTimeTextField.setText("");
465         durationTextField.setText("");
466         categoryComboBox.setSelectedItem("none");
467         locationComboBox.setSelectedItem(null);
468         securityComboBox.setSelectedIndex(0);
469         detailsTextArea.setText("");
470     }
471
472     /** The background panel of this */
473     protected JPanel panel;
474
475     /** The title text field */
476     protected JTextField titleTextField;
477
478     /** The (start) date label.  This needs to be a persistent data field since
479         it changes from "Date" to "Start Date" for recurring appointments.
480         Also, it has different text in the ScheduleMeetingDialog, subclass. */
481     protected JLabel startDateLabel;
482
483     /** The (start) date text field */
484     protected JTextField startDateTextField;
485
486     /** The start time text field. */
487     protected JTextField startTimeTextField;
488
489     /** The end date label.  This needs to be a persistent data field since
490         it's enabled or disabled depending on whether the appointment is
491         recurring.  Also, it has different text in the ScheduleMeetingDialog,
492         subclass.  */
493     protected JLabel endDateLabel;
494
495     /** The end date text field */
496     protected JTextField endDateTextField;
497
498     /** The start time text field */
499     protected JTextField durationTextField;
500
501     /** Subview for recurring info */
502     protected RecurringInfoSubdialog recurringInfo;
503
504     /** The category combo box */
505     protected JComboBox categoryComboBox;
506
507     /** The security combo box */
508     protected JComboBox securityComboBox;
509
510     /** The location combo box */
511     protected JComboBox locationComboBox;
512
513     /** The priority combo box */
514     protected JComboBox priorityComboBox;
515
516     /** Subview for recurring info */
517     protected RemindInfoSubdialog remindInfo;
518
519     /** The details text area */
520     protected JTextArea detailsTextArea;
521
522     /** The max height of a text field or combobox; this is necessary since
523         these components stretch when the outer frame is resized, and look very
524         funky when they do */
525     protected final double maxComponentHeight;
526
527     /** The max width of any component; this is only necessary because the max
528         height cannot be set separately, so we must pick some max width. */
529     protected final int maxComponentWidth;
530
531 }
```