```
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 ScheduleTaskDialog provides a view of Task as an input to the
 * scheduleTask method.  Hence, the dialog is a view of both an Task object as
 * well as the scheduleTask method.  The data-entry components of the dialog
 * constitute the Task view.  The 'OK' button is the view of the scheduleTask
 * method.
 *                                                                         <p>
 * For expedience, ScheduleTaskDialog extends ScheduleAppointmentDialog.
 * Appointment and task dialogs have a significant amount of structure in
 * common, though there are some key differences that require specialiation
 * here.  The additional design comments in the definition of <a href=
 * ScheduleAppointmentDialog.html> ScheduleAppointmentDialog </a> are generally
 * relevant here.
 *
 */

public class ScheduleTaskDialog extends ScheduleAppointmentDialog {

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

    /**
     * Compose this in six parts: (1) a top part consisting of the title, date,
     * end date, due time, and duration components; (2) a part consisting of
     * recurring info components; (3) a middle part with category, 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 a Task");

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

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

    /**
     * Compose the start date row using two pairs of labels and text fields.
     */
    protected Box composeStartDateRow() {

        Box hbox = Box.createHorizontalBox();

        /*
         * Construct the labels and text fields.  See internal comments in the
         * composeTitle method for further explanatory details.
         */
        startDateLabel = new JLabel("Due Date:  ");
        startDateLabel.setForeground(Color.black);
        startDateTextField = new JTextField(15);
        startDateTextField.setMaximumSize(
            new Dimension(maxComponentWidth, (int)(maxComponentHeight *
                startDateTextField.getFont().getSize())));
        JLabel startTimeLabel = new JLabel("Due Time:  ");
        startTimeLabel.setForeground(Color.black);
        startTimeTextField = new JTextField(15);
        startTimeTextField.setMaximumSize(
            new Dimension(maxComponentWidth, (int)(maxComponentHeight *
                startTimeTextField.getFont().getSize())));

        /*
         * Add them to the hbox and return it.
         */
        hbox.add(Box.createHorizontalStrut(15));
        hbox.add(startDateLabel);
        hbox.add(startDateTextField);
        hbox.add(Box.createHorizontalStrut(10));
        hbox.add(startTimeLabel);
```

```
        hbox.add(startTimeTextField);
        hbox.add(Box.createHorizontalStrut(15));
        return hbox;

    }

    /**
     * Compose the end date row as a label/textField pair.
     */
    protected Box composeEndDateRow() {

        Box hbox = Box.createHorizontalBox();

        /*
         * Construct the label and text field.  See internal comments in the
         * composeTitle method for further explanatory details.
         */
        endDateLabel = new JLabel("End Date:  ");
        endDateLabel.setForeground(Color.black);
        endDateLabel.setEnabled(false);

        endDateTextField = new JTextField(15);
        endDateTextField.setMaximumSize(
            new Dimension(maxComponentWidth, (int)(maxComponentHeight *
                endDateTextField.getFont().getSize())));
        endDateTextField.setEnabled(false);

        /*
         * Force decent-looking layout with a fixed-size horizontal strut.
         * There's got to be a better way to do this, such as using the width
         * of the Duration component, but so far the way has eluded me.
         */
        int blankSpacing = 254;

        /*
         * Add them to the hbox and return it.
         */
        hbox.add(Box.createHorizontalStrut(15));
        hbox.add(endDateLabel);
        hbox.add(endDateTextField);
        hbox.add(Box.createHorizontalStrut(blankSpacing));
        return hbox;

    }

    /**
     * Compose the middle part of the dialog, consisting of the category,
     * location, security, and priority.  The category and security are combo
     * boxes, laid out in a horizontal box.  Location and priority are also
     * combo boxes in a horizontal box.  Location is editable.  See the
     * description of <a href= "ScheduleEventDialog#composeTopPart()">
     * composeTopPart for a more detailed description of component layout.
     */
    protected Box composeMiddlePart() {
        Box vbox = Box.createVerticalBox();

        vbox.add(composeCategorySecurityRow());
        vbox.add(Box.createVerticalStrut(15));
        vbox.add(composePriorityRow());

        return vbox;
    }

    /**
     * Compose the priority row using two pairs of labels and text
     * fields.
     */
    protected Box composePriorityRow() {
        Box hbox = Box.createHorizontalBox();

        /*
         * Force decent-looking layout with a fixed-size horizontal strut.
         * There's got to be a better way to do this, such as using the width
         * of the Duration component, but so far the way has eluded me.
         */
        int blankSpacing = 248;

        JLabel priorityLabel = new JLabel("Priority:  ");
        priorityLabel.setForeground(Color.black);

        String[] selections = {"0 (lowest)", "1", "2", "3", "4", "5", "6", "7",
                               "8", "9", "10 (highest)"};
        priorityComboBox = new JComboBox(selections);
        priorityComboBox.setMaximumSize(
            new Dimension(maxComponentWidth, (int)(maxComponentHeight *
                priorityComboBox.getFont().getSize())));

        hbox.add(Box.createHorizontalStrut(blankSpacing));
        hbox.add(priorityLabel);
        hbox.add(priorityComboBox);
        hbox.add(Box.createHorizontalStrut(15));

        return hbox;

    }

    /**
     * Compose the buttons row with three JButtons.  The action listeners for
     * Clear and Cancel buttons are straightforward.  The action listener for
     * the OK button is responsible for communication with the Schedule model.
     * See the description of <a href= "OKScheduleTaskButtonListener.html">
     * OKScheduleTaskButtonListener </a> for explanatory details.
     */
    protected Box composeButtonRow() {

        Box hbox = Box.createHorizontalBox();

        /*
         * Construct the three buttons.
         */
        JButton okButton = new JButton("OK");
        JButton clearButton = new JButton("Clear");
```

```java
        JButton cancelButton = new JButton("Cancel");

        /*
         * Attach the appropriate action listeners to each button.
         */
        okButton.addActionListener(
            new OKScheduleTaskButtonListener((Schedule) model, this));

        clearButton.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    clear();
                }
            }
        );

        cancelButton.addActionListener(
            new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    hide();
                }
            }
        );

        /*
         * Add them to the hbox and return it.
         */
        hbox.add(okButton);
        hbox.add(Box.createHorizontalStrut(30));
        hbox.add(clearButton);
        hbox.add(Box.createHorizontalStrut(30));
        hbox.add(cancelButton);
        return hbox;

    }

    /**
     * Clear each of the text fields of this to empty.  Reset the combo boxes
     * to no selection.  NOTE: This method needs to be refined to use default
     * values for clearing, once options and defaults functionality is
     * implemented.  It also needs to be refined to clear the recurring and
     * remind check boxes and associated components.
     */
    protected void clear() {
        titleTextField.setText("");
        startDateTextField.setText("");
        endDateTextField.setText("");
        startTimeTextField.setText("");
        categoryComboBox.setSelectedIndex(0);
        securityComboBox.setSelectedIndex(0);
        detailsTextArea.setText("");
    }

}
```