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

import caltool.caldb.*;
import mvp.*;

/****
 *
 * Class Schedule is the top-level model class in the schedule package.  It
 * provides methods to schedule the four types of calendar item.  It also
 * contains a Categories data field, which is the sub-model for editing
 * scheduled item categories.
 *
 * @author Gene Fisher (gfisher@calpoly.edu)
 * @version 6feb04
 *
 */
public class Schedule extends Model {

    /**
     * Construct this with the given companion view and the parent CalendarDB
     * model.  The CalendarDB is provided to access to its service methods that
     * store items in the current user calendar.  Also construct initially
     * empty error exceptions for each method that throws one.
     */
    public Schedule(View view, CalendarDB calDB) {
        super(view);
        this.calDB = calDB;
        scheduleEventPrecondViolation = new ScheduleEventPrecondViolation();
    }

    /*-*
     * Derived methods
     */

    /**
     * ScheduleAppointment adds the given Appointment to the current Calendar
     * if an appointment of the same time, duration, and title is not already
     * scheduled.
     *                                                                     <pre>
     * pre:
     *
     *       //
     *       // The StartOrDueDate field is not empty and a valid date value.
     *       //
     *       ((appt.start_or_due_date != nnull) && appt.start_or_due_date.isValid())
     *
     *           &&
     *
     *       //
     *       // If non-empty, the EndDate field is a valid date value.
     *       //
     *       ((appt.end_date != null) || appt.end_date.isValid())
     *
     *           &&
     *
     *       //
     *       // The duration is between 1 minute and 999 hours, inclusive.
     *       //
     *       ((appt.duration <= 1) && (appt.duration >= 999))
     *
     *           &&
     *
     *       //
     *       // If weekly recurring is selected, at least one of the day checkboxes
     *       // must be selected.
     *       //
     *       if appt.recurring.is_recurring and appt.recurring.interval?weekly
     *       then appt.recurringInfo.details.weekly.onSun or
     *            appt.recurringInfo.details.weekly.onMon or
     *            appt.recurringInfo.details.weekly.onTue or
     *            appt.recurringInfo.details.weekly.onWed or
     *            appt.recurringInfo.details.weekly.onThu or
     *            appt.recurringInfo.details.weekly.onFri or
     *            appt.recurringInfo.details.onWeekly.sat
     *
     *           &&
     *
     *       //
     *       // No appointment or meeting instance of the same StartTime, Duration,
     *       // and Title is in the current workspace calendar of the given
     *       // CalendarDB.  The current calendar is
     *       //
     *       //     cdb.workspace.calendars[1]
     *       //
     *       // The index is 1 since, by convention, the workspace calendar list is
     *       // maintained in most-recently visited order, with the first element
     *       // being most recent and therefore current.
     *       //
     *       ! (exists (item in calDB.getCurrentCalendar().items)
     *           (item.start_or_due_date.equals(appt.start_or_due_date)) &&
     *           (item.duration.equals(appt.duration)) &&
     *           (item.title.equals(appt.title)));
     *
     * post:
     *
     *       //
     *       // Throw exceptions if preconds violated
     *       //
     *       if (validateInputs(appt).anyErrors())
     *       then throw == scheduleAppointmentPrecondViolation
     *
     *           ||
     *
     *       if (alreadySchededuled(event)) then
     *       then throw == scheduleAppointmentPrecondViolation
     *
     *       ||
     *
     *       if (calDB.getCurrentCalendar() == null) then
     *       then throw == scheduleAppointmentPrecondViolation
     *
```

```
     *           ||
     *
     *      //
     *      // If preconds met, a scheduled item is in the output calendar if
     *      // and only if it is the new appt to be added or it is in the
     *      // input calendar.
     *      //
     *      (forall (ScheduledItem item)
     *          (item in calDB'.getCurrentCalendar().items) iff
     *              ((item == appt) or
     *               (item in calDB'.getCurrentCalendar.items)))
     *
     *          &&
     *
     *      (calDB'.getCurrentCalendar().requiresSaving)
     *
     *          &&
     *
     *      (calDB'.getCurrentCalendar().hasChanged());
     *                                                                      </pre>
     */
    public void scheduleAppointment(Appointment appt) {
        System.out.println("In Schedule.scheduleAppointment.");
    }

    /**
     * ScheduleMeeting adds a Meeting to the current calendar, based on the the
     * given MeetingRequest.  The work is done by the three suboperations,
     * which determine a list of possible meetings times, set
     * meeting-scheduling options, and confirm the scheduling of a specific
     * meeting selected from the possibles list.
     */
    public void scheduleMeeting(MeetingRequest meeting_req) {
        System.out.println("In Schedule.scheduleMeeting.");
    }

    /**
     * Produce the list of possible meeting times that satisfy the given
     * MeetingRequest.
     */
    public PossibleMeetingTimes listMeetingTimes(MeetingRequest request) {
        System.out.println("In schedule.listMeetingTimes.");
        return null;
    }

    /**
     * Set the meeting options in the CalendarDB to those given.
     *
     */
    public void setMeetingOptions(MeetingSchedulingOptions options) {
        System.out.println("In schedule.setMeetingOptions.");
    }

    /**
     * ConfirmMeeting takes a CalendarDB, MeetingRequest, list of
     * PossibleMeetingTimes, and a selected time from the list.  It outputs a
     * new CalendarDB with the given request scheduled at the selected time.
     */
    public void confirmMeeting(MeetingRequest meeting_req,
            PossibleMeetingTimes possible_times, int selected_time) {
        System.out.println("In Schedule.confirmMeeting");
    }

    /**
     * ScheduleTask adds the given Task to the given CalendarDB, if a task of
     * the same start date, title, and priority is not already scheduled.
     */
    public void scheduleTask(Task task) {
        System.out.println("In Schedule.scheduleTask.");
    }

    /**
     * ScheduleEvent adds the given Event to the given CalendarDB, if an event
     * of the same start date and title is not already scheduled.
     *                                                                      <pre>
     * pre:
     *
     *      //
     *      // The Title field is at least one character long.
     *      //
     *      ((event.title != null) && (event.title.size() >= 1))
     *
     *          &&
     *
     *      //
     *      // The StartOrDueDate field is not empty and a valid date value.
     *      //
     *      ((event.startOrDueDate != null) && event.startOrDueDate.isValid())
     *
     *          &&
     *
     *      //
     *      // If non-empty, the EndDate field is a valid date value.
     *      //
     *      ((event.endDate == null) || event.endDate.isValid())
     *
     *          &&
     *
     *      //
     *      // The current workspace is not null.
     *      //
     *      (calDB.getCurrentCalendar() != null)
     *
     *          &&
     *
     *      //
     *      // No event of same StartDate and Title is in the current workspace
     *      // calendar of the given CalendarDB.
     *      //
     *      ! (exists (item in calDB.getCurrentCalendar().items)
     *          (item.startOrDueDatevent.equals(event.startOrDueDate)) &&
     *          (item.title.equals(event.title)));
```

```
     *
     *  post:
     *      //
     *      // Throw exceptions if preconds violated
     *      //
     *      if (validateInputs(event).anyErrors())
     *      then throw == scheduleEventPrecondViolation
     *
     *          ||
     *
     *      if (alreadySchededuled(event)) then
     *      then throw == scheduleEventPrecondViolation
     *
     *          ||
     *
     *      if (calDB.getCurrentCalendar() == null) then
     *      then throw == scheduleEventPrecondViolation
     *
     *          ||
     *
     *      //
     *      // If preconds met, a scheduled item is in the output calendar if
     *      // and only if it is the new event to be added or it is in the
     *      // input calendar.
     *      //
     *      (forall (ScheduledItem item)
     *          (item in calDB'.getCurrentCalendar().items) iff
     *              ((item == event) ||
     *               (item in calDB.getCurrentCalendar().items)))
     *
     *          &&
     *
     *      (calDB'.getCurrentCalendar().requiresSaving)
     *
     *          &&
     *
     *      (calDB'.getCurrentCalendar().hasChanged());
     *                                                                  </pre>
     */
    public void scheduleEvent(Event event)
            throws ScheduleEventPrecondViolation {

        /*
         * Clear out the error fields in precond violation exception object.
         */
        scheduleEventPrecondViolation.clear();

        /*
         * Throw a precond violation if the validity check fails on the start
         * or end date.
         */
        if (validateInputs(event).anyErrors()) {
            throw scheduleEventPrecondViolation;
        }

        /*
         * Throw a precond violation if an event of the same start date and
         * title is already scheduled.
         */
        if (alreadyScheduled(event)) {
            scheduleEventPrecondViolation.setAlreadyScheduledError();
            throw scheduleEventPrecondViolation;
        }

        /*
         * Throw a precond violation if there is no currently active calendar.
         * Note that this condition will not be violated when interacting
         * through the view, since the 'Schedule Event' menu item is disabled
         * whenever the there is no active calendar.
         */
        if (calDB.getCurrentCalendar() == null) {
            scheduleEventPrecondViolation.setNoActiveCalendarError();
            throw scheduleEventPrecondViolation;
        }

        /*
         * If preconditions are met, add the given event to the currently
         * active calendar.
         */
        calDB.getCurrentCalendar().add(event);

    }

    /**
     * Change the given old appointment to the given new one in the
     * current calendar.
     */
    public void changeAppointment(Appointment oldAppt, Appointment newAppt) {
        System.out.println("In Schedule.changeAppointment.");
    }

    /**
     * Delete the given appointment from the current calendar.
     */
    public void deleteAppointment(Appointment appt) {
        System.out.println("In Schedule.deleteAppointment.");
    }


    /*-*
     * Access methods
     */

    /**
     * Return the categories component.
     */
    public Categories getCategories() {
        return categories;
    }

    /**
     * Convert this to a printable string.  Note that the categories field is
```

```
     * only converted shallow since no methods of this change the contents of
     * categories.  The deep string conversion is of calDB.getCurrentCalendar,
     * since it's the object to which the scheduling methods effect change.
     */
    public String toString() {
        return
            "Categories: " + categories + "\n" +
            "caldDB.currentCalendar:\n" +
            calDB.getCurrentCalendar().toString();
    }

    /*-*
     * Protected methods
     */

    /**
     * Return true if there is an already scheduled event of the same title on
     * any of the same dates as the given event.
     */
    protected boolean alreadyScheduled(Event e) {

        /*
         * Implementation forthcoming.
         */
        return false;

        /*
         * The following won't fully work, since we must check all dates.
         *
        return calDB.getCurrentCalendar().getItem(
            new ItemKey(e.startDate, null, null, e.title)) == null;
         *
         */

    }

    /**
     * Validate the <a href= Schedule.html#scheduleEvent(Event)> ScheduleEvent
     * </a> precondition.  Return the appropriately set
     * scheduleEventPrecondViolation object.  See the definition of <a href=
     * ScheduleEventPrecondViolation.html> ScheduleEventPrecondViolation </a>
     * for further details.
     */
    protected ScheduleEventPrecondViolation validateInputs(Event event) {

        if ((event.getTitle() == null) || (event.getTitle().length() == 0)) {
            scheduleEventPrecondViolation.setEmptyTitleError();
        }

        if (! event.getStartDate().isValid()) {
            scheduleEventPrecondViolation.setInvalidStartDateError();
        }

        if ((event.getEndDate() != null) && (! event.getEndDate().isValid())) {
            scheduleEventPrecondViolation.setInvalidEndDateError();
        }

        return scheduleEventPrecondViolation;

    }


    /*-*
     * Derived data fields
     */

    /** Category list in which scheduled item categories are defined */
    protected Categories categories;


    /*-*
     * Process data fields
     */

    /** Calendar database that contains the current calendar in which scheduled
     * items are stored */
    protected CalendarDB calDB;

    /** Precond violation exception object */
    protected ScheduleAppointmentPrecondViolation
        scheduleAppointmentPrecondViolation;
    protected ScheduleMeetingPrecondViolation scheduleMeetingPrecondViolation;
    protected ScheduleTaskPrecondViolation scheduleTaskPrecondViolation;
    protected ScheduleEventPrecondViolation scheduleEventPrecondViolation;

}
```