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

import caltool.schedule.*;
import caltool.options.*;
import mvp.*;
import java.util.*;

/****
 *
 * The main data components of a user UserCalendar are a collection of
 * scheduled items and calendar-specific settings.  Calendar bookkeeping
 * components are the ID of the user who owns the calendar, the file it's
 * stored on, the currently selected date, and a flag indicating if the
 * calendar requires saving.
 *                                                                          <p>
 * In the current design, the concrete representation of the scheduled item
 * list is a TreeMap.  UserCalendar provides a getItem method to look up a
 * scheduled item by its unique key.  Based on the specs, the unique key for
 * each type of item is as follows:
 *                                                                        <pre>
 *     Item             Unique Key
 *     ========================================================================
 *     Appointment      {date, start time, duration, title}
 *     Meeting          {date, start time, duration, title}
 *     Task             {date, time, title, priority}
 *     Event            {date, title}
 *                                                                      </pre>
 * UserCalendar also provides an array-valued getItems method to retrieve all
 * of the items that are scheduled in a specified interval of date/time.  This
 * method is used by the caltool viewing methods to access the scheduled items
 * for a given day, week, or month.
 *                                                                          <p>
 * UserCalendar provides general-purpose methods to support the higher-level
 * model classes in the schedule and view packages.  The general-purpose
 * methods of UserCalendar do no input validity checking, assuming it has been
 * performed by the higher-level model methods.
 *
 */
public class UserCalendar extends Model {

    /*-*
     * Public methods.
     */

    /**
     * Construct this by constructing and initializing all components.
     */
    public UserCalendar(String uid) {
        items = new TreeMap();
        settings = null;
        this.uid = uid;
        file = null;
        selectedDate = null;
        requiresSaving = false;
        selectedItem = null;

        /*
         * For initial testing purposes, construct a fixed item to use as the
         * currently selected item.
         */
        selectedItem = new Appointment(
                "Dentist",                      // Title
                new caltool.schedule.Date(      // Date
                    "September 25, 1998"),
                null,                           // End Date
                new Time("8 AM"),               // Time
                new Duration(1, 30),            // Duration
                null,                           // Recurring info
                new Category("personal"),       // Category
                "1342 Sycamore Dr",             // Location
                Security.PublicTitle,           // Security
                Priority.Must,                  // Priority
                new RemindInfo(true,            // Remind info
                    new RemindWhen(1,
                        ReminderTimeUnit.DaysBefore),
                    RemindWhere.OnScreen),
                ""                              // Details
        );
    }

    /**
     * Add the given item to this.items.  Note that this method has no
1    * precondition.  All the validity and no-duplication requirements for the
     * given item are checked at the level of the Schedule model.
     *                                                                      <pre>
     * pre: ;
     *
     * post:
     *       //
     *       // The input item is added to items via items.put, which means
     *       // that item is added if an item of the same key is not already
     *       // there.  This is marked as changed via Observable.setChanged().
     *       //
     *       (items' == items.put(item.getKey(), item))
     *
     *           &&
     *
     *       this'.hasChanged();
     *                                                                    </pre>
     */
    public void add(ScheduledItem item) {

        /*
         * Put the given item into the items map with its generated unique key.
         */
        items.put(item.getKey(), item);

        /*
         * Indicate that this has changed in case anyone is observing.  The
         * setChanged method is inherited from Model, which in turn inherits
         * them from Observable.
```

```
112          */
113         setChanged();
114
115     }
116
117     /**
118      * Delete the given item from this.items.  Note that this method has no
119      * precondition.  All the validity and no-duplication requirements for the
120      * given item are checked at the level of the Schedule model.
121      *                                                                    <pre>
122      * pre: ;
123      *
124      * post:
125      *       //
126      *       // The input item is added to items via HashMap.put, which means
127      *       // that item is added if an item of the same key is not already
128      *       // there.  This is marked as changed via Observable.setChanged().
129      *       //
130      *       (items' == items.remove(item.getKey(), item))
131      *
132      *           &&
133      *
134      *       this'.hasChanged();
135      *                                                                 </pre>
136      */
137     public void delete(ScheduledItem item) {
138
139         items.remove(item);
140
141         /*
142          * Indicate that this has changed in case anyone is observing.  The
143          * setChanged method is inherited from Model, which in turn inherits
144          * them from Observable.
145          */
146         setChanged();
147     }
148
149     /**
150      * Return the scheduled item of the given unique key.
151      *                                                                    <pre>
152      * pre: ;
153      *
154      * post:
155      *       //
156      *       // If there is an item with the given key in this.items, then the
157      *       // return value is that item, otherwise the return is null.
158      *       //
159      *       (exists (item in items) (item.getKey().equals(key)) &&
160      *                               (return == item)
161      *           ||
162      *
163      *       (return == null);
164      *                                                                 </pre>
165      */
166     public ScheduledItem getItem(ItemKey key) {
167         return (ScheduledItem) items.get((Object) key);
168     }
169
170     /**
171      * Return an array of items in the given date range.  The start date must
172      * be <= the end date.
173      */
174     public ScheduledItem[] getItems(caltool.schedule.Date startDate,
175             caltool.schedule.Date endDate) {
176
177         /*
178          * Implementation forthcoming.
179          */
180
181         return null;
182     }
183
184     /**
185      * Return the previous item in item-key order after the item with the given
186      * key.  Return null if the given key is that of the first item
187      */
188     public ScheduledItem getPrev(ItemKey key) {
189
190         try {
191             return (ScheduledItem) items.get(items.headMap(key).lastKey());
192         }
193         catch (NoSuchElementException e) {
194             return null;
195         }
196
197     }
198
199     /**
200      * Return the next item in item-key order after the item with the given
201      * key.  Return null if the given key is that of the last item
202      */
203     public ScheduledItem getNextItem(ItemKey key) {
204
205         Iterator it = items.tailMap(key).keySet().iterator();
206         if (! it.hasNext()) {
207             return null;
208         }
209         it.next();
210         if (it.hasNext()) {
211             return (ScheduledItem) items.get(it.next());
212         }
213         else {
214             return null;
215         }
216
217     }
218
219     /**
220      * Return the user id of this calendar.
221      */
222     String getUid() {
223         return uid;
```

```
224     }
225
226     /**
227      * Return the file on which this calendar was most recently stored.
228      */
229     java.io.File getFile() {
230         return file;
231     }
232
233     /**
234      * Set the file on which this calendar is currently stored.
235      */
236     void setFile(java.io.File file) {
237         this.file = file;
238     }
239
240     /**
241      * Get the calendar-specific settings for this calendar.
242      */
243     CalendarSpecificSettings getSettings() {
244         return settings;
245     }
246
247     /**
248      * Return the date most recently selected by the user via clicking in some
249      * view.
250      */
251     public caltool.schedule.Date getSelectedDate() {
252         return selectedDate;
253     }
254
255     /**
256      * Set the currently selected date to the given date.
257      */
258     public void setSelectedDate(caltool.schedule.Date date) {
259         selectedDate = date;
260     }
261
262     /**
263      * Return the item most recently selected by the user via clicking in some
264      * view.
265      */
266     public ScheduledItem getSelectedItem() {
267         return selectedItem;
268     }
269
270     /**
271      * Set the currently selected date to the given date.
272      */
273     public void setSelectedItem(ScheduledItem item) {
274         selectedItem = item;
275     }
276
277     /**
278      * Convert this to a printable string.  The items are dumped last, since
279      * there may be a lot of them.  Note that the settings field is only
280      * printed shallow since no methods of this change the contents of
281      * settings.
282      */
283     public String toString() {
284         return
285             "User id: " + uid + "\n" +
286             "File: " + (file == null ? "null" : file.toString()) + "\n" +
287             "Selected date: " +
288                 (selectedDate == null ? "null" : selectedDate.toString()) +
289                     "\n" +
290             "Requires saving: " + String.valueOf(requiresSaving) + "\n" +
291             "Selected item: " +
292                 (selectedItem == null ? "null" : selectedItem.toString()) +
293                     "\n" +
294             items.size() + " Items: \n" + items.toString() + "\n" +
295             settings + "\n";
296     }
297
298     /**
299      * Return the number of items in this.items, for testing purposes.
300      */
301     public int numItems() {
302         return items.size();
303     }
304
305     /*-*
306      * Derived data fields
307      */
308
309     /** The collection of all scheduled items for this calendar */
310     TreeMap items;
311
312     /** Calendar-specific settings for this calendar */
313     CalendarSpecificSettings settings;
314
315     /** Id of user who owns this calendar */
316     String uid;
317
318     /** File this calendar is stored on */
319     java.io.File file;
320
321     /** Currently selected date, if any */
322     caltool.schedule.Date selectedDate;
323
324     /** True if this requires saving */
325     boolean requiresSaving;
326
327
328     /*-*
329      * Additional data fields
330      */
331
332     /** Currently selected item, if any */
333     ScheduledItem selectedItem;
334
335 }
```