

```

Loading vc-cvs...
1 package caltool.caldb;
2
3 import caltool.schedule.*;
4 import caltool.options.*;
5 import mvp.*;
6 import java.util.*;
7
8 /****
9 *
10 * The main data components of a user UserCalendar are a collection of
11 * scheduled items and calendar-specific settings. Calendar bookkeeping
12 * components are the ID of the user who owns the calendar, the file it's
13 * stored on, the currently selected date, and a flag indicating if the
14 * calendar requires saving.
15 *
16 * In the current design, the concrete representation of the scheduled item
17 * list is a TreeMap. UserCalendar provides a getItem method to look up a
18 * scheduled item by its unique key. Based on the specs, the unique key for
19 * each type of item is as follows:
20 *
21 * Item Unique Key
22 * =====
23 * Appointment {date, start time, duration, title}
24 * Meeting {date, start time, duration, title}
25 * Task {date, time, title, priority}
26 * Event {date, title}
27 *
28 * UserCalendar also provides an array-valued getItems method to retrieve all
29 * of the items that are scheduled in a specified interval of date/time. This
30 * method is used by the caltool viewing methods to access the scheduled items
31 * for a given day, week, or month.
32 *
33 * UserCalendar provides general-purpose methods to support the higher-level
34 * model classes in the schedule and view packages. The general-purpose
35 * methods of UserCalendar do no input validity checking, assuming it has been
36 * performed by the higher-level model methods.
37 *
38 */
39 public class UserCalendar extends Model {
40
41     /*-
42     * Public methods.
43     */
44
45     /**
46     * Construct this by constructing and initializing all components.
47     */
48     public UserCalendar(String uid) {
49         items = new TreeMap();
50         settings = null;
51         this.uid = uid;
52         file = null;
53         selectedDate = null;
54         requiresSaving = false;
55         selectedItem = null;
56
57         /*
58         * For initial testing purposes, construct a fixed item to use as the
59         * currently selected item.
60         */
61         selectedItem = new Appointment(
62             "Dentist", // Title
63             new caltool.schedule.Date( // Date
64                 "September 25, 1998"),
65             null, // End Date
66             new Time("8 AM"), // Time
67             new Duration(1, 30), // Duration
68             null, // Recurring info
69             new Category("personal"), // Category
70             "1342 Sycamore Dr", // Location
71             Security.PublicTitle, // Security
72             Priority.Must, // Priority
73             new RemindInfo(true, // Remind info
74                 new RemindWhen(1,
75                     ReminderTimeUnit.DaysBefore),
76                 RemindWhere.OnScreen),
77             "" // Details
78         );
79     }
80
81     /**
82     * Add the given item to this.items. Note that this method has no
83     * precondition. All the validity and no-duplication requirements for the
84     * given item are checked at the level of the Schedule model.
85     *
86     * pre: ;
87     *
88     * post:
89     * //
90     * // The input item is added to items via items.put, which means
91     * // that item is added if an item of the same key is not already
92     * // there. This is marked as changed via Observable.setChanged().
93     * //
94     * (items' == items.put(item.getKey(), item))
95     *
96     * &&
97     *
98     * this'.hasChanged();
99     *
100     */
101     public void add(ScheduledItem item) {
102
103         /*
104         * Put the given item into the items map with its generated unique key.
105         */
106         items.put(item.getKey(), item);
107
108         /*
109         * Indicate that this has changed in case anyone is observing. The
110         * setChanged method is inherited from Model, which in turn inherits
111         * 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  *
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  *
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  *
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  *
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  * Return the next item in item-key order after the item with the given
200  * key. Return null if the given key is that of the last item
201  */
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  * Return the user id of this calendar.
220  */
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
281     * printed shallow since no methods of this change the contents of
282     * settings.
283     */
284     public String toString() {
285         return
286             "User id: " + uid + "\n" +
287             "File: " + (file == null ? "null" : file.toString()) + "\n" +
288             "Selected date: " +
289                 (selectedDate == null ? "null" : selectedDate.toString()) +
290                 "\n" +
291             "Requires saving: " + String.valueOf(requiresSaving) + "\n" +
292             "Selected item: " +
293                 (selectedItem == null ? "null" : selectedItem.toString()) +
294                 "\n" +
295             items.size() + " Items: \n" + items.toString() + "\n" +
296             settings + "\n";
297     }
298
299     /**
300     * Return the number of items in this.items, for testing purposes.
301     */
302     public int numItems() {
303         return items.size();
304     }
305
306     /**-
307     * Derived data fields
308     */
309
310     /** The collection of all scheduled items for this calendar */
311     TreeMap items;
312
313     /** Calendar-specific settings for this calendar */
314     CalendarSpecificSettings settings;
315
316     /** Id of user who owns this calendar */
317     String uid;
318
319     /** File this calendar is stored on */
320     java.io.File file;
321
322     /** Currently selected date, if any */
323     caltool.schedule.Date selectedDate;
324
325     /** True if this requires saving */
326     boolean requiresSaving;
327
328     /**-
329     * Additional data fields
330     */
331
332     /** Currently selected item, if any */
333     ScheduledItem selectedItem;
334
335 }

```

