```
1  package rolodex;
2
3  import mvp.*;
4
5  /****
6   *
7   * Class RolodexStub is the model class for the basic rolodex object, including
8   * the edit operations to add, delete, change, and find rolodex entries.
9   *
10  * RolodexStub is a stubbed implementation, for GUI testing purposes.  Cf.
11  * <a href="Rolodex.html"> Rolodex.java </a> for the complete implementation.
12  *
13  */
14
15 public class RolodexStub extends Model {
16
17     /**
18      * Construct an initially empty rolodex.  Initialize the error objects for
19      * each function that throws an exception.
20      *                                                                    <pre>
21      * post: (exists (CardList cl) (this'.cl == cl) and (cl.Len() == 0))
22      *           and
23      *       (exists (AddInputErrors aie) (this'.aie == aie));
24      *           and
25      *       (exists (DeleteInputError die) (this'.die == die));
26      *           and
27      *       (exists (ChangeInputErrors cie) (this'.cie == cie));
28      *                                                                  </pre>
29      */
30     public RolodexStub(View v) {
31         super(v);
32     }
33
34     /**
35      * Add the given card to this if a card of the same id is not already there
36      * and there is room for another card.
37      *                                                                    <pre>
38      * pre:
39      *      // There is no card in the input this with the same id as the given
40      *      // input card.
41      *      not CardAlreadyThere(c)
42      *
43      *          and
44      *
45      *      // The values of the given card fields are valid.
46      *      CardIsValid(c)
47      *
48      *          and
49      *
50      *      // There is room for another card.
51      *      (cl.Len() < MAXCARDS)
52      *
53      * post:
54      *      // If none of the preconditions is violated, then a normal
55      *      // postcondition holds, else an exception is thrown.
56      *      if (not AddPrecondsViolated(c)) (
57      *          // A card is in the output this iff it is in the input this or
58      *          // it's the new card to be added.
59      *          forall (Card c') this'.In(c') iff (In(c') or (c' == c))
60      *
61      *              and
62      *
63      *          // The size of this is increased by 1.
64      *          (cl'.Length() == cl.Length() + 1)
65      *      )
66      *      else (
67      *          // An AddInputException is thrown if any of the preconditions
68      *          // is violated.  The value of the exception flag is set for
69      *          // each violated precondition via the ValidataAddInput
70      *          // function.
71      *          (throw == aie) and (aie == ValidateAddInput(c));
72      *                                                                  </pre>
73      */
74     public void add(Card c) {
75         System.out.println("In Rolodex.add");
76     }
77
78     /**
79      * Delete the card of the given id from this.
80      *                                                                    <pre>
81      * pre:
82      *      // There is a card of the given id in the input this.
83      *      exists (Card c' | In(c')) c'.getId() == id;
84      *
85      * post:
86      *      // If the precondition is not violated, then a normal postcondition
87      *      // holds, else an exception is thrown.
88      *      if (DeletePrecondViolated(c)) then (
89      *          // A card is in the output this iff it is in the input this and
90      *          // its id is different than the given id.
91      *          forall (Card c')
92      *              this'.In(c') iff (In(c') and (c'.id != id));
93      *      )
94      *      else
95      *          // A DeleteInputException is thrown if the precondition is
96      *          // violated.
97      *          throw == die;
98      *                                                                  </pre>
99      */
100    public void delete(Id id) {
101        System.out.println("In Rolodex.delete");
102    }
103
104    /**
105     * Change the card of the given id in this to the given new card.
106     *
107     *                                                                    <pre>
108     * pre:
109     *      // There is a card of the given id in the input this and that card
110     *      // is not identical with the given input card, and the given card
111     *      // is valid.
112     *      exists (Card c' | In(c') (c'.getId() == id) and
```

```
113      *          (c' != c) and CardIsValid(c);
114      *
115      * post:
116      *      // If the precondition is not violated, then a normal postcondition
117      *      // holds, else an exception is thrown.
118      *      if (ChangePrecondViolated(c)) then (
119      *          // A card is in the output this iff it is in the input this and
120      *          // is not the card with the given id in the input this, or it's
121      *          // the given input card.
122      *          forall (Card c')
123      *              this'.In(c') iff (In(c') and
124      *                  (not exists (Card c'' | In(c''))
125      *                      (c''.getId() == id) and (c'' == c')) or
126      *                  (c' == c))
127      *      )
128      *      else
129      *          // A ChangeInputException is thrown if the precondition is
130      *          // violated.
131      *          throw == cie;
132      *                                                                  </pre>
133      */
134     public void change(Id id, Card c) {
135         System.out.println("In Rolodex.change");
136     }
137
138     /**
139      * Find all of the cards of the given name in this.
140      *                                                                  <pre>
141      * post:
142      *      // Cards in the output list consist of those in the input this with
143      *      // the given name.
144      *      forall (Card c)
145      *           (cl.In(c) iff (In(c) and (c.getName() == n))
146      *
147      *          and
148      *
149      *      // The output card list is sorted by ascending order of card id.
150      *      forall (int i, int j |
151      *          (i >= 1) and (i <= cl.Length()) and
152      *                  (j >= 1) and (j <= cl.Length()))
153      *              if i < j then
154      *                  cl.get(i).getId() < cl.get(j).getId();
155      *                                                                  </pre>
156      */
157     public CardList find(Name n) {
158         System.out.println("In Rolodex.find");
159         return null;
160     }
161
162     /**
163      * Print this to stdout.  In the current design, the specs are the same as
164      * Dump below, q.v.  In general, if there is a distinction between Print
165      * and Dump it is that the former produces a more nicely formated output
166      * than the latter.
167      */
168     public void print() {
169         System.out.println("In Rolodex.print");
170     }
171
172     /**
173      * Dump this to stdout by dumping each card, in List.iterator order.
174      */
175     public void dump() {
176         System.out.println("In Rolodex.dump");
177     }
178
179
180     /*-*
181      * View Service functions.  These are functions that provide data
182      * manipulation services to the views that the views cannot know how to
183      * implement.
184      */
185
186     /**
187      * Convert to the given string to one of the Sex enumeration literals.  The
188      * String value must be all lower case and only the first character is
189      * significant, i.e., "m*" or "f*".  If s is not one of these values,
190      * Female is returned.
191      *                                                                  <pre>
192      * post: if (s[1] == "m") then return == Male else return == Female
193      *                                                                  </pre>
194      */
195     public Sex stringToSex(String s) {return null;}
196
197     /**
198      * Convert to the given Sex enumeration literal to an lowercase string.
199      *                                                                  <pre>
200      * post: if (s == Mail) then return == "m" else return == "f"
201      *                                                                  </pre>
202      */
203     String sexToString(Sex s) {return null;}
204
205
206     /*-*
207      * Protected Functions
208      */
209
210     /**
211      * Validate the given card per the precondition of Add.  Return the
212      * appropriately set AddInputErrors value.  See the definition of
213      * AddInputErrors for further information.
214      */
215     protected AddInputErrors validateAddInput(Card c) {return null;}
216
217
218     /*-*
219      * Data Members
220      */
221
222     /** Simple list rep'n of the card list */
223     protected CardList cl;
224
```

```
225     /** Error messages for the Add function */
226     protected AddInputErrors aie;
227
228     /** Error message for the Delerte function */
229     protected DeleteInputError die;
230
231     /** Error messages for the Change function */
232     protected ChangeInputErrors cie;
233
234
235     /**
236      * Auxiliary spec functions.
237      *
238      * boolean cardAlreadyThere(Card c) {
239      *      not (exists (Card c' | In(c'))
240      *              c'.getId() == c.getId())
241      * }
242      *
243      * boolean cardIsValid(Card c) {
244      *      // The length of the name is <= 30 characters
245      *      (c.getName().Length() <= 30);
246      *
247      *          and
248      *
249      *      // The length (i.e, numer of digits) of the id is 9
250      *      c.getId().toString.Length() == 9)
251      *
252      *          and
253      *
254      *      // Age is a reasonable range
255      *      ((c.getAge() >= 0) and (c.getAge() <= 200))
256      *
257      *          and
258      *
259      *      // Sex is valid
260      *      (c.getSex() != InvalidSex)
261      *
262      *          and
263      *
264      *      // The length of the address is <= 40 chars
265      *      (c.getAddr() < 40)
266      *
267      * }
268      *
269      * boolean addPrecondsViolated(Card c) {
270      *     cardAlreadyThere(c) or (not cardIsValid(c));
271      * }
272      *
273      * boolean deletePrecondViolated(Card c) {
274      *     not (exists (Card c' | In(c')) c'.getId() == id)
275      * }
276      *
277      * boolean changePrecondViolated(Card c) {
278      *     not (exists (Card c' | In(c')) (c'.getId() == id) and
279      *                                         (c' != c));
280      * }
281      *
282      * boolean in(Card c);
283      * // Return true if the given card is anywhere in this.
284      *     Find(c) != null
285      *
286      */
287
288 };
```