```
package rolodex;

import java.util.*;
import java.io.*;
import mvp.*;

/****
 *
 * Class Rolodex is the model class for the basic rolodex object, including the
 * operations to add, delete, change, and find rolodex entries.
 *
 */

public class Rolodex extends Model implements Serializable {

    /*-*
     * Public Functions
     */

    /**
     * Construct an initially empty rolodex.  Initialize the error objects for
     * each function that throws an exception.
     *                                                                    <pre>
     * post: (exists (CardList cl) (this'.cl == cl) and (cl.Len() == 0))
     *           and
     *       (exists (AddInputErrors aie) (this'.aie == aie));
     *           and
     *       (exists (DeleteInputError die) (this'.die == die));
     *           and
     *       (exists (ChangeInputErrors cie) (this'.cie == cie));
     *                                                                  </pre>
     */
    public Rolodex(View v) {
        super(v);
        cl = new CardList();
        aie = new AddInputErrors();
        die = new DeleteInputError();
        cie = new ChangeInputErrors();
    }

    /**
     * Add the given card to this if a card of the same id is not already there
     * and there is room for another card.
     *                                                                    <pre>
     * pre:
     *      // There is no card in the input this with the same id as the given
     *      // input card.
     *      not CardAlreadyThere(c)
     *
     *          and
     *
     *      // The values of the given card fields are valid.
     *      CardIsValid(c)
     *
     *          and
     *
     *      // There is room for another card.
     *      (cl.Len() < MAXCARDS)
     *
     * post:
     *      // If none of the preconditions is violated, then a normal
     *      // postcondition holds, else an exception is thrown.
     *      if (not addPrecondViolated(c)) (
     *          // A card is in the output this iff it is in the input this or
     *          // it's the new card to be added.
     *          forall (Card c') this'.In(c') iff (In(c') or (c' == c))
     *
     *              and
     *
     *          // The size of this is increased by 1.
     *          (cl'.Length() == cl.Length() + 1)
     *
     *              and
     *
     *          // This has changed (as an observable object).
     *          this'.hasChanged() == true;
     *      )
     *      else (
     *          // An AddInputException is thrown if any of the preconditions
     *          // is violated.  The value of the exception flag is set for
     *          // each violated precondition via the ValidataAddInput
     *          // function.
     *          (throw == aie) and (aie == ValidateAddInput(c))
     *
     *          and
     *
     *                                                                  </pre>
     */
    public void add(Card c) throws AddInputErrors {

        /*
         * Clear out the input error data.
         */
        aie.clear();

        /*
         * Throw a precond violation if a card of the given id is already in
         * this.cl.
         */
        if (find(c.getId()) != null) {
            aie.setAlreadyThereError();
            throw aie;
        }

        /*
         * Throw a precond violation if the validy check fails for one or more
         * of the card fields.  The exception value is a tuple of values for
         * each of the invalid fields.
         */
        if (validateAddInput(c).anyErrors().booleanValue()) {
            throw aie;
        }
```

```
113
114         /*
115          * If preconditions are met, add the given card to this.cl using the
116          * inherited CardList.add function.
117          */
118         cl.add(c);
119
120         /*
121          * Indicate that this has changed, in case anyone's observing.  The
122          * setChanged method is inherited from Model, which in turn inherits it
123          * from Observable.
124          */
125         setChanged();
126
127     }
128
129     /**
130      * Delete the card of the given id from this.
131      *                                                                  <pre>
132      * pre:
133      *      // There is a card of the given id in the input this.
134      *      exists (Card c' | In(c')) c'.getId() == id;
135      *
136      * post:
137      *      // If the precondition is not violated, then a normal postcondition
138      *      // holds, else an exception is thrown.
139      *      if (deletePrecondViolated(c)) then (
140      *          // A card is in the output this iff it is in the input this and
141      *          // its id is different than the given id.
142      *          forall (Card c')
143      *              this'.In(c') iff (In(c') and (c'.id != id));
144      *
145      *              and
146      *
147      *          // This has changed (as an observable object).
148      *          this'.hasChanged() == true;
149      *      )
150      *      else
151      *          // A DeleteInputException is thrown if the precondition is
152      *          // violated.
153      *          throw == die;
154      *                                                                  </pre>
155      */
156     public void delete(Id id) throws DeleteInputError {
157
158         /*
159          * Clear out the input error data.
160          */
161         aie.clear();
162
163         /*
164          * Throw a precond violation if a card of the given id is not in
165          * this.cl.
166          */
167         if (find(id) == null) {
168             die.setNotThereErrorMessage();
169             throw die;
170         }
171
172         /*
173          * If the precondition is met, delete the given card from this.cl using
174          * the this.find(Id) and CardList.remove.
175          */
176         cl.remove(find(id));
177
178         /*
179          * Indicate that this has changed, in case anyone's observing.  The
180          * setChanged method is inherited from Model, which in turn inherits it
181          * from Observable.
182          */
183         setChanged();
184
185     }
186
187     /**
188      * Change the card of the given id in this to the given new card.
189      *
190      *                                                                  <pre>
191      * pre:
192      *      // There is a card of the given id in the input this and that card
193      *      // is not identical with the given input card, and the given card
194      *      // is valid.
195      *      exists (Card c' | In(c') (c'.getId() == id) and
196      *          (c' != c) and CardIsValid(c);
197      *
198      * post:
199      *      // If the precondition is not violated, then a normal postcondition
200      *      // holds, else an exception is thrown.
201      *      if (changePrecondViolated(c)) then (
202      *          // A card is in the output this iff it is in the input this and
203      *          // is not the card with the given id in the input this, or it's
204      *          // the given input card.
205      *          forall (Card c')
206      *              this'.In(c') iff (In(c') and
207      *                  (not exists (Card c'' | In(c''))
208      *                      (c''.getId() == id) and (c'' == c')) or
209      *                  (c' == c))
210      *
211      *              and
212      *
213      *          // This has changed (as an observable object).
214      *          this'.hasChanged() == true;
215      *      )
216      *      else
217      *          // A ChangeInputException is thrown if the precondition is
218      *          // violated.
219      *          throw == cie;
220      *                                                                  </pre>
221      */
222     public void change(Id id, Card c)
223             throws AddInputErrors, ChangeInputErrors, DeleteInputError {
224
```

```
        Card extant;                    // Extant rolodex card of given id

        /*
         * Clear out the errors in all three error lists, since any one could
         * be thrown.
         */
        aie.clear();
        die.clear();
        cie.clear();

        /*
         * Throw a precond violation if a card of the given id is not already
         * in this.cl.
         */
        if ((extant = find(id)) == null) {
            cie.setNotThereError();
            throw cie;
        }

        /*
         * Throw a precond violation if the given card is identical to the
         * extant card of the given id.
         */
        if (extant.equals(c)) {
            cie.setIdenticalToExtantError();
            throw cie;
        }

        /*
         * Throw a precond violation if the validy check fails for one or more
         * of the card fields.  The exception value is a tuple of values for
         * each of the invalid fields.
         */
        if (validateAddInput(c).anyErrors().booleanValue()) {
            throw aie;
        }

        /*
         * If preconditions are met, delete the extant card and add the new
         * one.  Note that exceptions are not caught here because we can
         * provide that preconds are met at this point.
         */
        delete(id);
        add(c);
    }

    /**
     * Find all of the cards of the given name in this.
     *                                                                  <pre>
     * post:
     *      // Cards in the output list consist of those in the input this with
     *      // the given name.
     *      forall (Card c)
     *           (cl.In(c) iff (In(c) and (c.getName() == n))
     *
     *          and
     *
     *      // The output card list is sorted by ascending order of card id.
     *      forall (int i, int j |
     *          (i >= 1) and (i <= cl.Length()) and
     *                  (j >= 1) and (j <= cl.Length()))
     *              if i < j then
     *                  cl.get(i).getId() < cl.get(j).getId();
     *                                                                  </pre>
     */
    public CardList find(Name n) {
        return cl.find(n);
    }

    /**
     * Provide an abstract iterator that will enumerate the cards of this.
     */
    public Iterator getIterator() {
        return cl.listIterator();
    }

    /**
     * Print this to stdout.  In the current design, the specs are the same as
     * dump below, q.v.  In general, if there is a distinction between print
     * and dump it is that the former produces a more nicely formated output
     * than the latter, and printing may be in some sorted order whereas
     * dumping will be in what ever the most convenient internal order is.
     */
    public void print() {
        dump();
    }

    /**
     * Dump this to stdout by dumping each card, in List.iterator order.
     */
    public void dump() {
        cl.dump();
    }

    /**
     * Find the card of the given id.  Return null of no such card is in this.
     */
    protected Card find(Id id) {
        return cl.find(id);
    }


    /*-*
     * Protected Functions
     */

    /**
     * Validate the given card per the precondition of Add.  Return the
     * appropriately set AddInputErrors value.  See the definition of
     * AddInputErrors for further information.
     */
    protected AddInputErrors validateAddInput(Card c) {
```

```
337         if (c.getName().stringValue().length() > 30)
338             aie.setNameError();
339
340         if (c.getId().integerValue() == null)
341             aie.setIdError();
342         else if ((c.getId().integerValue().intValue() < 100000000) ||
343                 (c.getId().integerValue().intValue() > 999999999))
344             aie.setIdError();
345
346         if (c.getAge().integerValue() == null)
347             aie.setAgeError();
348         else if ((c.getAge().integerValue().intValue() < 0) ||
349                 (c.getAge().integerValue().intValue() > 200))
350             aie.setAgeError();
351
352         if (!(c.getSex().isValid()))
353             aie.setSexError();
354
355         if (c.getAddr().stringValue().length() > 40)
356             aie.setAddressError();
357
358         return aie;
359     }
360
361
362     /*-*
363      * Data Members
364      */
365
366     /** Simple list rep'n of the card list */
367     protected CardList cl;
368
369     /** Error messages for the Add function */
370     protected AddInputErrors aie;
371
372     /** Error message for the Delerte function */
373     protected DeleteInputError die;
374
375     /** Error messages for the Change function */
376     protected ChangeInputErrors cie;
377
378
379     /**
380      * Auxiliary spec functions.
381      *
382      * boolean cardAlreadyThere(Card c) {
383      *      not (exists (Card c' | In(c'))
384      *              c'.getId() == c.getId())
385      * }
386      *
387      * boolean cardIsValid(Card c) {
388      *      // The length of the name is <= 30 characters
389      *      (c.getName().Length() <= 30);
390      *
391      *          and
392      *
393      *      // The length (i.e, numer of digits) of the id is 9
394      *      c.getId().toString.Length() == 9)
395      *
396      *          and
397      *
398      *      // Age is a reasonable range
399      *      ((c.getAge() >= 0) and (c.getAge() <= 200))
400      *
401      *          and
402      *
403      *      // Sex is valid
404      *      (c.getSex() != InvalidSex)
405      *
406      *          and
407      *
408      *      // The length of the address is <= 40 chars
409      *      (c.getAddr() < 40)
410      *
411      * }
412      *
413      * boolean addPrecondViolated(Card c) {
414      *     cardAlreadyThere(c) or (not cardIsValid(c));
415      * }
416      *
417      * boolean deletePrecondViolated(Card c) {
418      *     not (exists (Card c' | In(c')) c'.getId() == id)
419      * }
420      *
421      * boolean changePrecondViolated(Card c) {
422      *     not (exists (Card c' | In(c')) (c'.getId() == id) and
423      *                                          (c' != c));
424      * }
425      *
426      * boolean in(Card c);
427      * // Return true if the given card is anywhere in this.
428      *     Find(c) != null
429      *
430      */
431
432 };
```