File - D:\Users\gjr8050\262refactoring\src\Main\Lane.java

```
 1 package Main;
 2 /* $Id$
 3  *
 4  * Revisions:
 5  *   $Log: Lane.java,v $
 6  *   Revision 1.52  2003/02/20 20:27:45  ???
 7  *   Fouls disables.
 8  *
 9  *   Revision 1.51  2003/02/20 20:01:32  ???
10  *   Added things.
11  *
12  *   Revision 1.50  2003/02/20 19:53:52  ???
13  *   Added foul support.  Still need to update laneview and test this.
14  *
15  *   Revision 1.49  2003/02/20 11:18:22  ???
16  *   Works beautifully.
17  *
18  *   Revision 1.48  2003/02/20 04:10:58  ???
19  *   Score reporting code should be good.
20  *
21  *   Revision 1.47  2003/02/17 00:25:28  ???
22  *   Added disbale controls for View objects.
23  *
24  *   Revision 1.46  2003/02/17 00:20:47  ???
25  *   fix for event when game ends
26  *
27  *   Revision 1.43  2003/02/17 00:09:42  ???
28  *   fix for event when game ends
29  *
30  *   Revision 1.42  2003/02/17 00:03:34  ???
31  *   Bug fixed
32  *
33  *   Revision 1.41  2003/02/16 23:59:49  ???
34  *   Reporting of sorts.
35  *
36  *   Revision 1.40  2003/02/16 23:44:33  ???
37  *   added mechnanical problem flag
38  *
39  *   Revision 1.39  2003/02/16 23:43:08  ???
40  *   added mechnanical problem flag
41  *
42  *   Revision 1.38  2003/02/16 23:41:05  ???
43  *   added mechnanical problem flag
44  *
45  *   Revision 1.37  2003/02/16 23:00:26  ???
46  *   added mechnanical problem flag
47  *
48  *   Revision 1.36  2003/02/16 21:31:04  ???
49  *   Score logging.
50  *
51  *   Revision 1.35  2003/02/09 21:38:00  ???
52  *   Added lots of comments
53  *
54  *   Revision 1.34  2003/02/06 00:27:46  ???
55  *   Fixed a race condition
56  *
57  *   Revision 1.33  2003/02/05 11:16:34  ???
58  *   Boom-Shacka-Lacka!!!
59  *
60  *   Revision 1.32  2003/02/05 01:15:19  ???
```

```
 61  *   Real close now.  Honest.
 62  *
 63  *   Revision 1.31  2003/02/04 22:02:04  ???
 64  *   Still not quite working...
 65  *
 66  *   Revision 1.30  2003/02/04 13:33:04  ???
 67  *   Lane may very well work now.
 68  *
 69  *   Revision 1.29  2003/02/02 23:57:27  ???
 70  *   fix on pinsetter hack
 71  *
 72  *   Revision 1.28  2003/02/02 23:49:48  ???
 73  *   Pinsetter generates an event when all pins are reset
 74  *
 75  *   Revision 1.27  2003/02/02 23:26:32  ???
 76  *   Alley now runs its own thread and polls for free lanes to assign queue members to
 77  *
 78  *   Revision 1.26  2003/02/02 23:11:42  ???
 79  *   parties can now play more than 1 game on a lane, and lanes are properly released after
     games
 80  *
 81  *   Revision 1.25  2003/02/02 22:52:19  ???
 82  *   Lane compiles
 83  *
 84  *   Revision 1.24  2003/02/02 22:50:10  ???
 85  *   Lane compiles
 86  *
 87  *   Revision 1.23  2003/02/02 22:47:34  ???
 88  *   More observering.
 89  *
 90  *   Revision 1.22  2003/02/02 22:15:40  ???
 91  *   Add accessor for pinsetter.
 92  *
 93  *   Revision 1.21  2003/02/02 21:59:20  ???
 94  *   added conditions for the party choosing to play another game
 95  *
 96  *   Revision 1.20  2003/02/02 21:51:54  ???
 97  *   LaneEvent may very well be observer method.
 98  *
 99  *   Revision 1.19  2003/02/02 20:28:59  ???
100  *   fixed sleep thread bug in lane
101  *
102  *   Revision 1.18  2003/02/02 18:18:51  ???
103  *   more changes. just need to fix scoring.
104  *
105  *   Revision 1.17  2003/02/02 17:47:02  ???
106  *   Things are pretty close to working now...
107  *
108  *   Revision 1.16  2003/01/30 22:09:32  ???
109  *   Worked on scoring.
110  *
111  *   Revision 1.15  2003/01/30 21:45:08  ???
112  *   Fixed speling of received in Lane.
113  *
114  *   Revision 1.14  2003/01/30 21:29:30  ???
115  *   Fixed some MVC stuff
116  *
117  *   Revision 1.13  2003/01/30 03:45:26  ???
118  *   *** empty log message ***
119  *
```

```
120  *   Revision 1.12  2003/01/26 23:16:10  ???
121  *   Improved thread handeling in lane/controldesk
122  *
123  *   Revision 1.11  2003/01/26 22:34:44  ???
124  *   Total rewrite of lane and pinsetter for R2's observer model
125  *   Added Lane/Pinsetter Observer
126  *   Rewrite of scoring algorythm in lane
127  *
128  *   Revision 1.10  2003/01/26 20:44:05  ???
129  *   small changes
130  *
131  *
132  */
133
134 import Views.EndGamePrompt;
135 import Views.EndGameReport;
136
137 import java.lang.reflect.Array;
138 import java.text.SimpleDateFormat;
139 import java.util.*;
140
141 public class Lane extends Observable implements Observer {
142     private Thread laneThread;
143
144     private Vector<Bowler> party;
145     private Pinsetter pinsetter;
146
147     private boolean gameIsHalted;
148
149     private boolean partyAssigned;
150     private boolean gameFinished;
151     private Iterator<Bowler> bowlerIterator;
152     private int ball;
153     private int bowlIndex;
154     private int frameNumber;
155     private boolean tenthFrameStrike;
156
157     private boolean canThrowAgain;
158
159     private int[][] finalScores;
160     private int gameNumber;
161
162     private Bowler currentThrower;          // = the thrower who just took a throw
163
164     /** Lane()
165      *
166      * Constructs a new lane and starts its thread
167      *
168      * @pre none
169      * @post a new lane has been created and its thered is executing
170      */
171     public Lane() {
172         pinsetter = new Pinsetter();
173         gameIsHalted = false;
174         partyAssigned = false;
175
176         gameNumber = 0;
177
178         pinsetter.addObserver(this);
179
```

```
180         runThread();
181     }
182 
183     /** run()
184      *
185      * entry point for execution of this lane
186      */
187     public void runThread() {
188         laneThread = new Thread(() -> {
189                 while (true) {
190                     if (partyAssigned && !gameFinished) {    // we have a party on this
    lane,
191                         // so next bower can take a throw
192 
193                         while (gameIsHalted) {
194                             try {
195                                 Thread.sleep(10);
196                             } catch (Exception e) {
197                             }
198                         }
199 
200                         if (bowlerIterator.hasNext()) {
201                             currentThrower = bowlerIterator.next();
202 
203                             canThrowAgain = true;
204                             tenthFrameStrike = false;
205                             ball = 0;
206                             while (canThrowAgain) {
207                                 pinsetter.ballThrown();        // simulate the thrower's
    ball hiting
208                                 ball++;
209                             }
210 
211                             if (frameNumber == 9) {
212                                 int score = party.get(bowlIndex).getFrames().get(9).
    getTotalScore();
213                                 finalScores[bowlIndex][gameNumber] = score;
214                                 try {
215                                     SimpleDateFormat format = new SimpleDateFormat("H:m M/d
    /yyyy");
216                                     ScoreHistoryFile.addScore(currentThrower.getNick(),
    format.format(new Date()), Integer.toString(score));
217                                 } catch (Exception e) {
218                                     System.err.println("Exception in addScore. " + e);
219                                 }
220                             }
221 
222                             pinsetter.reset();
223                             bowlIndex++;
224 
225                         } else {
226                             frameNumber++;
227                             resetBowlerIterator();
228                             bowlIndex = 0;
229                             if (frameNumber > 9) {
230                                 gameFinished = true;
231                                 gameNumber++;
232                             }
233                         }
234                     } else if (partyAssigned && gameFinished) {
```

```
235                         EndGamePrompt prompt = new EndGamePrompt((party.get(0)).getNickName
    () + "'s Party");
236                         int result = prompt.getResult();
237                         prompt.distroy();
238                         prompt = null;
239
240                         System.out.println("result was: " + result);
241
242                         // TODO: send record of scores to control desk
243                         if (result == 1) {                    // yes, want to play again
244                             resetScores();
245                             resetBowlerIterator();
246
247                         } else if (result == 2) {// no, dont want to play another game
248                             Vector printVector;
249                             EndGameReport report = new EndGameReport(party.get(0).
    getNickName() + "'s Party", party);
250                             printVector = report.getResult();
251                             partyAssigned = false;
252                             Iterator scoreIt = party.iterator();
253                             party = null;
254                             partyAssigned = false;
255
256                             setChanged();
257                             notifyObservers();
258
259                             int myIndex = 0;
260                             while (scoreIt.hasNext()) {
261                                 Bowler thisBowler = (Bowler) scoreIt.next();
262                                 ScoreReport sr = new ScoreReport(thisBowler, finalScores[
    myIndex++], gameNumber);
263                                 sr.sendEmail(thisBowler.getEmail());
264                                 Iterator printIt = printVector.iterator();
265                                 while (printIt.hasNext()) {
266                                     if (thisBowler.getNick().equals(printIt.next())) {
267                                         System.out.println("Printing " + thisBowler.getNick
    ());
268                                         sr.sendPrintout();
269                                     }
270                                 }
271                             }
272                         }
273                     }
274
275                     try {
276                         Thread.sleep(10);
277                     } catch (Exception e) {
278                     }
279                 }
280             });
281         laneThread.start();
282
283     }
284
285     /** resetBowlerIterator()
286      *
287      * sets the current bower iterator back to the first bowler
288      *
289      * @pre the party as been assigned
290      * @post the iterator points to the first bowler in the party
```

```
291      */
292     private void resetBowlerIterator() {
293         bowlerIterator = party.iterator();
294     }
295 
296     /** resetScores()
297      *
298      * resets the scoring mechanism, must be called before scoring starts
299      *
300      * @pre the party has been assigned
301      * @post scoring system is initialized
302      */
303     private void resetScores() {
304         for(Object bowler : party){
305             ((Bowler) bowler).getFrames().get(0).reset();
306         }
307 
308         gameFinished = false;
309         frameNumber = 0;
310     }
311 
312     /** assignParty()
313      *
314      * assigns a party to this lane
315      *
316      * @pre none
317      * @post the party has been assigned to the lane
318      *
319      * @param theParty      Party to be assigned
320      */
321     public void assignParty( Vector<Bowler> theParty ) {
322         party = theParty;
323         resetBowlerIterator();
324         partyAssigned = true;
325 
326         finalScores = new int[party.size()][128]; //Hardcoding a max of 128 games, bite me.
327         gameNumber = 0;
328 
329         resetScores();
330     }
331 
332     /** markScore()
333      *
334      * Method that marks a bowlers score on the board.
335      *
336      * @param bowler        The current bowler
337      * @param pins  The bowler's score
338      */
339     private void markScore( Bowler bowler, int pins ){
340         bowler.markThrow(pins);
341         setChanged();
342         notifyObservers();
343     }
344 
345     /** isPartyAssigned()
346      *
347      * checks if a party is assigned to this lane
348      *
349      * @return true if party assigned, false otherwise
350      */
```

```
351     public boolean isPartyAssigned() {
352         return partyAssigned;
353     }
354 
355     public boolean isMechanicalProblem() {
356         return gameIsHalted;
357     }
358 
359     public int getCurrentFrame() {
360         return frameNumber;
361     }
362 
363     public int getIndex() {
364         return bowlIndex;
365     }
366 
367     public int getBall( ) {
368         return ball;
369     }
370 
371 
372     public Vector getParty() {
373         return party;
374     }
375 
376     public Bowler getBowler() {
377         return currentThrower;
378     }
379 
380     /**
381      * Accessor to get this Lane's pinsetter
382      *
383      * @return      A reference to this lane's pinsetter
384      */
385 
386     public Pinsetter getPinsetter() {
387         return pinsetter;
388     }
389 
390     /**
391      * Pause the execution of this game
392      */
393     public void pauseGame() {
394         gameIsHalted = true;
395 
396         setChanged();
397         notifyObservers();
398     }
399 
400     /**
401      * Resume the execution of this game
402      */
403     public void unPauseGame() {
404         gameIsHalted = false;
405 
406         setChanged();
407         notifyObservers();
408     }
409 
410     @Override
```

File - D:\Users\gjr8050\262refactoring\src\Main\Lane.java

```
411     public void update(Observable o, Object arg) {
412
413         int pinsDownThisThrow = (Integer) arg;
414
415         if (pinsDownThisThrow >=  0) {          // this is a real throw
416             markScore(currentThrower, pinsDownThisThrow);
417
418             // next logic handles the ?: what conditions dont allow them another throw?
419             // handle the case of 10th frame first
420             if (frameNumber == 9) {
421                 if (pinsetter.totalPinsDown() == 10) {
422                     pinsetter.resetPins();
423                     if(pinsetter.getThrowNumber() != 3) {
424                         tenthFrameStrike = true;
425                     }
426                 }
427
428                 if ((pinsetter.totalPinsDown() != 10) && (pinsetter.getThrowNumber() == 2
    && !tenthFrameStrike)) {
429                     canThrowAgain = false;
430                     //publish( lanePublish() );
431                 }
432
433                 if (pinsetter.getThrowNumber() == 3) {
434                     canThrowAgain = false;
435                     //publish( lanePublish() );
436                 }
437             } else { // its not the 10th frame
438
439                 if (pinsDownThisThrow == 10) {      // threw a strike
440                     canThrowAgain = false;
441                     //publish( lanePublish() );
442                 } else if (pinsetter.getThrowNumber() == 2) {
443                     canThrowAgain = false;
444                     //publish( lanePublish() );
445                 } else if (pinsetter.getThrowNumber() == 3)
446                     System.out.println("I'm here...");
447             }
448         }   //  this is not a real throw, probably a reset
449     }
450 }
451
```

File - D:\Users\gjr8050\262refactoring\src\Main\Alley.java

```
 1 package Main;
 2 /* Alley.java
 3  *
 4  *  Version:
 5  *          $Id$
 6  *
 7  *  Revisions:
 8  *      $Log: Alley.java,v $
 9  *      Revision 1.13  2003/02/02 23:26:32  ???
10  *      Alley now runs its own thread and polls for free lanes to assign queue members to
11  *
12  *      Revision 1.12  2003/02/02 20:46:13  ???
13  *      Added " 's Party" to party names.
14  *
15  *      Revision 1.11  2003/02/02 20:43:25  ???
16  *      misc cleanup
17  *
18  *      Revision 1.10  2003/02/02 17:49:10  ???
19  *      Fixed problem in getPartyQueue that was returning the first element as every element
    .
20  *
21  *      Revision 1.9  2003/02/02 17:39:48  ???
22  *      Added accessor for lanes.
23  *
24  *      Revision 1.8  2003/02/02 16:53:59  ???
25  *      Updated comments to match javadoc format.
26  *
27  *      Revision 1.7  2003/02/02 16:29:52  ???
28  *      Added AlleyEvent and AlleyObserver. Updated Queue to allow access to Vector so that
    contents could be viewed without destroying. Implemented observer model for most of Alley.
29  *
30  *      Revision 1.6  2003/02/02 06:09:39  ???
31  *      Updated many classes to support the AlleyView.
32  *
33  *      Revision 1.5  2003/01/26 23:16:10  ???
34  *      Improved thread handeling in lane/controldesk
35  *
36  *
37  */
38
39 /**
40  * Class that represents control desk
41  *
42  */
43
44 import java.util.*;
45 import java.io.*;
46
47 public class Alley extends Observable{
48
49     /** The collection of Lanes */
50     private HashSet lanes;
51
52     /** The party wait queue */
53     private Vector partyQueue;
54
55     /** The number of lanes represented */
56     private int numLanes;
57
58     /** The collection of subscribers */
```

```
59      private Vector subscribers;
60
61      private Thread alleyThread;
62
63      /**
64       *
65       * Constructor for the Alley class
66       *
67       * @param numLanes  the numbler of lanes to be represented
68       *
69       */
70      public Alley(int numLanes) {
71          this.numLanes = numLanes;
72          lanes = new HashSet(numLanes);
73          partyQueue = new Vector();
74
75          subscribers = new Vector();
76
77          for (int i = 0; i < numLanes; i++) {
78              lanes.add(new Lane());
79          }
80
81          runThread();
82      }
83
84      /**
85       * Main loop for Alley's thread
86       *
87       */
88      public void runThread() {
89          alleyThread = new Thread(new Runnable() {
90
91              @Override
92              public void run() {
93                  while (true) {
94
95                      assignLane();
96
97                      try {
98                          Thread.sleep(250);
99                      } catch (Exception e) {}
100                 }
101             }
102
103         });
104
105         alleyThread.start();
106     }
107
108
109     /**
110      * Retrieves a matching Bowler from the bowler database.
111      *
112      * @param nickName  The NickName of the Bowler
113      *
114      * @return a Bowler object.
115      *
116      */
117     private Bowler registerPatron(String nickName) {
118         Bowler patron = null;
```

```
119
120         try {
121             // only one patron / nick.... no dupes, no checks
122
123             patron = BowlerFile.getBowlerInfo(nickName);
124
125         } catch (FileNotFoundException e) {
126             System.err.println("Error..." + e);
127         } catch (IOException e) {
128             System.err.println("Error..." + e);
129         }
130
131         return patron;
132     }
133
134     /**
135      * Iterate through the available lanes and assign the paties in the wait queue if lanes
    are available.
136      *
137      */
138     public void assignLane() {
139         Iterator it = lanes.iterator();
140
141         while (it.hasNext() && !partyQueue.isEmpty()) {
142             Lane curLane = (Lane) it.next();
143
144             if (curLane.isPartyAssigned() == false) {
145                 System.out.println("ok... assigning this party");
146                 curLane.assignParty(((Vector) partyQueue.remove(0)));
147             }
148         }
149
150         setChanged();
151         notifyObservers();
152     }
153
154     /**
155      */
156
157     public void viewScores(Lane ln) {
158         // TODO: attach a LaneScoreView object to that lane
159     }
160
161     /**
162      * Creates a party from a Vector of nickNAmes and adds them to the wait queue.
163      *
164      * @param partyNicks    A Vector of NickNames
165      *
166      */
167
168     public void addPartyQueue(Vector partyNicks) {
169         Vector partyBowlers = new Vector();
170         for (int i = 0; i < partyNicks.size(); i++) {
171             Bowler newBowler = registerPatron(((String) partyNicks.get(i)));
172             partyBowlers.add(newBowler);
173         }
174         partyQueue.add(partyBowlers);
175
176         setChanged();
177         notifyObservers();
```

```
178     }
179
180     /**
181      * Returns a Vector of party names to be displayed in the GUI representation of the
    wait queue.
182      *
183      * @return a Vecotr of Strings
184      *
185      */
186
187     public Vector getPartyQueue() {
188         Vector displayPartyQueue = new Vector();
189         for ( int i=0; i < partyQueue.size(); i++ ) {
190             String nextParty =
191                 ((Bowler) ((Vector) ( partyQueue.get( i ) ))
192                     .get(0))
193                     .getNickName() + "'s Party";
194             displayPartyQueue.addElement(nextParty);
195         }
196         return displayPartyQueue;
197     }
198
199     /**
200      * Accessor for the number of lanes represented by the Alley
201      *
202      * @return an int containing the number of lanes represented
203      *
204      */
205
206     public int getNumLanes() {
207         return numLanes;
208     }
209
210     /**
211      * Accessor method for lanes
212      *
213      * @return a HashSet of Lanes
214      *
215      */
216
217     public HashSet getLanes() {
218         return lanes;
219     }
220 }
221
```

File - D:\Users\gjr8050\262refactoring\src\Main\drive.java

```
1  package Main;
2
3  import Views.AlleyView;
4
5  public class drive {
6
7      public static void main(String[] args) {
8
9          int numLanes = 3;
10         int maxPatronsPerParty=5;
11
12         Alley alley = new Alley(numLanes);
13
14         AlleyView cdv = new AlleyView(alley, maxPatronsPerParty);
15         alley.addObserver(cdv);
16
17     }
18 }
19
```

File - D:\Users\gjr8050\262refactoring\src\Main\Score.java

```java
package Main;

/**
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */

public class Score {

    private String nick;
    private String date;
    private String score;

    public Score( String nick, String date, String score ) {
        this.nick=nick;
        this.date=date;
        this.score=score;
    }

    public String getNickName() {
        return nick;
    }

    public String getDate() {
        return date;
    }

    public String getScore() {
        return score;
    }

    public String toString() {
        return nick + "\t" + date + "\t" + score;
    }

}

```

File - D:\Users\gjr8050\262refactoring\src\Main\Bowler.java

```java
package Main;
/*
 * Bowler.java
 *
 * Version:
 *     $Id$
 *
 * Revisions:
 *     $Log: Bowler.java,v $
 *     Revision 1.3  2003/01/15 02:57:40  ???
 *     Added accessors and and equals() method
 *
 *     Revision 1.2  2003/01/12 22:23:32  ???
 *     *** empty log message ***
 *
 *     Revision 1.1  2003/01/12 19:09:12  ???
 *     Adding Party, Lane, Bowler, and Alley.
 *
 */

import Main.Frame.Frame;

import java.util.ArrayList;

/**
 *  Class that holds all bowler info
 *
 */

public class Bowler {

    private String fullName;
    private String nickName;
    private String email;

    private ArrayList<Frame> frames;

    public Bowler( String nick, String full, String mail ) {
        nickName = nick;
        fullName = full;
        email = mail;

        frames = new ArrayList<>();
        for(int i = 0; i < 10; i++){
            Frame frame = new Frame();
            frames.add(frame);
            if(i > 0){
                frames.get(i - 1).setNext(frame);
                frame.setPrev(frames.get(i - 1));
            }
        }
    }

    public void markThrow(int pins){
        frames.get(0).setPinCount(pins);
    }

    public ArrayList<Frame> getFrames(){
        return frames;
    }
```

```
61
62      public String getNickName() {
63
64          return nickName;
65
66      }
67
68      public String getFullName ( ) {
69              return fullName;
70      }
71
72      public String getNick ( ) {
73          return nickName;
74      }
75
76      public String getEmail ( ) {
77          return email;
78      }
79
80      public boolean equals ( Bowler b) {
81          boolean retval = true;
82          if ( !(nickName.equals(b.getNickName())) ) {
83                  retval = false;
84          }
85          if ( !(fullName.equals(b.getFullName())) ) {
86                  retval = false;
87          }
88          if ( !(email.equals(b.getEmail())) ) {
89                  retval = false;
90          }
91          return retval;
92      }
93  }
```

File - D:\Users\gjr8050\262refactoring\src\Main\Pinsetter.java

```
 1 package Main;
 2 /*
 3  * Pinsetter.java
 4  *
 5  * Version:
 6  *   $Id$
 7  *
 8  * Revisions:
 9  *   $Log: Pinsetter.java,v $
10  *   Revision 1.21  2003/02/20 20:27:45  ???
11  *   Fouls disables.
12  *
13  *   Revision 1.20  2003/02/20 19:53:52  ???
14  *   Added foul support.  Still need to update laneview and test this.
15  *
16  *   Revision 1.19  2003/02/06 22:28:51  ???
17  *   added delay
18  *
19  *   Revision 1.18  2003/02/06 00:27:46  ???
20  *   Fixed a race condition
21  *
22  *   Revision 1.17  2003/02/05 23:56:07  ???
23  *   *** empty log message ***
24  *
25  *   Revision 1.16  2003/02/05 23:51:09  ???
26  *   Better random numbers.
27  *
28  *   Revision 1.15  2003/02/02 23:49:48  ???
29  *   Pinsetter generates an event when all pins are reset
30  *
31  *   Revision 1.14  2003/02/02 23:26:32  ???
32  *   Alley now runs its own thread and polls for free lanes to assign queue members to
33  *
34  *   Revision 1.13  2003/02/02 23:21:30  ???
35  *   pinsetter should give better results
36  *
37  *   Revision 1.12  2003/02/02 23:20:28  ???
38  *   pinsetter should give better results
39  *
40  *   Revision 1.11  2003/02/02 23:11:41  ???
41  *   parties can now play more than 1 game on a lane, and lanes are properly released after
    games
42  *
43  *   Revision 1.10  2003/02/01 19:14:42  ???
44  *   Will now set the pins back up at times other than the 10th frame.
45  *
46  *   Revision 1.9  2003/01/30 21:44:25  ???
47  *   Fixed speling of received in may places.
48  *
49  *   Revision 1.8  2003/01/26 22:34:44  ???
50  *   Total rewrite of lane and pinsetter for R2's observer model
51  *   Added Lane/Pinsetter Observer
52  *   Rewrite of scoring algorythm in lane
53  *
54  *   Revision 1.7  2003/01/19 21:55:24  ???
55  *   updated pinsetter to new spec
56  *
57  *   Revision 1.6  2003/01/16 04:59:59  ???
58  *   misc fixes across the board
59  *
```

```
 60  *   Revision 1.5  2003/01/13 22:35:21  ???
 61  *   Scoring works.
 62  *
 63  *   Revision 1.3  2003/01/12 22:37:20  ???
 64  *   Wrote a better algorythm for knocking down pins
 65  *
 66  *
 67  */
 68
 69 /**
 70  * Class to represent the pinsetter
 71  *
 72  */
 73
 74 import java.util.*;
 75
 76 public class Pinsetter extends Observable {
 77
 78     private Random rnd;
 79
 80     private boolean[] pins;
 81             /* 0-9 of state of pine, true for standing,
 82             false for knocked down
 83
 84             6   7   8   9
 85               3   4   5
 86                 2   1
 87                   0
 88
 89             */
 90     private boolean foul;
 91     private int throwNumber;
 92
 93     /** Pinsetter()
 94      *
 95      * Constructs a new pinsetter
 96      *
 97      * @pre none
 98      * @post a new pinsetter is created
 99      * @return Pinsetter object
100      */
101     public Pinsetter() {
102         pins = new boolean[10];
103         rnd = new Random();
104         foul = false;
105         reset();
106     }
107
108     /** ballThrown()
109      *
110      * Called to simulate a ball thrown comming in contact with the pinsetter
111      *
112      * @pre none
113      * @post pins may have been knocked down and the thrownumber has been incremented
114      */
115     public void ballThrown() {  // simulated event of ball hits sensor
116         int count = 0;
117         foul = false;
118         double skill = rnd.nextDouble();
119         for (int i=0; i <= 9; i++) {
```

```
120             if (pins[i]) {
121                 double pinluck = rnd.nextDouble();
122                 if (pinluck <= .04){
123                     foul = true;
124                 }
125                 if ( ((skill + pinluck)/2.0 * 1.2) > .5 ){
126                     pins[i] = false;
127                 }
128                 if (!pins[i]) {     // this pin just knocked down
129                     count++;
130                 }
131             }
132         }
133 
134         try {
135             Thread.sleep(500);              // pinsetter is where delay will be in a real
    game
136         } catch (Exception e) {}
137 
138         setChanged();
139         notifyObservers(count);
140 
141         throwNumber++;
142     }
143 
144     /** reset()
145      *
146      * Reset the pinsetter to its complete state
147      *
148      * @pre none
149      * @post pinsetters state is reset
150      */
151     public void reset() {
152         foul = false;
153         throwNumber = 1;
154         resetPins();
155 
156         try {
157             Thread.sleep(1000);
158         } catch (Exception e) {}
159 
160 
161         setChanged();
162         notifyObservers(-1);
163     }
164 
165     /** resetPins()
166      *
167      * Reset the pins on the pinsetter
168      *
169      * @pre none
170      * @post pins array is reset to all pins up
171      */
172     public void resetPins() {
173         for (int i=0; i <= 9; i++) {
174             pins[i] = true;
175         }
176     }
177 
178 
```

File - D:\Users\gjr8050\262refactoring\src\Main\Pinsetter.java

```
179     public boolean pinKnockedDown(int i) {
180         return !pins[i];
181     }
182 
183 
184     /** totalPinsDown()
185      *
186      * @return the total number of pins down for pinsetter that generated the event
187      */
188     public int totalPinsDown() {
189         int count = 0;
190 
191         for (int i=0; i <= 9; i++) {
192             if (pinKnockedDown(i)) {
193                 count++;
194             }
195         }
196 
197         return count;
198     }
199 
200     /** isFoulCommited()
201      *
202      * @return true if a foul was commited on the lane, false otherwise
203      */
204     public boolean isFoulCommited() {
205         return foul;
206     }
207 
208     /** getThrowNumber()
209      *
210      * @return current number of throws taken on this lane after last reset
211      */
212     public int getThrowNumber() {
213         return throwNumber;
214     }
215 
216 };
217 
218 
```

File - D:\Users\gjr8050\262refactoring\src\Main\BowlerFile.java

```
1  package Main;
2  /* BowlerFile.java
3   *
4   *  Version:
5   *          $Id$
6   *
7   *  Revisions:
8   *      $Log: BowlerFile.java,v $
9   *      Revision 1.5  2003/02/02 17:36:45  ???
10  *      Updated comments to match javadoc format.
11  *
12  *      Revision 1.4  2003/02/02 16:29:52  ???
13  *      Added AlleyEvent and AlleyObserver. Updated Queue to allow access to Vector so that
    contents could be viewed without destroying. Implemented observer model for most of Alley.
14  *
15  *
16  */
17
18 /**
19  * Class for interfacing with Bowler database
20  *
21  */
22
23 import java.util.*;
24 import java.io.*;
25
26 public class BowlerFile {
27
28     /** The location of the bowelr database */
29     private static String BOWLER_DAT = "BOWLERS.DAT";
30
31     /**
32      * Retrieves bowler information from the database and returns a Bowler objects with
    populated fields.
33      *
34      * @param nickName  the nickName of the bolwer to retrieve
35      *
36      * @return a Bowler object
37      *
38      */
39
40     public static Bowler getBowlerInfo(String nickName)
41         throws IOException, FileNotFoundException {
42
43         BufferedReader in = new BufferedReader(new FileReader(BOWLER_DAT));
44         String data;
45         while ((data = in.readLine()) != null) {
46             // File format is nick\tfname\te-mail
47             String[] bowler = data.split("\t");
48             if (nickName.equals(bowler[0])) {
49                 System.out.println(
50                     "Nick: "
51                         + bowler[0]
52                         + " Full: "
53                         + bowler[1]
54                         + " email: "
55                         + bowler[2]);
56                 return (new Bowler(bowler[0], bowler[1], bowler[2]));
57             }
58         }
```

```
59          System.out.println("Nick not found...");
60          return null;
61      }
62
63      /**
64       * Stores a Bowler in the database
65       *
66       * @param nickName  the NickName of the Bowler
67       * @param fullName  the FullName of the Bowler
68       * @param email the E-mail Address of the Bowler
69       *
70       */
71
72      public static void putBowlerInfo(
73          String nickName,
74          String fullName,
75          String email)
76          throws IOException, FileNotFoundException {
77
78          String data = nickName + "\t" + fullName + "\t" + email + "\n";
79
80          RandomAccessFile out = new RandomAccessFile(BOWLER_DAT, "rw");
81          out.skipBytes((int) out.length());
82          out.writeBytes(data);
83          out.close();
84      }
85
86      /**
87       * Retrieves a list of nicknames in the bowler database
88       *
89       * @return a Vector of Strings
90       *
91       */
92
93      public static Vector getBowlers()
94          throws IOException, FileNotFoundException {
95
96          Vector allBowlers = new Vector();
97
98          BufferedReader in = new BufferedReader(new FileReader(BOWLER_DAT));
99          String data;
100         while ((data = in.readLine()) != null) {
101             // File format is nick\tfname\te-mail
102             String[] bowler = data.split("\t");
103             //"Nick: bowler[0] Full: bowler[1] email: bowler[2]
104             allBowlers.add(bowler[0]);
105         }
106         return allBowlers;
107     }
108
109 }
```

File - D:\Users\gjr8050\262refactoring\src\Main\ScoreReport.java

```
1  package Main;
2  /**
3   *
4   * SMTP implementation based on code by R□al Gagnon mailto:real@rgagnon.com
5   *
6   */
7
8
9  import java.io.*;
10 import java.util.Vector;
11 import java.util.Iterator;
12 import java.net.*;
13 import java.awt.print.*;
14
15 public class ScoreReport {
16
17     private String content;
18
19     public ScoreReport( Bowler bowler, int[] scores, int games ) {
20         String nick = bowler.getNick();
21         String full = bowler.getFullName();
22         Vector v = null;
23         try{
24             v = ScoreHistoryFile.getScores(nick);
25         } catch (Exception e){System.err.println("Error: " + e);}
26
27         Iterator scoreIt = v.iterator();
28
29         content = "";
30         content += "--Lucky Strike Bowling Alley Score Report--\n";
31         content += "\n";
32         content += "Report for " + full + ", aka \"" + nick + "\":\n";
33         content += "\n";
34         content += "Final scores for this session: ";
35         content += scores[0];
36         for (int i = 1; i < games; i++){
37             content += ", " + scores[i];
38         }
39         content += ".\n";
40         content += "\n";
41         content += "\n";
42         content += "Previous scores by date: \n";
43         while (scoreIt.hasNext()){
44             Score score = (Score) scoreIt.next();
45             content += "   " + score.getDate() + " - " +  score.getScore();
46             content += "\n";
47         }
48         content += "\n\n";
49         content += "Thank you for your continuing patronage.";
50
51     }
52
53     public void sendEmail(String recipient) {
54         try {
55             Socket s = new Socket("osfmail.rit.edu", 25);
56             BufferedReader in =
57                 new BufferedReader(
58                     new InputStreamReader(s.getInputStream(), "8859_1"));
59             BufferedWriter out =
60                 new BufferedWriter(
```

```
61                      new OutputStreamWriter(s.getOutputStream(), "8859_1"));
62
63              String boundary = "DataSeparatorString";
64
65              // here you are supposed to send your username
66              sendln(in, out, "HELO world");
67              sendln(in, out, "MAIL FROM: <abc1234@rit.edu>");
68              sendln(in, out, "RCPT TO: <" + recipient + ">");
69              sendln(in, out, "DATA");
70              sendln(out, "Subject: Bowling Score Report ");
71              sendln(out, "From: <Lucky Strikes Bowling Club>");
72
73              sendln(out, "Content-Type: text/plain; charset=\"us-ascii\"\r\n");
74              sendln(out, content + "\n\n");
75              sendln(out, "\r\n");
76
77              sendln(in, out, ".");
78              sendln(in, out, "QUIT");
79              s.close();
80          } catch (Exception e) {
81              e.printStackTrace();
82          }
83      }
84
85      public void sendPrintout() {
86          PrinterJob job = PrinterJob.getPrinterJob();
87
88          PrintableText printobj = new PrintableText(content);
89
90          job.setPrintable(printobj);
91
92          if (job.printDialog()) {
93              try {
94                  job.print();
95              } catch (PrinterException e) {
96                  System.out.println(e);
97              }
98          }
99
100     }
101
102     public void sendln(BufferedReader in, BufferedWriter out, String s) {
103         try {
104             out.write(s + "\r\n");
105             out.flush();
106             // System.out.println(s);
107             s = in.readLine();
108             // System.out.println(s);
109         } catch (Exception e) {
110             e.printStackTrace();
111         }
112     }
113
114     public void sendln(BufferedWriter out, String s) {
115         try {
116             out.write(s + "\r\n");
117             out.flush();
118             System.out.println(s);
119         } catch (Exception e) {
120             e.printStackTrace();
```

```
121             }
122         }
123
124
125 }
126
```

File - D:\Users\gjr8050\262refactoring\src\Main\PrintableText.java

```
1  package Main;
2  /**
3   *
4   */
5
6  import java.awt.*;
7  import java.awt.print.*;
8  import java.awt.geom.*;
9  import java.awt.font.*;
10 import java.text.*;
11
12 public class PrintableText implements Printable {
13     String text;
14     int POINTS_PER_INCH;
15
16     public PrintableText(String t) {
17         POINTS_PER_INCH = 72;
18         text = t;
19     }
20
21     public int print(Graphics g, PageFormat pageFormat, int pageIndex) {
22         if (pageIndex > 0) {
23             return NO_SUCH_PAGE;
24         }
25
26         Graphics2D g2d = (Graphics2D) g; // Allow use of Java 2 graphics on
27
28         g2d.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
29         g2d.setPaint(Color.black);
30
31         Point2D.Double pen = new Point2D.Double(0.25 * POINTS_PER_INCH, 0.25 *
   POINTS_PER_INCH);
32
33         Font font = new Font ("courier", Font.PLAIN, 12);
34         FontRenderContext frc = g2d.getFontRenderContext();
35
36         String lines[] = text.split("\n");
37
38         for (int i=0; i < lines.length; i++) {
39             if (lines[i].length() > 0) {
40                 TextLayout layout = new TextLayout(lines[i], font, frc);
41                 layout.draw(g2d, (float) pen.x, (float) (pen.y + i*14));
42             }
43         }
44
45         return PAGE_EXISTS;
46     }
47
48 }
49
```

File - D:\Users\gjr8050\262refactoring\src\Main\ScoreHistoryFile.java

```
1  package Main;
2  /**
3   *
4   * To change this generated comment edit the template variable "typecomment":
5   * Window>Preferences>Java>Templates.
6   * To enable and disable the creation of type comments go to
7   * Window>Preferences>Java>Code Generation.
8   */
9
10 import java.util.*;
11 import java.io.*;
12
13 public class ScoreHistoryFile {
14
15     private static String SCOREHISTORY_DAT = "SCOREHISTORY.DAT";
16
17     public static void addScore(String nick, String date, String score)
18         throws IOException, FileNotFoundException {
19
20         String data = nick + "\t" + date + "\t" + score + "\n";
21
22         RandomAccessFile out = new RandomAccessFile(SCOREHISTORY_DAT, "rw");
23         out.skipBytes((int) out.length());
24         out.writeBytes(data);
25         out.close();
26     }
27
28     public static Vector getScores(String nick)
29         throws IOException, FileNotFoundException {
30         Vector scores = new Vector();
31
32         BufferedReader in =
33             new BufferedReader(new FileReader(SCOREHISTORY_DAT));
34         String data;
35         while ((data = in.readLine()) != null) {
36             // File format is nick\tfname\te-mail
37             String[] scoredata = data.split("\t");
38             //"Nick: scoredata[0] Date: scoredata[1] Score: scoredata[2]
39             if (nick.equals(scoredata[0])) {
40                 scores.add(new Score(scoredata[0], scoredata[1], scoredata[2]));
41             }
42         }
43         return scores;
44     }
45
46 }
47
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\Ball.java

```java
package Main.Frame;

/**
 * provides a type-safe way to reference the pin counts stored in each frame
 * Created by gjr8050 on 5/1/2016.
 */
public enum Ball {
    ONE(0),
    TWO(1),
    THREE(2);

    private final int index;

    Ball(int num){
        this.index = num;
    }

    int getIndex(){
        return index;
    }
}
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\Frame.java

```java
package Main.Frame;

import java.util.HashMap;

/**
 * Stores pin counts, total score, and state; provides interface for managing
 * those values through a state-machine and chain-of-responsibility
 * Created by gjr8050 on 5/1/2016.
 */
public class Frame {
    static final int Unset = -1;

    private HashMap<Ball, Integer> pinCounts;

    Frame next;
    Frame prev;

    private FrameState state;

    private int totalScore = Unset;

    public Frame(){
        state = FrameState.ACTIVE;
        pinCounts = new HashMap<>();
        pinCounts.put(Ball.ONE, Unset);
        pinCounts.put(Ball.TWO, Unset);
        pinCounts.put(Ball.THREE, Unset);
    }

    public void setNext(Frame frame){
        next = frame;
    }

    public void setPrev(Frame frame){
        prev = frame;
    }

    boolean isStrike(){
        return getPinCount(Ball.ONE) == 10;
    }

    boolean isSpare(){
        return !isStrike() && getPinCount(Ball.ONE) + getPinCount(Ball.TWO) == 10;
    }

    int getPinCount(Ball ball){
        return isBowled(ball) ? pinCounts.get(ball) : 0;
    }

    int getScore(Ball ball){
        return pinCounts.get(ball);
    }

    /**
     * Gets the string mark for a ball, does not handle spares
     * @param ball the ball to get a mark for
     * @return an empty string for un-bowled, a dash for no pins, or the count as a string
     */
    String getMark(Ball ball){
        if(!isBowled(ball))
```

```
61              return "";
62          else {
63              int pins = getPinCount(ball);
64              switch (pins){
65                  case 0: return "-";
66                  case 10: return "X";
67                  default: return Integer.toString(pins);
68              }
69          }
70      }
71
72      /**
73       * returns the base pin count on a frame
74       * @return numeric value or Frame.Unset if can't be determined
75       */
76      int getBaseScore(){
77          return isLast()
78                  ? scoreAdd(scoreAdd(getPinCount(Ball.ONE), getPinCount(Ball.TWO)),
    getPinCount(Ball.THREE))
79                  : scoreAdd(getPinCount(Ball.ONE), getPinCount(Ball.TWO));
80      }
81
82      /**
83       * Indicates if the frame has been bowled
84       */
85      boolean isBowled(){
86          return state != FrameState.ACTIVE;
87      }
88
89      /**
90       * Indicates if a specific ball on a frame has been bowled
91       * @param ball the ball to check
92       * @return whether the given ball is Frame.Unset
93       */
94      boolean isBowled(Ball ball){
95          return pinCounts.get(ball) != Unset;
96      }
97
98      /**
99       * Calculate the number of points this frame would contribute to a special scoring case
     on a previous frame
100      * @param forState the special scoring case being handled
101      * @return the points to add or Frame.Unset
102      */
103     int getChainScore(FrameState forState){
104         return state.getChainScore(this, forState);
105     }
106
107     /**
108      * Sets pin count for the given ball
109      * @param ball the ball to set count for
110      * @param count pins bowled on that ball
111      */
112     void setPinCount(Ball ball, int count){
113         pinCounts.put(ball, count);
114     }
115
116     void setFrameState(FrameState state){
117         this.state = state;
118     }
```

```
119
120     public void setPinCount(int count){
121         state.setPinCount(this, count);
122     }
123
124     int calculateFrameScore(){
125         return state.calculateFrameScore(this);
126     }
127
128     /**
129      * indicates if this is the last frame
130      */
131     boolean isLast(){
132         return next == null;
133     }
134
135     /**
136      * Calculates total score for this frame or unsets it if not bowled yet
137      */
138     void calculateTotalScore(){
139         if(!isBowled())
140             totalScore = Unset;
141         else {
142             totalScore = Frame.scoreAdd(calculateFrameScore(), (prev != null ? prev.
    getTotalScore() : 0));
143         }
144     }
145
146     /**
147      * returns the total scored and calculates total score for this frame if it has not yet
     been calculated
148      * @return the total score on this frame
149      */
150     public int getTotalScore(){
151         if(totalScore == Unset)
152             calculateTotalScore();
153
154         return totalScore;
155     }
156
157     /**
158      * Gets the total score for this frame or an empty string if not calculated yet
159      * @return string representation of total score
160      */
161     public String getScoreMark(){
162         return getTotalScore() == Unset ? "" : Integer.toString(getTotalScore());
163     }
164
165     public String[] getBallMarks(){
166         return state.getMarks(this);
167     }
168
169     /**
170      * Resets all properties of the frame
171      */
172     public void reset(){
173         totalScore = Unset;
174         pinCounts.put(Ball.ONE, Unset);
175         pinCounts.put(Ball.TWO, Unset);
176         pinCounts.put(Ball.THREE, Unset);
```

```
177         setFrameState(FrameState.ACTIVE);
178
179         if(next != null)
180             next.reset();
181     }
182
183     /**
184      * Ensures Unset scores propagate throughout a calculation
185      * @param a number
186      * @param b number
187      * @return the sum of a and b, or Unset if either one is unset
188      */
189     static int scoreAdd(int a, int b){
190         return a == Frame.Unset || b == Frame.Unset ? Frame.Unset : a + b;
191     }
192 }
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\FrameState.java

```java
package Main.Frame;

/**
 * Provides static reference to pre-initialized instances of
 * FrameHandler subclasses
 * Created by gjr8050 on 5/1/2016.
 */
enum FrameState {
    ACTIVE(new ActiveFrame()),
    BOWLED(new BowledFrame()),
    STRIKE(new StrikeFrame()),
    SPARE(new SpareFrame());

    private final FrameHandler state;

    FrameState(FrameHandler handler){
        state = handler;
    }

    public void setPinCount(Frame frame, int count){
        state.setPinCount(frame, count);
    }

    public int calculateFrameScore(Frame frame){
        return state.calculateFrameScore(frame);
    }

    public int getChainScore(Frame frame, FrameState forState){
        return state.getChainScore(frame, forState);
    }

    public String[] getMarks(Frame frame){return state.getMarks(frame); }
}
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\SpareFrame.java

```java
package Main.Frame;

/**
 * Defines behaviors when a frame has pin counts that result in a Spare
 * Created by gjr8050 on 5/1/2016.
 */
public class SpareFrame extends FrameHandler {
    @Override
    int calculateFrameScore(Frame frame) {
        int score = frame.getBaseScore();

        if(frame.isLast()){
            if(!frame.isBowled(Ball.THREE))
                return Frame.Unset;
        }
        else
            score = Frame.scoreAdd(score, frame.next.getChainScore(FrameState.SPARE));

        return score;
    }

    @Override
    void setPinCount(Frame frame, int pinCount) {
        if(frame.isLast()){
            if(!frame.isBowled(Ball.THREE))
                frame.setPinCount(Ball.THREE, pinCount);
            else
                throw new UnsupportedOperationException("No valid frame to set ball on");
        }
        else
            frame.next.setPinCount(pinCount);
    }

    @Override
    int getChainScore(Frame frame, FrameState forState) {
        switch(forState){
            case STRIKE:
                return Frame.scoreAdd(frame.getScore(Ball.ONE), frame.getScore(Ball.TWO));
            case SPARE:
                return frame.getScore(Ball.ONE);
            default:
                return 0;
        }
    }

    @Override
    String[] getMarks(Frame frame) {
        return new String[]{
                frame.getMark(Ball.ONE),
                "/",
                frame.getMark(Ball.THREE)
        };
    }
}
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\ActiveFrame.java

```java
package Main.Frame;

/**
 * Behaviors when a frame does not have it's pin counts finalized
 * Created by gjr8050 on 5/1/2016.
 */
public class ActiveFrame extends FrameHandler {
    @Override
    int calculateFrameScore(Frame frame) {
        return Frame.Unset;
    }

    @Override
    void setPinCount(Frame frame, int count) {
        if(!frame.isBowled(Ball.ONE))
            frame.setPinCount(Ball.ONE, count);
        else if(!frame.isBowled(Ball.TWO))
            frame.setPinCount(Ball.TWO, count);

        if(frame.isStrike())
            frame.setFrameState(FrameState.STRIKE);
        else if(frame.isSpare())
            frame.setFrameState(FrameState.SPARE);
        else if(frame.isBowled(Ball.TWO))
            frame.setFrameState(FrameState.BOWLED);
    }

    @Override
    int getChainScore(Frame frame, FrameState forState) {
        switch(forState){
            case STRIKE:
                return Frame.scoreAdd(frame.getScore(Ball.ONE), frame.getScore(Ball.TWO));
            case SPARE:
                return frame.getScore(Ball.ONE);
            default:
                return 0;
        }
    }

    @Override
    String[] getMarks(Frame frame) {
        return new String[]{"","",""};
    }
}
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\BowledFrame.java

```java
package Main.Frame;

/**
 * Defines behavior when frame has finalized pin counts and no special scoring
 * Created by gjr8050 on 5/1/2016.
 */
public class BowledFrame extends FrameHandler {
    @Override
    int calculateFrameScore(Frame frame) {
        return frame.getBaseScore();
    }

    @Override
    void setPinCount(Frame frame, int pinCount) {
        if(frame.isLast())
            throw new UnsupportedOperationException("No valid frame to set ball on");

        frame.next.setPinCount(pinCount);
    }

    @Override
    int getChainScore(Frame frame, FrameState forState) {
        switch(forState){
            case STRIKE:
                return Frame.scoreAdd(frame.getScore(Ball.ONE), frame.getScore(Ball.TWO));
            case SPARE:
                return frame.getScore(Ball.ONE);
            default:
                return 0;
        }
    }

    @Override
    String[] getMarks(Frame frame) {
        return new String[] {
                frame.getMark(Ball.ONE),
                frame.getMark(Ball.TWO),
                frame.getMark(Ball.THREE)
        };
    }
}

```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\StrikeFrame.java

```java
package Main.Frame;

/**
 * Defines behaviors when a frame has pin counts that result in a Strike
 * Created by gjr8050 on 5/1/2016.
 */
public class StrikeFrame extends FrameHandler {
    @Override
    int calculateFrameScore(Frame frame) {
        int score = frame.getBaseScore();

        if(frame.isLast()){
            if(!frame.isBowled(Ball.THREE))
                return Frame.Unset;
        }
        else
            score = Frame.scoreAdd(score, frame.next.getChainScore(FrameState.STRIKE));

        return score;
    }

    @Override
    void setPinCount(Frame frame, int pinCount) {
        if(frame.isLast()){
            if(!frame.isBowled(Ball.TWO))
                frame.setPinCount(Ball.TWO, pinCount);
            else if(!frame.isBowled(Ball.THREE))
                frame.setPinCount(Ball.THREE, pinCount);
            else
                throw new UnsupportedOperationException("No valid frame to set ball on");
        }
        else
            frame.next.setPinCount(pinCount);
    }

    @Override
    int getChainScore(Frame frame, FrameState forState) {
        switch(forState){
            case STRIKE:
                if(frame.isLast())
                    return Frame.scoreAdd(frame.getScore(Ball.ONE), frame.getScore(Ball.TWO)
);
                else
                    return Frame.scoreAdd(frame.getScore(Ball.ONE), frame.next.getScore(Ball
.ONE));
            case SPARE:
                return frame.getScore(Ball.ONE);
            default:
                return 0;
        }
    }

    @Override
    String[] getMarks(Frame frame) {
        /*
        getMark handles the strike case easily, because if a ball was 10 pins, it is a
strike,
        and the 2nd ball on a non-last frame is un-bowled, we just need to handle the last
frame
         */
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\StrikeFrame.java

```
57          String mark3 = frame.isLast() && frame.getScore(Ball.TWO) + frame.getScore(Ball.
    THREE) == 10 ? "/" : frame.getMark(Ball.THREE);
58          return new String[] {
59                  frame.getMark(Ball.ONE),
60                  frame.getMark(Ball.TWO),
61                  mark3
62          };
63      }
64  }
```

File - D:\Users\gjr8050\262refactoring\src\Main\Frame\FrameHandler.java

```java
package Main.Frame;

/**
 * Defines operations that will be handled through the state machine
 * Created by gjr8050 on 5/1/2016.
 */

abstract class FrameHandler {

    /**
     * Calculates the number of points that a frame contributes to the total score
     * @param frame the frame to calculate points for
     * @return score or Frame.Unset if can't yet be calculated
     */
    abstract int calculateFrameScore(Frame frame);

    /**
     * Sets the next pin count on a frame or passes to the next frame if frame has been bowled
     * @param frame frame to set pin count on
     * @param pinCount number of pins
     */
    abstract void setPinCount(Frame frame, int pinCount);

    /**
     * Gets the number of points this frame would contribute to a special scoring case on a previous frame
     * @param frame the frame to get points from
     * @param forState the type of special condition for which points should be calculated
     * @return points or Frame.Unset if can't be determined
     */
    abstract int getChainScore(Frame frame, FrameState forState);

    /**
     * Gets a series of string marks to be displayed in scoring interface
     * @param frame frame to determine marks for
     * @return a String array of length 3 with appropriate scoring marks
     */
    abstract String[] getMarks(Frame frame);
}
```

File - D:\Users\gjr8050\262refactoring\src\Views\LaneView.java

```
1  package Views;
2  /*
3   *  constructs a prototype Lane View
4   *
5   */
6
7  import Main.*;
8
9  import java.awt.*;
10 import java.awt.event.*;
11 import javax.swing.*;
12 import java.util.*;
13
14 public class LaneView implements ActionListener, Observer {
15
16     private boolean initDone = true;
17
18     JFrame frame;
19     Container cpanel;
20     Vector bowlers;
21
22     JPanel[][] balls;
23     JLabel[][] ballLabel;
24     JPanel[][] scores;
25     JLabel[][] scoreLabel;
26     JPanel[][] ballGrid;
27     JPanel[] pins;
28
29     JButton maintenance;
30     Lane lane;
31
32     public LaneView(Lane lane, int laneNum) {
33
34         this.lane = lane;
35
36         initDone = true;
37         frame = new JFrame("Lane " + laneNum + ":");
38         cpanel = frame.getContentPane();
39         cpanel.setLayout(new BorderLayout());
40
41         frame.addWindowListener(new WindowAdapter() {
42             public void windowClosing(WindowEvent e) {
43                 frame.hide();
44             }
45         });
46
47         cpanel.add(new JPanel());
48
49     }
50
51     public void show() {
52         frame.show();
53     }
54
55     public void hide() {
56         frame.hide();
57     }
58
59     private JPanel makeFrame(Vector party) {
60
```

```
61          initDone = false;
62          bowlers = party;
63          int numBowlers = bowlers.size();
64
65          JPanel panel = new JPanel();
66
67          panel.setLayout(new GridLayout(0, 1));
68
69          balls = new JPanel[numBowlers][23];
70          ballLabel = new JLabel[numBowlers][23];
71          scores = new JPanel[numBowlers][10];
72          scoreLabel = new JLabel[numBowlers][10];
73          ballGrid = new JPanel[numBowlers][10];
74          pins = new JPanel[numBowlers];
75
76          for (int i = 0; i != numBowlers; i++) {
77              for (int j = 0; j != 23; j++) {
78                  ballLabel[i][j] = new JLabel(" ");
79                  balls[i][j] = new JPanel();
80                  balls[i][j].setBorder(
81                      BorderFactory.createLineBorder(Color.BLACK));
82                  balls[i][j].add(ballLabel[i][j]);
83              }
84          }
85
86          for (int i = 0; i != numBowlers; i++) {
87              for (int j = 0; j != 9; j++) {
88                  ballGrid[i][j] = new JPanel();
89                  ballGrid[i][j].setLayout(new GridLayout(0, 3));
90                  ballGrid[i][j].add(new JLabel("  "), BorderLayout.EAST);
91                  ballGrid[i][j].add(balls[i][2 * j], BorderLayout.EAST);
92                  ballGrid[i][j].add(balls[i][2 * j + 1], BorderLayout.EAST);
93              }
94              int j = 9;
95              ballGrid[i][j] = new JPanel();
96              ballGrid[i][j].setLayout(new GridLayout(0, 3));
97              ballGrid[i][j].add(balls[i][2 * j]);
98              ballGrid[i][j].add(balls[i][2 * j + 1]);
99              ballGrid[i][j].add(balls[i][2 * j + 2]);
100         }
101
102         for (int i = 0; i != numBowlers; i++) {
103             pins[i] = new JPanel();
104             pins[i].setBorder(
105                 BorderFactory.createTitledBorder(
106                     ((Bowler) bowlers.get(i)).getNick()));
107             pins[i].setLayout(new GridLayout(0, 10));
108             for (int k = 0; k != 10; k++) {
109                 scores[i][k] = new JPanel();
110                 scoreLabel[i][k] = new JLabel("  ", SwingConstants.CENTER);
111                 scores[i][k].setBorder(
112                     BorderFactory.createLineBorder(Color.BLACK));
113                 scores[i][k].setLayout(new GridLayout(0, 1));
114                 scores[i][k].add(ballGrid[i][k], BorderLayout.EAST);
115                 scores[i][k].add(scoreLabel[i][k], BorderLayout.SOUTH);
116                 pins[i].add(scores[i][k], BorderLayout.EAST);
117             }
118             panel.add(pins[i]);
119         }
120
```

```
121         initDone = true;
122         return panel;
123     }
124 
125     public void actionPerformed(ActionEvent e) {
126         if (e.getSource().equals(maintenance)) {
127             lane.pauseGame();
128         }
129     }
130 
131     private void createLaneView(){
132         System.out.println("Making the frame.");
133         cpanel.removeAll();
134         cpanel.add(makeFrame(lane.getParty()), "Center");
135 
136         // Button Panel
137         JPanel buttonPanel = new JPanel();
138         buttonPanel.setLayout(new FlowLayout());
139 
140         maintenance = new JButton("Maintenance Call");
141         JPanel maintenancePanel = new JPanel();
142         maintenancePanel.setLayout(new FlowLayout());
143         maintenance.addActionListener(this);
144         maintenancePanel.add(maintenance);
145 
146         buttonPanel.add(maintenancePanel);
147 
148         cpanel.add(buttonPanel, "South");
149 
150         frame.pack();
151     }
152 
153     @Override
154     public void update(Observable o, Object arg) {
155         if (lane.isPartyAssigned()) {
156             int numBowlers = lane.getParty().size();
157             while (!initDone) {
158                 //System.out.println("chillin' here.");
159                 try {
160                     Thread.sleep(1);
161                 } catch (Exception e) {
162                 }
163             }
164 
165             if (lane.getCurrentFrame() == 0 && lane.getBall() == 0 && lane.getIndex() == 0)
    {
166                 createLaneView();
167             }
168 
169             for (int k = 0; k < numBowlers; k++) {
170                 Bowler bowler = (Bowler)lane.getParty().get(k);
171                 ArrayList<Main.Frame.Frame> frames = bowler.getFrames();
172                 //Display Pin Counts
173                 for(int i = 0; i < 10; i++){
174                     String[] marks = frames.get(i).getBallMarks();
175                     ballLabel[k][i * 2].setText(marks[0]);
176                     ballLabel[k][i * 2 + 1].setText(marks[1]);
177 
178                     if(i == 9)
179                         ballLabel[k][i * 2 + 2].setText(marks[2]);
```

```
180                     }
181                     //Display Scores
182                     for(int i = 0; i < 10; i++){
183                         scoreLabel[k][i].setText(frames.get(i).getScoreMark());
184                     }
185                 }
186             }
187         }
188 }
189
190
191
```

File - D:\Users\gjr8050\262refactoring\src\Views\AlleyView.java

```
1  package Views;
2  /* AlleyView.java
3   *
4   *  Version:
5   *          $Id$
6   *
7   *  Revisions:
8   *      $Log$
9   *
10  */
11
12 /**
13  * Class for representation of the control desk
14  *
15  */
16
17 import Main.*;
18
19 import java.awt.*;
20 import java.awt.event.*;
21 import javax.swing.*;
22 import javax.swing.border.*;
23
24 import java.util.*;
25
26 public class AlleyView implements ActionListener, Observer {
27
28     private JButton addParty, finished, assign;
29     private JFrame win;
30     private JList partyList;
31
32     /** The maximum  number of members in a party */
33     private int maxMembers;
34
35     private Alley alley;
36
37     /**
38      * Displays a GUI representation of the Alley
39      *
40      */
41
42     public AlleyView(Alley alley, int maxMembers) {
43
44         this.alley = alley;
45         this.maxMembers = maxMembers;
46         int numLanes = alley.getNumLanes();
47
48         win = new JFrame("Control Desk");
49         win.getContentPane().setLayout(new BorderLayout());
50         ((JPanel) win.getContentPane()).setOpaque(false);
51
52         JPanel colPanel = new JPanel();
53         colPanel.setLayout(new BorderLayout());
54
55         // Controls Panel
56         JPanel controlsPanel = new JPanel();
57         controlsPanel.setLayout(new GridLayout(3, 1));
58         controlsPanel.setBorder(new TitledBorder("Controls"));
59
60         addParty = new JButton("Add Party");
```

```
61          JPanel addPartyPanel = new JPanel();
62          addPartyPanel.setLayout(new FlowLayout());
63          addParty.addActionListener(this);
64          addPartyPanel.add(addParty);
65          controlsPanel.add(addPartyPanel);
66
67          assign = new JButton("Assign Lanes");
68          JPanel assignPanel = new JPanel();
69          assignPanel.setLayout(new FlowLayout());
70          assign.addActionListener(this);
71          assignPanel.add(assign);
72  //      controlsPanel.add(assignPanel);
73
74          finished = new JButton("Finished");
75          JPanel finishedPanel = new JPanel();
76          finishedPanel.setLayout(new FlowLayout());
77          finished.addActionListener(this);
78          finishedPanel.add(finished);
79          controlsPanel.add(finishedPanel);
80
81          // Lane Status Panel
82          JPanel laneStatusPanel = new JPanel();
83          laneStatusPanel.setLayout(new GridLayout(numLanes, 1));
84          laneStatusPanel.setBorder(new TitledBorder("Lane Status"));
85
86          HashSet lanes= alley.getLanes();
87          Iterator it = lanes.iterator();
88          int laneCount=0;
89          while (it.hasNext()) {
90              Lane curLane = (Lane) it.next();
91              LaneStatusView laneStat = new LaneStatusView(curLane,(laneCount+1));
92              curLane.addObserver(laneStat);
93              JPanel lanePanel = laneStat.showLane();
94              lanePanel.setBorder(new TitledBorder("Lane" + ++laneCount ));
95              laneStatusPanel.add(lanePanel);
96
97              ((Pinsetter)curLane.getPinsetter()).addObserver(laneStat);
98          }
99
100         // Party Queue Panel
101         JPanel partyPanel = new JPanel();
102         partyPanel.setLayout(new FlowLayout());
103         partyPanel.setBorder(new TitledBorder("Party Queue"));
104
105         Vector empty = new Vector();
106         empty.add("(Empty)");
107
108         partyList = new JList(empty);
109         partyList.setFixedCellWidth(120);
110         partyList.setVisibleRowCount(10);
111         JScrollPane partyPane = new JScrollPane(partyList);
112         partyPane.setVerticalScrollBarPolicy(
113             JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
114         partyPanel.add(partyPane);
115         //      partyPanel.add(partyList);
116
117         // Clean up main panel
118         colPanel.add(controlsPanel, "East");
119         colPanel.add(laneStatusPanel, "Center");
120         colPanel.add(partyPanel, "West");
```

```
121
122         win.getContentPane().add("Center", colPanel);
123
124         win.pack();
125
126         /* Close program when this window closes */
127         win.addWindowListener(new WindowAdapter() {
128             public void windowClosing(WindowEvent e) {
129                 System.exit(0);
130             }
131         });
132
133         // Center Window on Screen
134         Dimension screenSize = (Toolkit.getDefaultToolkit()).getScreenSize();
135         win.setLocation(
136             ((screenSize.width) / 2) - ((win.getSize().width) / 2),
137             ((screenSize.height) / 2) - ((win.getSize().height) / 2));
138         win.show();
139
140     }
141
142     /**
143      * Handler for actionEvents
144      *
145      * @param e the ActionEvent that triggered the handler
146      *
147      */
148
149     public void actionPerformed(ActionEvent e) {
150         if (e.getSource().equals(addParty)) {
151             AddPartyView addPartyWin = new AddPartyView(this, maxMembers);
152         }
153         if (e.getSource().equals(assign)) {
154             alley.assignLane();
155         }
156         if (e.getSource().equals(finished)) {
157             win.hide();
158             System.exit(0);
159         }
160     }
161
162     /**
163      * Receive a new party from andPartyView.
164      *
165      * @param addPartyView  the AddPartyView that is providing a new party
166      *
167      */
168
169     public void updateAddParty(AddPartyView addPartyView) {
170         alley.addPartyQueue(addPartyView.getParty());
171     }
172
173
174     @Override
175     public void update(Observable o, Object arg) {
176         partyList.setListData(((Alley)o).getPartyQueue());
177     }
178 }
179
```

File - D:\Users\gjr8050\262refactoring\src\Views\AddPartyView.java

```
1  package Views;
2  /* AddPartyView.java
3   *
4   *  Version:
5   *       $Id$
6   *
7   *  Revisions:
8   *      $Log: AddPartyView.java,v $
9   *      Revision 1.7  2003/02/20 02:05:53  ???
10  *      Fixed addPatron so that duplicates won't be created.
11  *
12  *      Revision 1.6  2003/02/09 20:52:46  ???
13  *      Added comments.
14  *
15  *      Revision 1.5  2003/02/02 17:42:09  ???
16  *      Made updates to migrate to observer model.
17  *
18  *      Revision 1.4  2003/02/02 16:29:52  ???
19  *      Added AlleyEvent and AlleyObserver. Updated Queue to allow access to Vector so that
    contents could be viewed without destroying. Implemented observer model for most of Alley.
20  *
21  *
22  */
23
24 /**
25  * Class for GUI components need to add a party
26  *
27  */
28
29 import java.awt.*;
30 import java.awt.event.*;
31 import javax.swing.*;
32 import javax.swing.border.*;
33 import javax.swing.event.*;
34 import Main.BowlerFile;
35 import Main.Bowler;
36
37 import java.util.*;
38
39 /**
40  * Constructor for GUI used to Add Parties to the waiting party queue.
41  *
42  */
43
44 public class AddPartyView implements ActionListener, ListSelectionListener {
45
46     private int maxSize;
47
48     private JFrame win;
49     private JButton addPatron, newPatron, remPatron, finished;
50     private JList partyList, allBowlers;
51     private Vector party, bowlerdb;
52     private Integer lock;
53
54     private AlleyView alley;
55
56     private String selectedNick, selectedMember;
57
58     public AddPartyView(AlleyView alley, int max) {
59
```

```
60          this.alley = alley;
61          maxSize = max;
62
63          win = new JFrame("Add Party");
64          win.getContentPane().setLayout(new BorderLayout());
65          ((JPanel) win.getContentPane()).setOpaque(false);
66
67          JPanel colPanel = new JPanel();
68          colPanel.setLayout(new GridLayout(1, 3));
69
70          // Party Panel
71          JPanel partyPanel = new JPanel();
72          partyPanel.setLayout(new FlowLayout());
73          partyPanel.setBorder(new TitledBorder("Your Party"));
74
75          party = new Vector();
76          Vector empty = new Vector();
77          empty.add("(Empty)");
78
79          partyList = new JList(empty);
80          partyList.setFixedCellWidth(120);
81          partyList.setVisibleRowCount(5);
82          partyList.addListSelectionListener(this);
83          JScrollPane partyPane = new JScrollPane(partyList);
84          //        partyPane.setVerticalScrollBarPolicy(JScrollPane.
    VERTICAL_SCROLLBAR_ALWAYS);
85          partyPanel.add(partyPane);
86
87          // Bowler Database
88          JPanel bowlerPanel = new JPanel();
89          bowlerPanel.setLayout(new FlowLayout());
90          bowlerPanel.setBorder(new TitledBorder("Bowler Database"));
91
92          try {
93              bowlerdb = new Vector(BowlerFile.getBowlers());
94          } catch (Exception e) {
95              System.err.println("File Error");
96              bowlerdb = new Vector();
97          }
98          allBowlers = new JList(bowlerdb);
99          allBowlers.setVisibleRowCount(8);
100         allBowlers.setFixedCellWidth(120);
101         JScrollPane bowlerPane = new JScrollPane(allBowlers);
102         bowlerPane.setVerticalScrollBarPolicy(
103             JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
104         allBowlers.addListSelectionListener(this);
105         bowlerPanel.add(bowlerPane);
106
107         // Button Panel
108         JPanel buttonPanel = new JPanel();
109         buttonPanel.setLayout(new GridLayout(4, 1));
110
111         Insets buttonMargin = new Insets(4, 4, 4, 4);
112
113         addPatron = new JButton("Add to Party");
114         JPanel addPatronPanel = new JPanel();
115         addPatronPanel.setLayout(new FlowLayout());
116         addPatron.addActionListener(this);
117         addPatronPanel.add(addPatron);
118
```

```
119         remPatron = new JButton("Remove Member");
120         JPanel remPatronPanel = new JPanel();
121         remPatronPanel.setLayout(new FlowLayout());
122         remPatron.addActionListener(this);
123         remPatronPanel.add(remPatron);
124 
125         newPatron = new JButton("New Patron");
126         JPanel newPatronPanel = new JPanel();
127         newPatronPanel.setLayout(new FlowLayout());
128         newPatron.addActionListener(this);
129         newPatronPanel.add(newPatron);
130 
131         finished = new JButton("Finished");
132         JPanel finishedPanel = new JPanel();
133         finishedPanel.setLayout(new FlowLayout());
134         finished.addActionListener(this);
135         finishedPanel.add(finished);
136 
137         buttonPanel.add(addPatronPanel);
138         buttonPanel.add(remPatronPanel);
139         buttonPanel.add(newPatronPanel);
140         buttonPanel.add(finishedPanel);
141 
142         // Clean up main panel
143         colPanel.add(partyPanel);
144         colPanel.add(bowlerPanel);
145         colPanel.add(buttonPanel);
146 
147         win.getContentPane().add("Center", colPanel);
148 
149         win.pack();
150 
151         // Center Window on Screen
152         Dimension screenSize = (Toolkit.getDefaultToolkit()).getScreenSize();
153         win.setLocation(
154             ((screenSize.width) / 2) - ((win.getSize().width) / 2),
155             ((screenSize.height) / 2) - ((win.getSize().height) / 2));
156         win.show();
157 
158     }
159 
160     public void actionPerformed(ActionEvent e) {
161         if (e.getSource().equals(addPatron)) {
162             if (selectedNick != null && party.size() < maxSize) {
163                 if (party.contains(selectedNick)) {
164                     System.err.println("Member already in Party");
165                 } else {
166                     party.add(selectedNick);
167                     partyList.setListData(party);
168                 }
169             }
170         }
171         if (e.getSource().equals(remPatron)) {
172             if (selectedMember != null) {
173                 party.removeElement(selectedMember);
174                 partyList.setListData(party);
175             }
176         }
177         if (e.getSource().equals(newPatron)) {
178             NewPatronView newPatron = new NewPatronView( this );
```

```
179         }
180         if (e.getSource().equals(finished)) {
181             if ( party != null && party.size() > 0) {
182                 alley.updateAddParty( this );
183             }
184             win.hide();
185         }
186 
187     }
188 
189 /**
190  * Handler for List actions
191  * @param e the ListActionEvent that triggered the handler
192  */
193 
194     public void valueChanged(ListSelectionEvent e) {
195         if (e.getSource().equals(allBowlers)) {
196             selectedNick =
197                 ((String) ((JList) e.getSource()).getSelectedValue());
198         }
199         if (e.getSource().equals(partyList)) {
200             selectedMember =
201                 ((String) ((JList) e.getSource()).getSelectedValue());
202         }
203     }
204 
205 /**
206  * Accessor for Party
207  */
208 
209     public Vector getNames() {
210         return party;
211     }
212 
213 /**
214  * Called by NewPatronView to notify AddPartyView to update
215  *
216  * @param newPatron the NewPatronView that called this method
217  */
218 
219     public void updateNewPatron(NewPatronView newPatron) {
220         try {
221             Bowler checkBowler = BowlerFile.getBowlerInfo( newPatron.getNick() );
222             if ( checkBowler == null ) {
223                 BowlerFile.putBowlerInfo(
224                     newPatron.getNick(),
225                     newPatron.getFull(),
226                     newPatron.getEmail());
227                 bowlerdb = new Vector(BowlerFile.getBowlers());
228                 allBowlers.setListData(bowlerdb);
229                 party.add(newPatron.getNick());
230                 partyList.setListData(party);
231             } else {
232                 System.err.println( "A Bowler with that name already exists." );
233             }
234         } catch (Exception e2) {
235             System.err.println("File I/O Error");
236         }
237     }
238 
```

```
239 /**
240  * Accessor for Party
241  */
242 
243     public Vector getParty() {
244         return party;
245     }
246 
247 }
248 
```

File - D:\Users\gjr8050\262refactoring\src\Views\EndGamePrompt.java

```java
package Views;
/**
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

import java.util.*;
import java.text.*;

public class EndGamePrompt implements ActionListener {

    private JFrame win;
    private JButton yesButton, noButton;

    private int result;

    private String selectedNick, selectedMember;

    public EndGamePrompt( String partyName ) {

        result =0;

        win = new JFrame("Another Game for " + partyName + "?" );
        win.getContentPane().setLayout(new BorderLayout());
        ((JPanel) win.getContentPane()).setOpaque(false);

        JPanel colPanel = new JPanel();
        colPanel.setLayout(new GridLayout( 2, 1 ));

        // Label Panel
        JPanel labelPanel = new JPanel();
        labelPanel.setLayout(new FlowLayout());

        JLabel message = new JLabel( "Party " + partyName
            + " has finished bowling.\nWould they like to bowl another game?" );

        labelPanel.add( message );

        // Button Panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(1, 2));

        Insets buttonMargin = new Insets(4, 4, 4, 4);

        yesButton = new JButton("Yes");
        JPanel yesButtonPanel = new JPanel();
        yesButtonPanel.setLayout(new FlowLayout());
        yesButton.addActionListener(this);
        yesButtonPanel.add(yesButton);

        noButton = new JButton("No");
```

File - D:\Users\gjr8050\262refactoring\src\Views\EndGamePrompt.java

```
61          JPanel noButtonPanel = new JPanel();
62          noButtonPanel.setLayout(new FlowLayout());
63          noButton.addActionListener(this);
64          noButtonPanel.add(noButton);
65
66          buttonPanel.add(yesButton);
67          buttonPanel.add(noButton);
68
69          // Clean up main panel
70          colPanel.add(labelPanel);
71          colPanel.add(buttonPanel);
72
73          win.getContentPane().add("Center", colPanel);
74
75          win.pack();
76
77          // Center Window on Screen
78          Dimension screenSize = (Toolkit.getDefaultToolkit()).getScreenSize();
79          win.setLocation(
80              ((screenSize.width) / 2) - ((win.getSize().width) / 2),
81              ((screenSize.height) / 2) - ((win.getSize().height) / 2));
82          win.show();
83
84      }
85
86      public void actionPerformed(ActionEvent e) {
87          if (e.getSource().equals(yesButton)) {
88              result=1;
89          }
90          if (e.getSource().equals(noButton)) {
91              result=2;
92          }
93
94      }
95
96      public int getResult() {
97          while ( result == 0 ) {
98              try {
99                  Thread.sleep(10);
100             } catch ( InterruptedException e ) {
101                 System.err.println( "Interrupted" );
102             }
103         }
104         return result;
105     }
106
107     public void distroy() {
108         win.hide();
109     }
110
111 }
112
113
```

File - D:\Users\gjr8050\262refactoring\src\Views\EndGameReport.java

```
1  package Views;
2  /**
3   *
4   * To change this generated comment edit the template variable "typecomment":
5   * Window>Preferences>Java>Templates.
6   * To enable and disable the creation of type comments go to
7   * Window>Preferences>Java>Code Generation.
8   */
9
10 import Main.Bowler;
11
12 import java.awt.*;
13 import java.awt.event.*;
14 import javax.swing.*;
15 import javax.swing.border.*;
16 import javax.swing.event.*;
17
18 import java.util.*;
19
20 public class EndGameReport implements ActionListener, ListSelectionListener {
21
22     private JFrame win;
23     private JButton printButton, finished;
24     private JList memberList;
25     private Vector myVector;
26     private Vector retVal;
27
28     private int result;
29
30     private String selectedMember;
31
32     public EndGameReport( String partyName, Vector party ) {
33
34         result =0;
35         retVal = new Vector();
36         win = new JFrame("End Game Report for " + partyName + "?" );
37         win.getContentPane().setLayout(new BorderLayout());
38         ((JPanel) win.getContentPane()).setOpaque(false);
39
40         JPanel colPanel = new JPanel();
41         colPanel.setLayout(new GridLayout( 1, 2 ));
42
43         // Member Panel
44         JPanel partyPanel = new JPanel();
45         partyPanel.setLayout(new FlowLayout());
46         partyPanel.setBorder(new TitledBorder("Party Members"));
47
48         Vector myVector = new Vector();
49         Iterator iter = (party).iterator();
50         while (iter.hasNext()){
51             myVector.add( ((Bowler)iter.next()).getNick() );
52         }
53         memberList = new JList(myVector);
54         memberList.setFixedCellWidth(120);
55         memberList.setVisibleRowCount(5);
56         memberList.addListSelectionListener(this);
57         JScrollPane partyPane = new JScrollPane(memberList);
58         //        partyPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS
   );
59         partyPanel.add(partyPane);
```

```
60
61          partyPanel.add( memberList );
62
63          // Button Panel
64          // Button Panel
65          JPanel buttonPanel = new JPanel();
66          buttonPanel.setLayout(new GridLayout(2, 1));
67
68          Insets buttonMargin = new Insets(4, 4, 4, 4);
69
70          printButton = new JButton("Print Report");
71          JPanel printButtonPanel = new JPanel();
72          printButtonPanel.setLayout(new FlowLayout());
73          printButton.addActionListener(this);
74          printButtonPanel.add(printButton);
75
76          finished = new JButton("Finished");
77          JPanel finishedPanel = new JPanel();
78          finishedPanel.setLayout(new FlowLayout());
79          finished.addActionListener(this);
80          finishedPanel.add(finished);
81
82          buttonPanel.add(printButton);
83          buttonPanel.add(finished);
84
85          // Clean up main panel
86          colPanel.add(partyPanel);
87          colPanel.add(buttonPanel);
88
89          win.getContentPane().add("Center", colPanel);
90
91          win.pack();
92
93          // Center Window on Screen
94          Dimension screenSize = (Toolkit.getDefaultToolkit()).getScreenSize();
95          win.setLocation(
96              ((screenSize.width) / 2) - ((win.getSize().width) / 2),
97              ((screenSize.height) / 2) - ((win.getSize().height) / 2));
98          win.show();
99
100     }
101
102     public void actionPerformed(ActionEvent e) {
103         if (e.getSource().equals(printButton)) {
104             //Add selected to the vector.
105             retVal.add(selectedMember);
106         }
107         if (e.getSource().equals(finished)) {
108             win.hide();
109             result = 1;
110         }
111
112     }
113
114     public void valueChanged(ListSelectionEvent e) {
115         selectedMember =
116             ((String) ((JList) e.getSource()).getSelectedValue());
117     }
118
119     public Vector getResult() {
```

```
120         while ( result == 0 ) {
121             try {
122                 Thread.sleep(10);
123             } catch ( InterruptedException e ) {
124                 System.err.println( "Interrupted" );
125             }
126         }
127         return retVal;
128     }
129 
130     public void destroy() {
131         win.hide();
132     }
133 
134     public static void main( String args[] ) {
135         Vector bowlers = new Vector();
136         for ( int i=0; i<4; i++ ) {
137             bowlers.add( new Bowler( "aaaaa", "aaaaa", "aaaaa" ) );
138         }
139         String partyName="wank";
140         EndGameReport e = new EndGameReport( partyName, bowlers );
141     }
142 
143 }
144 
145 
```

File - D:\Users\gjr8050\262refactoring\src\Views\NewPatronView.java

```
1  package Views;
2  /* AddPartyView.java
3   *
4   *  Version
5   *  $Id$
6   *
7   *  Revisions:
8   *      $Log: NewPatronView.java,v $
9   *      Revision 1.3  2003/02/02 16:29:52  ???
10  *      Added AlleyEvent and AlleyObserver. Updated Queue to allow access to Vector so that
    contents could be viewed without destroying. Implemented observer model for most of Alley.
11  *
12  *
13  */
14
15 /**
16  * Class for GUI components need to add a patron
17  *
18  */
19
20 import Views.AddPartyView;
21
22 import java.awt.*;
23 import java.awt.event.*;
24 import javax.swing.*;
25 import javax.swing.border.*;
26
27 public class NewPatronView implements ActionListener {
28
29     private int maxSize;
30
31     private JFrame win;
32     private JButton abort, finished;
33     private JLabel nickLabel, fullLabel, emailLabel;
34     private JTextField nickField, fullField, emailField;
35     private String nick, full, email;
36
37     private boolean done;
38
39     private String selectedNick, selectedMember;
40     private AddPartyView addParty;
41
42     public NewPatronView(AddPartyView v) {
43
44         addParty=v;
45         done = false;
46
47         win = new JFrame("Add Patron");
48         win.getContentPane().setLayout(new BorderLayout());
49         ((JPanel) win.getContentPane()).setOpaque(false);
50
51         JPanel colPanel = new JPanel();
52         colPanel.setLayout(new BorderLayout());
53
54         // Patron Panel
55         JPanel patronPanel = new JPanel();
56         patronPanel.setLayout(new GridLayout(3, 1));
57         patronPanel.setBorder(new TitledBorder("Your Info"));
58
59         JPanel nickPanel = new JPanel();
```

```
60          nickPanel.setLayout(new FlowLayout());
61          nickLabel = new JLabel("Nick Name");
62          nickField = new JTextField("", 15);
63          nickPanel.add(nickLabel);
64          nickPanel.add(nickField);
65
66          JPanel fullPanel = new JPanel();
67          fullPanel.setLayout(new FlowLayout());
68          fullLabel = new JLabel("Full Name");
69          fullField = new JTextField("", 15);
70          fullPanel.add(fullLabel);
71          fullPanel.add(fullField);
72
73          JPanel emailPanel = new JPanel();
74          emailPanel.setLayout(new FlowLayout());
75          emailLabel = new JLabel("E-Mail");
76          emailField = new JTextField("", 15);
77          emailPanel.add(emailLabel);
78          emailPanel.add(emailField);
79
80          patronPanel.add(nickPanel);
81          patronPanel.add(fullPanel);
82          patronPanel.add(emailPanel);
83
84          // Button Panel
85          JPanel buttonPanel = new JPanel();
86          buttonPanel.setLayout(new GridLayout(4, 1));
87
88          Insets buttonMargin = new Insets(4, 4, 4, 4);
89
90          finished = new JButton("Add Patron");
91          JPanel finishedPanel = new JPanel();
92          finishedPanel.setLayout(new FlowLayout());
93          finished.addActionListener(this);
94          finishedPanel.add(finished);
95
96          abort = new JButton("Abort");
97          JPanel abortPanel = new JPanel();
98          abortPanel.setLayout(new FlowLayout());
99          abort.addActionListener(this);
100         abortPanel.add(abort);
101
102         buttonPanel.add(abortPanel);
103         buttonPanel.add(finishedPanel);
104
105         // Clean up main panel
106         colPanel.add(patronPanel, "Center");
107         colPanel.add(buttonPanel, "East");
108
109         win.getContentPane().add("Center", colPanel);
110
111         win.pack();
112
113         // Center Window on Screen
114         Dimension screenSize = (Toolkit.getDefaultToolkit()).getScreenSize();
115         win.setLocation(
116             ((screenSize.width) / 2) - ((win.getSize().width) / 2),
117             ((screenSize.height) / 2) - ((win.getSize().height) / 2));
118         win.show();
119
```

```
120     }
121
122     public void actionPerformed(ActionEvent e) {
123         if (e.getSource().equals(abort)) {
124             done = true;
125             win.hide();
126         }
127
128         if (e.getSource().equals(finished)) {
129             nick = nickField.getText();
130             full = fullField.getText();
131             email = emailField.getText();
132             done = true;
133             addParty.updateNewPatron( this );
134             win.hide();
135         }
136
137     }
138
139     public boolean done() {
140         return done;
141     }
142
143     public String getNick() {
144         return nick;
145     }
146
147     public String getFull() {
148         return full;
149     }
150
151     public String getEmail() {
152         return email;
153     }
154
155 }
156
```

File - D:\Users\gjr8050\262refactoring\src\Views\PinSetterView.java

```
1  package Views;
2  /*
3   * PinSetterView/.java
4   *
5   * Version:
6   *   $Id$
7   *
8   * Revision:
9   *   $Log$
10  */
11
12 /**
13  *  constructs a prototype PinSetter GUI
14  *
15  */
16
17 import Main.Pinsetter;
18
19 import java.awt.*;
20 import javax.swing.*;
21 import java.util.Observable;
22 import java.util.Observer;
23 import java.util.Vector;
24
25
26 public class PinSetterView implements Observer {
27
28
29     private Vector pinVect = new Vector ( );
30     private JPanel firstRoll;
31     private JPanel secondRoll;
32
33     /**
34      * Constructs a Pin Setter GUI displaying which roll it is with
35      * yellow boxes along the top (1 box for first roll, 2 boxes for second)
36      * and displays the pins as numbers in this format:
37      *
38      *                7   8   9   10
39      *                  4   5   6
40      *                    2   3
41      *                      1
42      *
43      */
44
45
46     private JFrame frame;
47
48     public PinSetterView ( int laneNum ) {
49
50     frame = new JFrame ( "Lane " + laneNum + ":" );
51
52     Container cpanel = frame.getContentPane ( );
53
54     JPanel pins = new JPanel ( );
55
56     pins.setLayout ( new GridLayout ( 4, 7 ) );
57
58     //********************Top of GUI indicates first or second roll
59
60     JPanel top = new JPanel ( );
```

```
61
62      firstRoll = new JPanel ( );
63      firstRoll.setBackground( Color.yellow );
64
65      secondRoll = new JPanel ( );
66      secondRoll.setBackground ( Color.black );
67
68      top.add ( firstRoll, BorderLayout.WEST );
69
70      top.add ( secondRoll, BorderLayout.EAST );
71
72      //*****************************************************************
73
74      //**********************Grid of the pins***************************
75
76
77      JPanel one = new JPanel ();
78      JLabel oneL = new JLabel ( "1" );
79      one.add (oneL);
80      JPanel two = new JPanel ();
81      JLabel twoL = new JLabel ( "2" );
82      two.add (twoL);
83      JPanel three = new JPanel ();
84      JLabel threeL = new JLabel ( "3" );
85      three.add (threeL);
86      JPanel four = new JPanel ();
87      JLabel fourL = new JLabel ( "4" );
88      four.add (fourL);
89      JPanel five = new JPanel ();
90      JLabel fiveL = new JLabel ( "5" );
91      five.add (fiveL);
92      JPanel six = new JPanel ();
93      JLabel sixL = new JLabel ( "6" );
94      six.add (sixL);
95      JPanel seven = new JPanel ();
96      JLabel sevenL = new JLabel ( "7" );
97      seven.add (sevenL);
98      JPanel eight = new JPanel ();
99      JLabel eightL = new JLabel ( "8" );
100     eight.add (eightL);
101     JPanel nine = new JPanel ();
102     JLabel nineL = new JLabel ( "9" );
103     nine.add (nineL);
104     JPanel ten = new JPanel ();
105     JLabel tenL = new JLabel ( "10" );
106     ten.add (tenL);
107
108     //This Vector will keep references to the pin labels to show
109     //which ones have fallen.
110
111     pinVect.add ( oneL );
112     pinVect.add ( twoL );
113     pinVect.add ( threeL );
114     pinVect.add ( fourL );
115     pinVect.add ( fiveL );
116     pinVect.add ( sixL );
117     pinVect.add ( sevenL );
118     pinVect.add ( eightL );
119     pinVect.add ( nineL );
120     pinVect.add ( tenL );
```

```
121
122
123     //*****************************Fourth Row**************
124
125     pins.add ( seven );
126     pins.add ( new JPanel ( ) );
127     pins.add ( eight );
128     pins.add ( new JPanel ( ) );
129     pins.add ( nine );
130     pins.add ( new JPanel ( ) );
131     pins.add ( ten );
132
133     //*****************************Third Row***********
134
135     pins.add ( new JPanel ( ) );
136     pins.add ( four );
137     pins.add ( new JPanel ( ) );
138     pins.add ( five );
139     pins.add ( new JPanel ( ) );
140     pins.add ( six );
141
142     //*****************************Second Row**************
143
144     pins.add ( new JPanel ( ) );
145     pins.add ( new JPanel ( ) );
146     pins.add ( new JPanel ( ) );
147     pins.add ( two );
148     pins.add ( new JPanel ( ) );
149     pins.add ( three );
150     pins.add ( new JPanel ( ) );
151     pins.add ( new JPanel ( ) );
152
153     //*****************************First Row*****************
154
155     pins.add ( new JPanel ( ) );
156     pins.add ( new JPanel ( ) );
157     pins.add ( new JPanel ( ) );
158     pins.add ( one );
159     pins.add ( new JPanel ( ) );
160     pins.add ( new JPanel ( ) );
161     pins.add ( new JPanel ( ) );
162     //*********************************************************
163
164     top.setBackground ( Color.black );
165
166     cpanel.add ( top, BorderLayout.NORTH );
167
168     pins.setBackground ( Color.black );
169     pins.setForeground ( Color.yellow );
170
171     cpanel.add ( pins, BorderLayout.CENTER );
172
173     frame.pack();
174
175
176 //  frame.show();
177     }
178
179     public void show() {
180         frame.show();
```

```
181     }
182
183     public void hide() {
184         frame.hide();
185     }
186
187     public static void main ( String args [ ] ) {
188         PinSetterView pg = new PinSetterView ( 1 );
189     }
190
191     @Override
192     public void update(Observable o, Object arg) {
193         Pinsetter pe = (Pinsetter)o;
194         int pinsDownThisThrow = (Integer) arg;
195         if ( !(pe.isFoulCommited()) ) {
196             JLabel tempPin = new JLabel ( );
197             for ( int c = 0; c < 10; c++ ) {
198                 boolean pin = pe.pinKnockedDown ( c );
199                 tempPin = (JLabel)pinVect.get ( c );
200                 if ( pin ) {
201                     tempPin.setForeground ( Color.lightGray );
202                 }
203             }
204         }
205         if ( pe.getThrowNumber() == 1 ) {
206             secondRoll.setBackground ( Color.yellow );
207         }
208         if ( pinsDownThisThrow == -1) {
209             for ( int i = 0; i != 10; i++){
210                 ((JLabel)pinVect.get(i)).setForeground(Color.black);
211             }
212             secondRoll.setBackground( Color.black);
213         }
214     }
215 }
216
```

File - D:\Users\gjr8050\262refactoring\src\Views\LaneStatusView.java

```
1  package Views;
2  /**
3   *
4   * To change this generated comment edit the template variable "typecomment":
5   * Window>Preferences>Java>Templates.
6   * To enable and disable the creation of type comments go to
7   * Window>Preferences>Java>Code Generation.
8   */
9
10 import Main.*;
11
12 import java.awt.*;
13 import java.awt.event.*;
14 import java.util.Observable;
15 import java.util.Observer;
16 import javax.swing.*;
17
18 public class LaneStatusView implements ActionListener, Observer {
19
20     private JPanel jp;
21
22     private JLabel curBowler, foul, pinsDown;
23     private JButton viewLane;
24     private JButton viewPinSetter, maintenance;
25
26     private PinSetterView psv;
27     private LaneView lv;
28     private Lane lane;
29     int laneNum;
30
31     boolean laneShowing;
32     boolean psShowing;
33
34     public LaneStatusView(Lane lane, int laneNum ) {
35
36         this.lane = lane;
37         this.laneNum = laneNum;
38
39         laneShowing=false;
40         psShowing=false;
41
42         psv = new PinSetterView( laneNum );
43         Pinsetter ps = lane.getPinsetter();
44         ps.addObserver(psv);
45
46         lv = new LaneView( lane, laneNum );
47         lane.addObserver(lv);
48
49
50         jp = new JPanel();
51         jp.setLayout(new FlowLayout());
52         JLabel cLabel = new JLabel( "Now Bowling: " );
53         curBowler = new JLabel( "(no one)" );
54         JLabel fLabel = new JLabel( "Foul: " );
55         foul = new JLabel( " " );
56         JLabel pdLabel = new JLabel( "Pins Down: " );
57         pinsDown = new JLabel( "0" );
58
59         // Button Panel
60         JPanel buttonPanel = new JPanel();
```

```
61          buttonPanel.setLayout(new FlowLayout());
62
63          Insets buttonMargin = new Insets(4, 4, 4, 4);
64
65          viewLane = new JButton("View Lane");
66          JPanel viewLanePanel = new JPanel();
67          viewLanePanel.setLayout(new FlowLayout());
68          viewLane.addActionListener(this);
69          viewLanePanel.add(viewLane);
70
71          viewPinSetter = new JButton("Pinsetter");
72          JPanel viewPinSetterPanel = new JPanel();
73          viewPinSetterPanel.setLayout(new FlowLayout());
74          viewPinSetter.addActionListener(this);
75          viewPinSetterPanel.add(viewPinSetter);
76
77          maintenance = new JButton("     ");
78          maintenance.setBackground( Color.GREEN );
79          JPanel maintenancePanel = new JPanel();
80          maintenancePanel.setLayout(new FlowLayout());
81          maintenance.addActionListener(this);
82          maintenancePanel.add(maintenance);
83
84          viewLane.setEnabled( false );
85          viewPinSetter.setEnabled( false );
86
87
88          buttonPanel.add(viewLanePanel);
89          buttonPanel.add(viewPinSetterPanel);
90          buttonPanel.add(maintenancePanel);
91
92          jp.add( cLabel );
93          jp.add( curBowler );
94  //      jp.add( fLabel );
95  //      jp.add( foul );
96          jp.add( pdLabel );
97          jp.add( pinsDown );
98
99          jp.add(buttonPanel);
100
101     }
102
103     public JPanel showLane() {
104         return jp;
105     }
106
107     public void actionPerformed( ActionEvent e ) {
108         if ( lane.isPartyAssigned() ) {
109             if (e.getSource().equals(viewPinSetter)) {
110                 if ( psShowing == false ) {
111                     psv.show();
112                     psShowing=true;
113                 } else if ( psShowing == true ) {
114                     psv.hide();
115                     psShowing=false;
116                 }
117             }
118         }
119         if (e.getSource().equals(viewLane)) {
120             if ( lane.isPartyAssigned() ) {
```

File - D:\Users\gjr8050\262refactoring\src\Views\LaneStatusView.java

```
121                 if ( laneShowing == false ) {
122                     lv.show();
123                     laneShowing=true;
124                 } else if ( laneShowing == true ) {
125                     lv.hide();
126                     laneShowing=false;
127                 }
128             }
129         }
130         if (e.getSource().equals(maintenance)) {
131             if ( lane.isPartyAssigned() ) {
132                 lane.unPauseGame();
133                 maintenance.setBackground( Color.GREEN );
134             }
135         }
136     }
137 
138     @Override
139     public void update(Observable o, Object arg) {
140         if(o instanceof Lane){
141             Lane newLane = (Lane)o;
142 
143             curBowler.setText( ( (Bowler)newLane.getBowler()).getNickName() );
144             if ( newLane.isMechanicalProblem() ) {
145                 maintenance.setBackground( Color.RED );
146             }
147             if ( lane.isPartyAssigned() == false ) {
148                 viewLane.setEnabled( false );
149                 viewPinSetter.setEnabled( false );
150             } else {
151                 viewLane.setEnabled( true );
152                 viewPinSetter.setEnabled( true );
153             }
154         }
155         else if(o instanceof Pinsetter){
156             pinsDown.setText( ( new Integer(((Pinsetter)o).totalPinsDown()) ).toString() );
157         }
158     }
159 }
160 
```