CS22510 - Assignment 1
Runners and Riders - "Out and About"

Samuel Jackson
slj11@aber.ac.uk

March 19, 2013

1 Event Creation Program Documentation

1.1 Code Listing

The following section provides the full code listing for the event creation program. This application is written using C++. Doxygen documentation is available via the provided CD.

Listing 1: eventcreator.h

```
1  /**
2   * @file eventcreator.h
3   * @author Samuel Jackson (slj11@aber.ac.uk)
4   * @date 09 March 2013
5   * @brief class to create courses, entrants and events.
6   */
7
8  #ifndef MENU_H
9  #define MENU_H
10
11 #include <vector>
12 #include "ioscanner.h"
13 #include "fileio.h"
14 #include "event.h"
15
16 class EventCreator {
17 public:
18     EventCreator();
19     virtual ~EventCreator();
20
21     void ShowMainMenu();
22 private:
23     FileIO fio;
24     IOScanner scanner;
25     std::vector<Event> events;
26
27     void MakeEvent();
28     void AddEntrants();
29     void CreateCourse();
30     int ChooseEvent();
31     char ChooseCourse(Event event);
32     void ViewEvent();
33 };
34
35 #endif /* MENU_H */
```

Listing 2: eventcreator.cpp

```
/**
 * @file eventcreator.cpp
 * @author Samuel Jackson (slj11@aber.ac.uk)
 * @date 09 March 2013
 * @brief class to create courses, entrants and events.
 * Also outputs and handles user navigation between menus.
 */

#include <iostream>
#include <string>
#include <ctime>
#include <algorithm>

#include "ioscanner.h"
#include "eventcreator.h"
#include "fileio.h"
#include "event.h"

/**
 * Initialises the event creator program and outputs startup message
 */
EventCreator::EventCreator() {
    using namespace std;

    cout << "------------------------" << endl;
    cout << "EVENT CREATION PROGRAM" << endl;
    cout << "------------------------" << endl << endl;
}

/**
 * Displays the main menu to the user and processes users choice
 */
void EventCreator::ShowMainMenu() {
    using namespace std;
    int input = 0;

    do {
        cout << "MAIN MENU" << endl;
        cout << "--------------------------" << endl;
        cout << "Enter an option: " << endl;
        cout << "1 - Make new event" << endl;
        cout << "2 - Add entrants to event" << endl;
        cout << "3 - Create course for event" << endl;
        cout << "4 - Write an event to file" << endl;
        cout << "5 - View an event in the system" << endl;
        cout << "6 - Exit Program" << endl;

        input = scanner.ReadInt();
        int evt_index;
        switch(input) {
            case 1:
                MakeEvent();
                break;
            case 2:
                AddEntrants();
                break;
            case 3:
                CreateCourse();
                break;
            case 4: //save event to file
                evt_index = ChooseEvent();
                if(evt_index >= 0) {
                    Event e = events[evt_index];
                    fio.WriteEvent(e);
                }
                break;
            case 5:
                ViewEvent();
                break;
        }

```

```
72      } while (input != 6);
73  }
74
75  /**
76   * Member function to create a new event on the system.
77   */
78  void EventCreator::MakeEvent() {
79      using namespace std;
80      string evt_name;
81      tm date, time;
82
83      cout << "Enter name of event:" << endl;
84      evt_name = scanner.ReadString(80);
85
86      cout << "Enter event date (DD/MM/YY):" << endl;
87      date = scanner.ReadDate();
88
89      cout << "Enter event start time (HH:MM):" << endl;
90      time = scanner.ReadTime();
91
92      cout << "Enter location of nodes file for event:" << endl;
93      string nodesfile = scanner.ReadString(100);
94      vector<int> nodes = fio.ReadNodesList(nodesfile);
95
96      Event e(evt_name, date, time);
97      e.SetNodes(nodes);
98      events.push_back(e);
99  }
100
101 /**
102  * Member function to add a new entrant to an event.
103  */
104 void EventCreator::AddEntrants() {
105     using namespace std;
106     int eventIndex = ChooseEvent();
107     int numEntrants = 0;
108     string name;
109     int id;
110     char course;
111
112     //if user picked an event
113     if(eventIndex >= 0) {
114         Event event = events[eventIndex];
115
116         //check if we have some courses already.
117         if(event.GetCourses().size() > 0) {
118             cout << "Enter number of entrants to add: " << endl;
119
120             do {
121                 numEntrants = scanner.ReadInt();
122                 if(numEntrants <=0) {
123                     cout << "Not a valid number of entrants" << endl;
124                 } else if (numEntrants > 50) {
125                     cout << "Too many entrants to create at once!" << endl;
126                 }
127             } while (numEntrants <= 0);
128
129             for(int i = 0; i < numEntrants; i++) {
130                 cout << "Enter entrant's name: " << endl;
131                 name = scanner.ReadString(50);
132                 course = ChooseCourse(event);
133                 id = event.GetEntrants().size()+1;
134                 event.AddEntrant(name, id, course);
135                 events[eventIndex] = event;
136             }
137         } else {
138             cout << "You must create at least one course first." << endl;
139         }
140     }
141 }
142
```

```
143 /**
144  * Choose an event to work with if there are events on the system.
145  * @return the id of the chosen event
146  */
147 int EventCreator::ChooseEvent() {
148     using namespace std;
149     int index = -1;
150     bool validChoice = false;
151 
152     if(events.size() > 0 ) {
153         cout << "Please choose an event:" << endl;
154         for(std::vector<int>::size_type i = 0; i != events.size(); i++) {
155             cout << i << " - " << events[i].GetName() << endl;
156         }
157 
158          do {
159             index = scanner.ReadInt();
160             if (index >= 0 && index < events.size()) {
161                 validChoice = true;
162             } else {
163                 cout << "Not a valid event choice." << endl;
164             }
165         } while(!validChoice);
166 
167     } else {
168         cout << "You must create at least one event first." << endl;
169     }
170 
171     return index;
172 }
173 
174 /**
175  * Choose a course based on the selected event
176  * @param event the currently selected event
177  * @return the id of the chosen course
178  */
179 char EventCreator::ChooseCourse(Event event) {
180     using namespace std;
181     bool validChoice = false;
182     int index;
183     char choice;
184     std::vector<Course> courses = event.GetCourses();
185 
186     if(courses.size() > 0 ) {
187         cout << "Please choose course for the entrant:" << endl;
188         for(std::vector<int>::size_type i = 0; i != courses.size(); i++) {
189             cout << i << " - " << courses[i].GetId() << endl;
190         }
191 
192          do {
193             index = scanner.ReadInt();
194             if (index >= 0 && index < courses.size()) {
195                 validChoice = true;
196             } else {
197                 cout << "Not a valid course choice." << endl;
198             }
199         } while(!validChoice);
200         choice = courses[index].GetId();
201     } else {
202         cout << "You must create at least one course first." << endl;
203     }
204 
205     return choice;
206 }
207 
208 /**
209  * Create a course based on the selected event
210  */
211 void EventCreator::CreateCourse() {
212     using namespace std;
213     int eventIndex = ChooseEvent();
```

```
214     int node;
215     vector<int> courseNodes;
216     vector<int> allowedNodes;
217     if(eventIndex >= 0) {
218         Event event = events[eventIndex];
219         allowedNodes = event.GetNodes();
220 
221         if(event.GetCourses().size() <= 26) {
222             cout << "Enter nodes for course. Enter 0 to finish: " << endl;
223 
224             do {
225                 node = scanner.ReadInt();
226                 if(find(allowedNodes.begin(), allowedNodes.end(), node)!=allowedNodes.end()) {
227                     courseNodes.push_back(node);
228                 } else if (node != 0) {
229                     cout << "Not a valid node number!" << endl;
230                 }
231             } while(node != 0);
232 
233             //convert numerical index to character index
234             // e.g. ASCII 'A' is 65, 'B' is 66 etc.
235             char id = (int)event.GetCourses().size()+65;
236 
237             event.AddCourse(id, courseNodes);
238             events[eventIndex] = event;
239 
240         } else {
241             cout << "Events can not have more than 26 courses" << endl;
242         }
243     }
244 }
245 
246 /**
247  * View an event on the system. This will list all course and
248  * entrants associated with the chosen event.
249  */
250 void EventCreator::ViewEvent() {
251     using namespace std;
252     int eventIndex = ChooseEvent();
253     if(eventIndex >= 0) {
254         Event event = events[eventIndex];
255 
256         cout << "----------------------------------------" << endl;
257         cout << event.GetName() << endl;
258         cout << event.GetFormattedDate() << endl;
259         cout << event.GetFormattedTime() << endl;
260         cout << "----------------------------------------" << endl;
261         cout << "COURSES" << endl;
262         cout << "----------------------------------------" << endl;
263 
264         if(event.GetCourses().size() > 0) {
265             for(vector<Course>::iterator it = event.GetCourses().begin();
266                     it != event.GetCourses().end(); ++it) {
267                 cout << it->GetId() << " ";
268                 cout << it->GetNodes().size() << " " << endl;
269 
270                 vector<int> nodes(it->GetNodes());
271                 for(vector<int>::iterator jt = nodes.begin();
272                         jt != nodes.end(); ++jt) {
273                     cout << *jt << endl;
274                 }
275 
276                 cout << endl;
277             }
278         } else {
279             cout << "This event has no courses yet!" << endl;
280         }
281 
282         cout << "----------------------------------------" << endl;
283         cout << "ENRTANTS" << endl;
284         cout << "----------------------------------------" << endl;
```

```
285
286             if(event.GetEntrants().size() > 0) {
287                 for (vector<Entrant>::iterator it = event.GetEntrants().begin();
288                         it != event.GetEntrants().end(); ++it) {
289                     cout << it->GetId() << " " << it->GetCourse() << " ";
290                     cout << it->GetName() << endl;
291                 }
292             } else {
293                 cout << "This event has no entrants yet!" << endl;
294             }
295         }
296     }
297
298     EventCreator::~EventCreator() {
299     }
300
301     /**
302      * Main method and application entry point.
303      * Simply shows the main menu.
304      *
305      * @param argc the number of command line arguments
306      * @param argv the char array of command line arguments
307      * @return program exit status (0)
308      */
309     int main(int argc, char** argv) {
310         EventCreator ec;
311         ec.ShowMainMenu();
312         return 0;
313     }
```

Listing 3: event.h

```
1   /**
2    * @file event.h
3    * @author Samuel Jackson (slj11@aber.ac.uk)
4    * @date 09 March 2013
5    * @brief class to hold data about an event.
6    */
7
8   #ifndef EVENT_H
9   #define EVENT_H
10
11  #include <string>
12  #include <vector>
13
14  #include "entrant.h"
15  #include "course.h"
16
17  class Event {
18      public:
19          Event(std::string name, tm date, tm time);
20          virtual ~Event();
21
22          void AddEntrant(std::string name, int id, char course);
23          void AddCourse(char id, std::vector<int> nodes);
24          void SetCourses(std::vector<Course> courses);
25          std::vector<Course> GetCourses() const;
26          void SetEntrants(std::vector<Entrant> entrants);
27          std::vector<Entrant> GetEntrants() const;
28          void SetName(std::string name);
29          std::string GetName() const;
30          void SetDate(tm date);
31          tm GetDate() const;
32          void SetTime(tm time);
33          tm GetTime() const;
34          void SetNodes(std::vector<int> nodes);
35          std::vector<int> GetNodes() const;
36
37          std::string GetFormattedDate();
38          std::string GetFormattedTime();
39      private:
```

```
40          tm time;
41          tm date;
42          std::string name;
43          std::vector<Entrant> entrants;
44          std::vector<Course> courses;
45          std::vector<int> nodes;
46
47          std::string GetDayPostfix(int day);
48  };
49
50  #endif /* EVENT_H */
```

Listing 4: event.cpp

```
1   /**
2    * @file event.cpp
3    * @author Samuel Jackson (slj11@aber.ac.uk)
4    * @date 09 March 2013
5    * @brief class to hold data about an event.
6    */
7
8   #include <string>
9   #include <sstream>
10  #include "event.h"
11  #include "entrant.h"
12
13  /**
14   * Create a new event and initilise it with a name, date and time.
15   * @param name the name of the event
16   * @param date the date of the event
17   * @param time the time of the event
18   */
19  Event::Event(std::string name, tm date, tm time) {
20      this->time = time;
21      this->date = date;
22      this->name = name;
23  }
24
25  Event::~Event() {
26  }
27
28  /**
29   * Add an entrant to this event.
30   * @param name the name of the entrant
31   * @param id the id of the entrant
32   * @param course the if of the entrant's course
33   */
34  void Event::AddEntrant(std::string name, int id, char course) {
35      Entrant entrant(id, name, course);
36      entrants.push_back(entrant);
37  }
38
39  /**
40   * Add a course to this event.
41   * @param id the id of the course
42   * @param nodes the vector of nodes for the course
43   */
44  void Event::AddCourse(char id, std::vector<int> nodes) {
45      Course course(id, nodes);
46      courses.push_back(course);
47  }
48
49  /**
50   * Set the list of courses for this event
51   * @param courses the vector of courses for an event
52   */
53  void Event::SetCourses(std::vector<Course> courses) {
54      this->courses = courses;
55  }
56
57  /**
```

```
58   * Get the list of courses for this event
59   * @return the vector of courses for an event
60   */
61  std::vector<Course> Event::GetCourses() const {
62      return courses;
63  }
64
65  /**
66   * Set the list of entrants for this event
67   * @param entrants the vector of entrants for an event
68   */
69  void Event::SetEntrants(std::vector<Entrant> entrants) {
70      this->entrants = entrants;
71  }
72
73  /**
74   * Get the list of entrants for this event
75   * @return the vector of entrants for an event
76   */
77  std::vector<Entrant> Event::GetEntrants() const {
78      return entrants;
79  }
80
81  /**
82   * Set the name of this event
83   * @param name the name of this event
84   */
85  void Event::SetName(std::string name) {
86      this->name = name;
87  }
88
89  /**
90   * Get the name of this event
91   * @return the name of this event
92   */
93  std::string Event::GetName() const {
94      return name;
95  }
96
97  /**
98   * Set the date of this event
99   * @param date the date of this event
100  */
101 void Event::SetDate(tm date) {
102     this->date = date;
103 }
104
105 /**
106  * Get the date of this event
107  * @return the date of this event
108  */
109 tm Event::GetDate() const {
110     return date;
111 }
112
113 /**
114  * Set the time of this event
115  * @param time the time of this event
116  */
117 void Event::SetTime(tm time) {
118     this->time = time;
119 }
120
121 /**
122  * Get the time of this event
123  * @return the time of this event
124  */
125 tm Event::GetTime() const {
126     return time;
127 }
128
```

```
129 /**
130  * Set the list of nodes for this event
131  * @param nodes the vector of nodes for this event
132  */
133 void Event::SetNodes(std::vector<int> nodes) {
134     this->nodes = nodes;
135 }
136
137 /**
138  * Get the list of nodes for this event
139  * @return the vector of nodes for this event
140  */
141 std::vector<int> Event::GetNodes() const {
142     return nodes;
143 }
144
145 /**
146  * Get the date of the event as a string in a long format
147  * e.g. 1st February 2012
148  * @return the date formatted and as a string
149  */
150 std::string Event::GetFormattedDate() {
151     using namespace std;
152     ostringstream outputDate;
153     char monthname[10];
154     char year[5];
155
156     strftime(monthname, 10, "%B", &date);
157     strftime(year, 5, "%Y", &date);
158
159     outputDate << date.tm_mday;
160     outputDate << GetDayPostfix(date.tm_mday) << " ";
161     outputDate << monthname;
162     outputDate << " ";
163     outputDate << year;
164
165     return outputDate.str();
166 }
167
168 /**
169  * Get the time of the event as a string
170  * e.g. 17:45
171  * @return the time as a string
172  */
173 std::string Event::GetFormattedTime() {
174     using namespace std;
175     ostringstream timeString;
176     char outputTime [6];
177
178     strftime(outputTime, 6, "%R", &time);
179     timeString << outputTime;
180
181     return timeString.str();
182 }
183
184
185 /**
186  * Member function to get the postfix of the date's day
187  * will return a string with either 'st', 'nd' or 'rd'.
188  * @param day the day to get the postfix for
189  * @return the postfix for the date's day
190  */
191 std::string Event::GetDayPostfix(int day) {
192     std::string postfix = "th";
193     switch(day) {
194         case 1:
195         case 21:
196         case 31:
197             postfix = "st";
198             break;
199         case 2:
```

```
200             case 22:
201                 postfix = "nd";
202                 break;
203             case 3:
204             case 23:
205                 postfix = "rd";
206                 break;
207         }
208 
209         return postfix;
210     }
```

Listing 5: entrant.h

```
1   /**
2    * @file entrant.cpp
3    * @author Samuel Jackson (slj11@aber.ac.uk)
4    * @date 09 March 2013
5    * @brief class to hold data about an entrant in an event.
6    */
7   
8   #ifndef ENTRANT_H
9   #define ENTRANT_H
10  
11  #include <string>
12  
13  class Entrant {
14      public:
15          Entrant(int id, std::string name, char course);
16          virtual ~Entrant();
17  
18          void SetCourse(char course);
19          char GetCourse() const;
20          void SetName(std::string name);
21          std::string GetName() const;
22          void SetId(int id);
23          int GetId() const;
24      private:
25          int id;
26          std::string name;
27          char course;
28  };
29  
30  #endif /* ENTRANT_H */
```

Listing 6: entrant.cpp

```
1   /**
2    * @file entrant.cpp
3    * @author Samuel Jackson (slj11@aber.ac.uk)
4    * @date 09 March 2013
5    * @brief class to hold data about an entrant in an event.
6    */
7   
8   #include "entrant.h"
9   
10  /**
11   * Initilises a new instance of an entrant with an ID,
12   * name and course.
13   *
14   * @param id the ID of the entrant
15   * @param name the name of entrant
16   * @param course the ID of the course the entrant is registered on.
17   */
18  Entrant::Entrant(int id, std::string name, char course) {
19      SetId(id);
20      SetName(name);
21      SetCourse(course);
22  }
23  
```

```
24 Entrant::~Entrant() {
25 }
26
27 /**
28  * Set the course the entrant is on.
29  * @param course the course id
30  */
31 void Entrant::SetCourse(char course) {
32     this->course = course;
33 }
34
35 /**
36  * Get the course the entrant is on.
37  * @return the course id
38  */
39 char Entrant::GetCourse() const {
40     return course;
41 }
42
43 /**
44  * Set the name of the entrant.
45  * @param name the name of the entrant
46  */
47 void Entrant::SetName(std::string name) {
48     this->name = name;
49 }
50
51 /**
52  * Get the name of the entrant.
53  * @return the name of the entrant
54  */
55 std::string Entrant::GetName() const {
56     return name;
57 }
58
59 /**
60  * Set the entrant's ID.
61  * @param id the entrant id
62  */
63 void Entrant::SetId(int id) {
64     this->id = id;
65 }
66
67 /**
68  * Get the entrant's ID.
69  * @return the id of the entrant
70  */
71 int Entrant::GetId() const {
72     return id;
73 }
```

Listing 7: course.h

```
1  /**
2   * @file course.cpp
3   * @author Samuel Jackson (slj11@aber.ac.uk)
4   * @date 09 March 2013
5   * @brief class to hold data about a course in an event.
6   */
7
8  #ifndef COURSE_H
9  #define COURSE_H
10
11 #include <vector>
12
13 class Course {
14     public:
15         Course(char id, std::vector<int> nodes);
16         virtual ~Course();
17
18         void SetNodes(std::vector<int> nodes);
```

```
19          std::vector<int> GetNodes() const;
20          void SetId(char id);
21          char GetId() const;
22      private:
23          char id;
24          std::vector<int> nodes;
25  };
26
27  #endif /* COURSE_H */
```

Listing 8: course.cpp

```
1   /**
2    * @file course.cpp
3    * @author Samuel Jackson (slj11@aber.ac.uk)
4    * @date 09 March 2013
5    * @brief class to hold data about a course in an event.
6    */
7
8   #include "course.h"
9
10  /**
11   * Initialises an instance of a course with an id
12   * and a set of nodes
13   * @param id the id of the course
14   * @param nodes the nodes in the course
15   */
16  Course::Course(char id, std::vector<int> nodes) {
17      SetId(id);
18      SetNodes(nodes);
19  }
20
21  Course::~Course() {
22  }
23
24  /**
25   * Set the list of nodes in this course
26   * @param nodes the vector of nodes.
27   */
28  void Course::SetNodes(std::vector<int> nodes) {
29      this->nodes = nodes;
30  }
31
32  /**
33   * Get the list of nodes in this course
34   * @return the vector of nodes.
35   */
36  std::vector<int> Course::GetNodes() const {
37      return nodes;
38  }
39
40  /**
41   * Set the ID of this course
42   * @param id the ID of the course
43   */
44  void Course::SetId(char id) {
45      this->id = id;
46  }
47
48  /**
49   * Set the list of nodes in this course
50   * @return the ID of the course.
51   */
52  char Course::GetId() const {
53      return id;
54  }
```

Listing 9: fileio.h

```
1   /**
```

```
 * @file fileio.h
 * @author Samuel Jackson (slj11@aber.ac.uk)
 * @date 09 March 2013
 * @brief class to read in data files and write out the created event.
 */

#ifndef FILEIO_H
#define FILEIO_H

#include <vector>
#include <string>

#include "event.h"
#include "entrant.h"
#include "course.h"

class FileIO {
public:
    FileIO();
    virtual ~FileIO();

    void WriteEvent(Event event);
    std::vector<int> ReadNodesList(std::string filename);
private:
    bool WriteCoursesFile(std::string filename, std::vector<Course> courses);
    bool WriteEntrantsFile(std::string filename, std::vector<Entrant> entrants);
    bool WriteEventFile(std::string filename, Event event);
};

#endif /* FILEIO_H */
```

Listing 10: fileio.cpp

```
/**
 * @file fileio.cpp
 * @author Samuel Jackson (slj11@aber.ac.uk)
 * @date 09 March 2013
 * @brief class to read in data files and write out the created event.
 */

#include <iostream>
#include <fstream>
#include <stdlib.h>
#include <sys/stat.h>

#include "fileio.h"
#include "entrant.h"
#include "course.h"
#include "event.h"

FileIO::FileIO() {
}

/**
 * Write an event to file. This makes the courses, entrants and
 * name files.
 * @param evt the event to be written to file
 */
void FileIO::WriteEvent(Event evt) {
    mkdir (evt.GetName().c_str(), 0755);

    WriteEventFile(evt.GetName() + "/name.txt", evt);
    WriteCoursesFile(evt.GetName() + "/courses.txt", evt.GetCourses());
    WriteEntrantsFile(evt.GetName() + "/entrants.txt", evt.GetEntrants());

}

/**
 * Member function to write a vector of courses to a file
 * @param filename the name and path to create the file
 * @param courses the vector of courses to write to file
```

```
39   * @return whether the write operation was successful
40   */
41  bool FileIO::WriteCoursesFile(std::string filename,
42          std::vector<Course> courses) {
43      using namespace std;
44      ofstream out(filename.c_str());
45      bool success = false;
46
47      if(out.is_open()) {
48          for(std::vector<Course>::iterator it = courses.begin();
49                  it != courses.end(); ++it) {
50              out << it->GetId() << " ";
51              out << it->GetNodes().size() << " ";
52
53              std::vector<int> nodes = it->GetNodes();
54              for(std::vector<int>::iterator jt = nodes.begin();
55                      jt != nodes.end(); ++jt) {
56                  out << *jt << " ";
57              }
58
59              out << endl;
60          }
61      }
62
63      return success;
64  }
65
66  /**
67   * Member function to write a vector of entrants to a file
68   * @param filename the name and path to create the file
69   * @param entrants the vector of entrants to write to file
70   * @return whether the write operation was successful
71   */
72  bool FileIO::WriteEntrantsFile(std::string filename,
73          std::vector<Entrant> entrants) {
74      using namespace std;
75      ofstream out(filename.c_str());
76      bool success = false;
77      if(out.is_open()) {
78          for(std::vector<Entrant>::iterator it = entrants.begin();
79                  it != entrants.end(); ++it) {
80              out << it->GetId() << " ";
81              out << it->GetCourse() << " ";
82              out << it->GetName() << endl;
83          }
84
85          out.close();
86          success = true;
87      }
88
89      return success;
90  }
91
92  /**
93   * Member function to read in a list of nodes for a given file
94   * @param filename the name and path to the nodes file
95   * @return vector of nodes read in from file.
96   */
97  std::vector<int> FileIO::ReadNodesList(std::string filename) {
98      using namespace std;
99      string input = "";
100     ifstream in(filename.c_str());
101     int number;
102     char buffer[5];
103     int line = 0;
104     vector<int> nodes;
105
106     if(in.is_open()) {
107         while(!in.eof()) {
108             line++;
109             getline(in, input);
```

```
110             int matches = sscanf (input.c_str(),"%d %s", &number, buffer);
111             if(matches != 2) {
112                 cout << "Error parsing nodes file on line: " << line << endl;
113                 exit(-1);
114             }
115 
116             nodes.push_back(number);
117         }
118     }
119 
120     in.close();
121 
122     return nodes;
123 
124 }
125 
126 /**
127  * Member function to write an event to a file
128  * @param filename the name and path to create the file
129  * @param event the event to write to file
130  * @return whether the write operation was successful
131  */
132 bool FileIO::WriteEventFile(std::string filename, Event event) {
133     using namespace std;
134     ofstream out(filename.c_str());
135 
136     string name = event.GetName();
137     string date = event.GetFormattedDate();
138     string time = event.GetFormattedTime();
139 
140     if (out.is_open()) {
141 
142         out << name << endl;
143         out << date << endl;
144         out << time << endl;
145 
146         out.close();
147         return true;
148     } else {
149         return false;
150     }
151 
152 }
153 
154 FileIO::~FileIO() {
155 }
```

Listing 11: ioscanner.h

```
1 
2  /**
3   * @file ioscanner.h
4   * @author Samuel Jackson (slj11@aber.ac.uk)
5   * @date 09 March 2013
6   * @brief class to read user input in from the command line in a variety of formats.
7   */
8 
9  #ifndef IOSCANNER_H
10 #define IOSCANNER_H
11 
12 #include <string>
13 
14 class IOScanner {
15 public:
16     IOScanner();
17     virtual ~IOScanner();
18 
19     int ReadInt();
20     std::string ReadString(int limit);
21     tm ReadDate();
22     tm ReadTime();
```

```
};

#endif /* CONSOLE_INPUT_H */
```

Listing 12: ioscanner.cpp

```
/**
 * @file ioscanner.cpp
 * @author Samuel Jackson (slj11@aber.ac.uk)
 * @date 09 March 2013
 * @brief class to read user input in from the command line in a variety of formats.
 */

#include <iostream>
#include <limits>
#include <string>
#include <iostream>
#include <locale>

#include "ioscanner.h"

IOScanner::IOScanner() {
}

/**
 * Member function to read a single integer from standard in.
 * @return The integer that was read in
 */
int IOScanner::ReadInt() {
    using namespace std;

    int input;
    while (!(cin >> input)) {
        cout << "Input wasn't a number!\n" ;
        cin.clear();
        cin.ignore(std::numeric_limits<streamsize>::max(), '\n') ;
    }
    cin.ignore(std::numeric_limits<streamsize>::max(), '\n');

    return input;
}

/**
 * Member function to read a string from standard in.
 * @param limit the limit of the number of characters to read in.
 * @return The string that was read in
 */
std::string IOScanner::ReadString(int limit) {
    using namespace std;
    string input = "";

    do {
        getline(cin, input);

        if(input.size() >= limit) {
            cout << "Input too long!" << endl;
        }
    } while(input.size() >= limit);

    return input;
}

/**
 * Member function to read a date from standard in. Dates must be entered in
 * the format DD/MM/YY
 * @return time structure containing the date that was read in
 */
tm IOScanner::ReadDate() {
    using namespace std;
    string date;
    tm when;
```

```
66      bool valid;
67
68      do {
69          valid = true;
70          date = ReadString(10);
71
72          if(!strptime(date.c_str(), "%d/%m/%y", &when)) {
73              cout << "That wasn't a date!\n" << endl;
74              valid = false;
75          }
76      } while (!valid);
77
78      return when;
79  }
80
81  /**
82   * Member function to read a time from standard in. Dates must be entered in
83   * the format HH:mm
84   * @return time structure containing the time that was read in
85   */
86  tm IOScanner::ReadTime() {
87      using namespace std;
88      string time;
89      tm when;
90      bool valid;
91      do {
92          valid = true;
93          time = ReadString(7);
94
95          if(!strptime(time.c_str(), "%R", &when)) {
96              cout << "That wasn't a time!" << endl;
97              valid = false;
98          }
99      } while(!valid);
100
101
102     return when;
103 }
104
105 IOScanner::~IOScanner() {
106 }
```

1.2 Compilation Output

The following section shows the build log generated by Eclipse when compiling the Event Creation program.

Listing 13: Build log of the C++ Event Creation Program

```
12:22:50 **** Build of configuration Debug for project Event Creator ****
make all
Building file: ../course.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"course.d" -MT"course.d" -o "course.o" "../course.cpp"
Finished building: ../course.cpp

Building file: ../entrant.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"entrant.d" -MT"entrant.d" -o "entrant.o" "../entrant.cpp"
Finished building: ../entrant.cpp

Building file: ../event.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"event.d" -MT"event.d" -o "event.o" "../event.cpp"
Finished building: ../event.cpp

Building file: ../eventcreator.cpp
```

```
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"eventcreator.d" -MT"eventcreator.d"
-o "eventcreator.o" "../eventcreator.cpp"
../eventcreator.cpp: In member function int EventCreator::ChooseEvent():
../eventcreator.cpp:160:51: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
../eventcreator.cpp: In member function char EventCreator::ChooseCourse(Event):
../eventcreator.cpp:194:52: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
Finished building: ../eventcreator.cpp

Building file: ../fileio.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"fileio.d" -MT"fileio.d" -o "fileio.o" "../fileio.cpp"
Finished building: ../fileio.cpp

Building file: ../ioscanner.cpp
Invoking: GCC C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"ioscanner.d" -MT"ioscanner.d" -o "ioscanner.o" "../ioscanner.cpp"
../ioscanner.cpp: In member function std::string IOScanner::ReadString(int):
../ioscanner.cpp:49:25: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
../ioscanner.cpp:52:29: warning: comparison between signed and unsigned integer expressions [-Wsign-compare]
Finished building: ../ioscanner.cpp

Building target: Event Creator
Invoking: GCC C++ Linker
g++ -o "Event Creator" ./course.o ./entrant.o ./event.o ./eventcreator.o ./fileio.o ./ioscanner.o
Finished building target: Event Creator


12:22:53 Build Finished (took 2s.319ms)
```

1.3 Session Output

This section shows the log of the Event Creation programs execution when creating an event with a short course and a couple of competitors.

Listing 14: Output of C++ Event Creation Program

```
----------------------
EVENT CREATION PROGRAM
----------------------

MAIN MENU
--------------------------
Enter an option:
1 - Make new event
2 - Add entrants to event
3 - Create course for event
4 - Write an event to file
5 - View an event in the system
6 - Exit Program
1
Enter name of event:
MyNewEvent
Enter event date (DD/MM/YY):
16/3/13
Enter event start time (HH:MM):
12:00
Enter location of nodes file for event:
../event_3/nodes.txt
MAIN MENU
--------------------------
Enter an option:
1 - Make new event
2 - Add entrants to event
3 - Create course for event
4 - Write an event to file
```

```
5 − View an event in the system
6 − Exit Program
3
Please choose an event:
0 − MyNewEvent
0
Enter nodes for course. Enter 0 to finish:
1
3
4
9
12
14
1
0
MAIN MENU
−−−−−−−−−−−−−−−−−−−−−−−−−
Enter an option:
1 − Make new event
2 − Add entrants to event
3 − Create course for event
4 − Write an event to file
5 − View an event in the system
6 − Exit Program
2
Please choose an event:
0 − MyNewEvent
0
Enter number of entrants to add:
3
Enter entrant's name:
Greg Jones
Please choose course for the entrant:
0 − A
0
Enter entrant's name:
Bob Jones
Please choose course for the entrant:
0 − A
0
Enter entrant's name:
Jane Doe
Please choose course for the entrant:
0 − A
0
MAIN MENU
−−−−−−−−−−−−−−−−−−−−−−−−−
Enter an option:
1 − Make new event
2 − Add entrants to event
3 − Create course for event
4 − Write an event to file
5 − View an event in the system
6 − Exit Program
4
Please choose an event:
0 − MyNewEvent
0
MAIN MENU
−−−−−−−−−−−−−−−−−−−−−−−−−
Enter an option:
1 − Make new event
2 − Add entrants to event
3 − Create course for event
4 − Write an event to file
5 − View an event in the system
6 − Exit Program
6
```

1.4 Generated Output Files

Below shows the files generated using the Event Creation program during the previous run.

Listing 15: name.txt file output from listing 14

```
MyNewEvent
16th March 2013
12:00
```

Listing 16: courses.txt file output from listing 14

```
A 7 1 3 4 9 12 14 1
```

Listing 17: entrants.txt file output from listing 14

```
1 A Greg Jones
2 A Bob Jones
3 A Jane Doe
```

2 Checkpoint Manager Program Documentation

This section provides documentation for the Checkpoint Manager application which is written in Java with a Swing GUI. Full Javadoc for this application is available via the CD.

2.1 Code Listing

Below is a full code listing for the source code to my Checkpoint Manager application.

Listing 18: CheckpointManagerGUI.java

```
1  package checkpoint.manager.gui;
2
3  import checkpoint.manager.FileIO;
4  import checkpoint.manager.datamodel.CPType;
5  import checkpoint.manager.datamodel.Checkpoint;
6  import checkpoint.manager.datamodel.CheckpointManager;
7  import checkpoint.manager.datamodel.Entrant;
8  import checkpoint.manager.exceptions.ArgumentParseException;
9  import java.awt.BorderLayout;
10 import java.awt.Dimension;
11 import java.awt.GridLayout;
12 import java.io.FileNotFoundException;
13 import java.io.IOException;
14 import java.text.ParseException;
15 import java.util.Date;
16 import java.util.HashMap;
17 import java.util.Iterator;
18 import java.util.Map.Entry;
19 import javax.swing.DefaultListModel;
20 import javax.swing.DefaultListSelectionModel;
21 import javax.swing.JButton;
22 import javax.swing.JCheckBox;
23 import javax.swing.JFrame;
```

```
24 import javax.swing.JLabel;
25 import javax.swing.JList;
26 import javax.swing.JOptionPane;
27 import javax.swing.JPanel;
28 import javax.swing.JScrollPane;
29 import javax.swing.JSpinner;
30 import javax.swing.SpinnerDateModel;
31
32 /**
33  * The Class CheckpointManagerGUI.
34  */
35 @SuppressWarnings("serial")
36 public class CheckpointManagerGUI extends JFrame {
37
38     /** The checkpoint list model to store checkpoints in the GUI. */
39     private final DefaultListModel cpListModel;
40
41     /** The checkpoint list to display checkpoints in order. */
42     private JList JLCheckpointList;
43
44     /** The entrant list to display entrants in order. */
45     private JList JLEntrantList;
46
47     /** The entrant list model to store the entrant list in the GUI. */
48     private DefaultListModel entrantListModel;
49
50     /** The checkbox for excluding an entrant. */
51     private final JCheckBox chkMCExcluded;
52
53     /** The button to check in and entrant. */
54     private final JButton btnCheckIn;
55
56     /** The arrival time of the entrant. */
57     private final JSpinner JarrivalTime;
58
59     /** The departure time of the entrant. */
60     private final JSpinner JdepartureTime;
61
62     /** The checkpoint manager GUI event listener. */
63     private final CheckpointManagerListener chkptListener;
64
65     /** The checkpoint manager to process the data model. */
66     private CheckpointManager cpManager;
67
68     /** The current entrant label. */
69     private final JLabel currentEntrant;
70
71     /** The current checkpoint label. */
72     private final JLabel currentCheckpoint;
73
74     /**
75      * Instantiates a new checkpoint manager GUI.
76      *
77      * @param args the args from the command line
78      * @throws FileNotFoundException exception thrown when file cannot be found.
79      * @throws IOException Signals that an unexpected I/O exception has occurred.
80      */
81     public CheckpointManagerGUI(HashMap<String, String> args) throws FileNotFoundException, IOException {
82         this.setSize(500, 600);
83
84         currentEntrant = new JLabel("Current Entrant: ");
85         currentCheckpoint = new JLabel("Current Checkpoint: ");
86
87         try {
88             cpManager = new CheckpointManager(args);
89             if(!cpManager.updateTimes()) {
90                 JOptionPane.showMessageDialog(null, "Could not read the times file!", "Error!", JOptionPane.ERROR_MESSAGE);
91                 System.exit(0);
92             } else {
93                 cpManager.updateLog("Read the times file.");
94             }
```

```
95          } catch (ParseException ex) {
96              JOptionPane.showMessageDialog(null, ex, "Could not Parse Text times file!", JOptionPane.ERROR_MESSAGE);
97              System.exit(0);
98          }
99
100         chkptListener = new CheckpointManagerListener(this);
101         cpListModel = new DefaultListModel();
102         entrantListModel = new DefaultListModel();
103         btnCheckIn = new JButton("Check In");
104         chkMCExcluded = new JCheckBox("Exclude entrant for medical reasons");
105         JarrivalTime = new JSpinner(new SpinnerDateModel());
106         JdepartureTime = new JSpinner(new SpinnerDateModel());
107
108         initGUI();
109
110         JLCheckpointList.setSelectedIndex(0);
111         JLEntrantList.setSelectedIndex(0);
112
113         setDefaultCloseOperation(EXIT_ON_CLOSE);
114         setLayout(new GridLayout(1, 3));
115         setVisible(true);
116         pack();
117     }
118
119     /**
120      * Initialises the GUI.
121      */
122     private void initGUI() {
123         JPanel temp = new JPanel();
124         JPanel rightPanel = new JPanel();
125         JPanel centrePanel = new JPanel();
126         JPanel leftPanel = new JPanel();
127
128         //create list of checkpoints
129         JLCheckpointList = new JList(cpListModel);
130         JLCheckpointList.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
131         JLCheckpointList.setLayoutOrientation(JList.VERTICAL);
132
133         //populate list of checkpoints
134         for (Entry<Integer, Checkpoint> entry : cpManager.getCheckpoints().entrySet()) {
135             Checkpoint chk = (Checkpoint) entry.getValue();
136             cpListModel.addElement(chk.getId() + " " + chk.getType().toString());
137         }
138
139         JLCheckpointList.addListSelectionListener(chkptListener);
140         JScrollPane listScroller = new JScrollPane(JLCheckpointList);
141         listScroller.setPreferredSize(new Dimension(250, 300));
142
143         //layout list of checkpoints
144         temp.add(new JLabel("Checkpoints: "));
145         leftPanel.setLayout(new BorderLayout());
146         leftPanel.add(temp, BorderLayout.NORTH);
147         temp = new JPanel();
148         temp.add(listScroller);
149         leftPanel.add(temp, BorderLayout.SOUTH);
150
151         //create list of entrants
152         JLEntrantList = new JList(entrantListModel);
153         JLEntrantList.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
154         JLEntrantList.setLayoutOrientation(JList.VERTICAL);
155         refreshEntrants();
156
157         JLEntrantList.addListSelectionListener(chkptListener);
158
159         listScroller = new JScrollPane(JLEntrantList);
160         listScroller.setPreferredSize(new Dimension(250, 300));
161
162         //layout list of entrants
163         rightPanel.setLayout(new BorderLayout());
164         temp = new JPanel();
165         temp.add(new JLabel("Entrants: "));
```

```
166             rightPanel.add(temp);
167             rightPanel.add(temp, BorderLayout.NORTH);
168             temp = new JPanel();
169             temp.add(listScroller);
170             rightPanel.add(temp, BorderLayout.SOUTH);
171
172             //create centre panel
173             JarrivalTime.setModel(new SpinnerDateModel());
174             JarrivalTime.setEditor(new JSpinner.DateEditor(JarrivalTime, "HH:mm"));
175             JdepartureTime.setModel(new SpinnerDateModel());
176             JdepartureTime.setEditor(new JSpinner.DateEditor(JdepartureTime, "HH:mm"));
177
178             btnCheckIn.setActionCommand("CheckIn");
179             btnCheckIn.addActionListener(chkptListener);
180
181             //layout elements in centre panel
182             centrePanel.setLayout(new BorderLayout());
183
184             temp = new JPanel();
185
186             JPanel first = new JPanel();
187             first.add(currentEntrant);
188             temp.add(first);
189             first = new JPanel();
190             first.add(currentCheckpoint);
191             temp.add(first);
192
193             JPanel second = new JPanel();
194             second.add(new JLabel("Arrival Time: "));
195             second.add(JarrivalTime);
196             temp.add(second);
197
198             JPanel third = new JPanel();
199             third.add(new JLabel("Dpearture Time: "));
200             third.add(JdepartureTime);
201             temp.add(third);
202
203             JPanel fourth = new JPanel();
204             fourth.add(chkMCExcluded);
205             temp.add(fourth);
206
207             JPanel fifth = new JPanel();
208             fifth.add(btnCheckIn);
209             temp.add(fifth);
210             centrePanel.add(temp, BorderLayout.CENTER);
211             centrePanel.setPreferredSize(new Dimension(300, 100));
212
213             getContentPane().add(leftPanel);
214             getContentPane().add(centrePanel);
215             getContentPane().add(rightPanel);
216         }
217
218         /**
219          * Parses the ID from the start of a list box item.
220          *
221          * @param list the list model
222          * @param index the index of the selected item
223          * @return the ID
224          */
225         private int parseIndex(DefaultListModel list, int index) {
226             return (Integer.parseInt(list.get(index).toString().split("[a-z ]")[0]));
227         }
228
229         /**
230          * Check in an entrant in response to a users click.
231          */
232         public void doCheckIn() {
233             int index = JLEntrantList.getSelectedIndex();
234             int entrantId = parseIndex(entrantListModel, index);
235             index = JLCheckpointList.getSelectedIndex();
236             int checkpointId = parseIndex(cpListModel, index);
```

```
237         Checkpoint checkpoint = cpManager.getCheckpoint(checkpointId);
238
239         Date arrivalTime = (Date) JarrivalTime.getValue();
240         Date departureTime = null;
241         boolean mcExcluded = chkMCExcluded.isSelected();
242         boolean successful = false;
243         boolean validInput = true;
244
245         //reload the times file.
246         try {
247             successful = cpManager.updateTimes();
248             if(!successful) {
249                 JOptionPane.showMessageDialog(this, "Could not reload times! Perhaps file was locked by another process?");
250             } else {
251                 cpManager.updateLog("Read the times file.");
252             }
253         } catch (FileNotFoundException ex) {
254             JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
255         } catch (IOException ex) {
256             JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
257         } catch (ParseException ex) {
258             JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
259         }
260
261         if(successful) {
262             //check if we're at a medical checkpoint
263             if(JdepartureTime.isEnabled()) {
264                 departureTime = (Date) JdepartureTime.getValue();
265             }
266
267             //check if the times entered were valid
268             if((checkpoint.getType()==CPType.MC && cpManager.compareTime(arrivalTime, departureTime))
269                     || !cpManager.checkValidTime(entrantId, arrivalTime)) {
270                 JOptionPane.showMessageDialog(this, "Invalid time data!");
271                 validInput = false;
272             }
273
274             if(validInput) {
275                 //check if the entrant will be excluded with this update
276                 if(cpManager.willExcludedEntrant(entrantId, checkpointId) || mcExcluded) {
277                     //confirm this with the user.
278                     int confirm = JOptionPane.showConfirmDialog(this,
279                             "This will exclude the entrant. Are you sure?",
280                             "Confirm Choice", JOptionPane.YES_NO_OPTION);
281                     validInput = (confirm == JOptionPane.YES_OPTION) ? true : false;
282                 }
283             }
284         }
285
286         if(validInput) {
287
288             //perform the update
289             try {
290                 successful = cpManager.checkInEntrant(entrantId, checkpointId, arrivalTime, departureTime, mcExcluded);
291                 refreshEntrants();
292             } catch (FileNotFoundException ex) {
293                 JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
294             } catch (IOException ex) {
295                 JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
296             } catch (ParseException ex) {
297                 JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
298             }
299
300             //feedback to the user if successful
301             if(successful) {
302                 JOptionPane.showMessageDialog(this, "Checked in!");
303             } else {
304                 JOptionPane.showMessageDialog(this, "Could not check in entrant! Perhaps file was locked by another process?");
305             }
306
307             try {
```

```
308                 successful = cpManager.updateLog("Checked in entrant " + entrantId + " @ node " + checkpointId);
309             } catch (FileNotFoundException ex) {
310                 JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
311             } catch (IOException ex) {
312                 JOptionPane.showMessageDialog(this, ex, "Error:", JOptionPane.ERROR_MESSAGE);
313             }
314 
315             if(!successful) {
316                 JOptionPane.showMessageDialog(this, "Could not write to log file!");
317             }
318         }
319     }
320 
321     /**
322      * Update the GUI "currently selected" labels in response to user interaction.
323      */
324     public void updateOutput() {
325         int index = JLCheckpointList.getSelectedIndex();
326 
327         if(index >= 0) {
328             String currentChkpt = cpListModel.get(index).toString();
329             currentCheckpoint.setText("Current Checkpoint: " + currentChkpt);
330         }
331 
332         index = JLEntrantList.getSelectedIndex();
333         if(index >= 0) {
334             String entrant = entrantListModel.get(index).toString();
335             currentEntrant.setText("Current Entrant: " + entrant);
336         }
337     }
338 
339     /**
340      * Toggle input for a medical checkpoint
341      */
342     public void toggleMedicalCPInput() {
343         int index = JLCheckpointList.getSelectedIndex();
344         int cpId = (Integer.parseInt(cpListModel.get(index).toString().split("[a-z ]")[0]));
345         if(cpManager.getCheckpoint(cpId).getType() == CPType.MC) {
346             JdepartureTime.setEnabled(true);
347             chkMCExcluded.setEnabled(true);
348         } else {
349             JdepartureTime.setEnabled(false);
350             chkMCExcluded.setEnabled(false);
351         }
352     }
353 
354     /**
355      * The main method and entry point to the application.
356      *
357      * @param args the command line arguments
358      */
359     public static void main(String[] args) {
360         try {
361             HashMap<String, String> cmdArgs;
362             cmdArgs = FileIO.parseArgs(args);
363             CheckpointManagerGUI gui = new CheckpointManagerGUI(cmdArgs);
364         } catch (ArgumentParseException ex) {
365             printHelp();
366             System.exit(0);
367         } catch (FileNotFoundException ex) {
368             JOptionPane.showMessageDialog(null, ex, "Error:", JOptionPane.ERROR_MESSAGE);
369             System.exit(0);
370         } catch (IOException ex) {
371             JOptionPane.showMessageDialog(null, ex, "Error:", JOptionPane.ERROR_MESSAGE);
372             System.exit(0);
373         }
374     }
375 
376     /**
377      * Prints the help menu to the console.
378      */
```

```
379     private static void printHelp() {
380             System.out.println("Checkpoint Manager -- Usage:");
381             System.out.println("Please supply the following files using the given flags");
382             System.out.println(" -E <entrants file>");
383             System.out.println(" -C <courses file>");
384             System.out.println(" -K <checkpoints file>");
385             System.out.println(" -T <times file>");
386             System.out.println(" -L <log file>");
387     }
388 
389     /**
390      * Refresh the list of entrants.
391      */
392     private void refreshEntrants() {
393         entrantListModel = new DefaultListModel();
394         Iterator<Entry<Integer,Entrant>> it = cpManager.getEntrants().entrySet().iterator();
395         while (it.hasNext()) {
396             Entrant e = (Entrant) ((Entry<Integer, Entrant>) it.next()).getValue();
397             if(!(e.isExcluded() || e.isFinished())) {
398                 entrantListModel.addElement(e.getId() + " " + e.getName());
399             }
400         }
401 
402         JLEntrantList.setModel(entrantListModel);
403         JLEntrantList.setSelectedIndex(0);
404     }
405 }
```

Listing 19: CheckpointManagerListener.java

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package checkpoint.manager.gui;
6 
7  import java.awt.event.ActionEvent;
8  import java.awt.event.ActionListener;
9  import javax.swing.event.ListSelectionEvent;
10 import javax.swing.event.ListSelectionListener;
11 
12 // TODO: Auto-generated Javadoc
13 /**
14  * The listener interface for receiving checkpointManager events.
15  * The class that is interested in processing a checkpointManager
16  * event implements this interface, and the object created
17  * with that class is registered with a component using the
18  * component's <code>addCheckpointManagerListener<code> method. When
19  * the checkpointManager event occurs, that object's appropriate
20  * method is invoked.
21  *
22  * @author samuel
23  */
24 public class CheckpointManagerListener implements ActionListener, ListSelectionListener {
25 
26     /** The parent. */
27     private final CheckpointManagerGUI parent;
28 
29     /**
30      * Instantiates a new checkpoint manager listener.
31      *
32      * @param parent the parent
33      */
34     CheckpointManagerListener(CheckpointManagerGUI parent) {
35         this.parent = parent;
36     }
37 
38     /* (non-Javadoc)
39      * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
40      */
41     @Override
```

```
42      public void actionPerformed(ActionEvent ae) {
43          if(ae.getActionCommand().equals("CheckIn")) {
44              parent.doCheckIn();
45          }
46      }
47
48      /* (non-Javadoc)
49       * @see javax.swing.event.ListSelectionListener#valueChanged(javax.swing.event.ListSelectionEvent)
50       */
51      @Override
52      public void valueChanged(ListSelectionEvent lse) {
53          parent.updateOutput();
54          parent.toggleMedicalCPInput();
55      }
56
57  }
```

Listing 20: CheckpointManager.java

```
1   package checkpoint.manager.datamodel;
2
3   import checkpoint.manager.FileIO;
4   import java.io.FileNotFoundException;
5   import java.io.IOException;
6   import java.text.ParseException;
7   import java.text.SimpleDateFormat;
8   import java.util.Date;
9   import java.util.HashMap;
10  import java.util.LinkedHashMap;
11  import java.util.PriorityQueue;
12
13  /**
14   * The Class CheckpointManager.
15   * Main management class to the underlying data model.
16   * Manages the processing and updating of data from user input via the GUI
17   * into the data files.
18   * @author Samuel Jackson (slj11@aber.ac.uk)
19   */
20  public class CheckpointManager {
21
22      /** The FileIO object to write to files. */
23      private final FileIO fio;
24
25      /** The LinkedHashMap of entrants. Entrant ID used as key. */
26      private LinkedHashMap<Integer, Entrant> entrants;
27
28      /** The LinkedHashMap of checkpoints. Checkpoint ID used as key */
29      private LinkedHashMap<Integer, Checkpoint> checkpoints;
30
31      /** The HashMap of courses. Course ID used as key */
32      private HashMap<Character, Course> courses;
33
34      /** The PriorityQueue of times. Oldest time has highest priority */
35      private PriorityQueue<CPTimeData> times;
36
37      /**
38       * Instantiates a new checkpoint manager.
39       *
40       * @param args the arguments supplied via the command line.
41       * @throws FileNotFoundException exception thrown when file cannot be found.
42       * @throws IOException Signals that an unexpected I/O exception has occurred.
43       * @throws ParseException the parse exception thrown by failing to parse a date.
44       */
45      public CheckpointManager(HashMap<String, String> args)
46              throws FileNotFoundException, IOException, ParseException {
47
48          fio = new FileIO(args);
49          entrants = fio.readEntrants();
50          checkpoints = fio.readCheckpoints();
51          courses = fio.readCourses(checkpoints);
52      }
```

```
53
54      /**
55       * Check if updating an entrant to the given checkpoint ID will cause the
56       * entrant to be excluded.
57       *
58       * @param entrantId the entrant's id
59       * @param chkptId the checkpoint id
60       * @return true, if successful
61       */
62      public boolean willExcludedEntrant(int entrantId, int chkptId) {
63
64          Entrant entrant = getEntrant(entrantId);
65          Course course = courses.get(entrant.getCourse());
66
67          if(!entrant.isFinished()) {
68              if(course.getNode(entrant.getPosition()+1) != chkptId
69                  && (!entrant.hasStarted() || entrant.getLatestTime().getUpdateType() != 'A')) {
70                  return true;
71              }
72          }
73
74          return false;
75      }
76
77      /**
78       * Re-read the times file and update all entrants with a new set of times.
79       *
80       * @return true, if successful in reading the file
81       * @throws FileNotFoundException exception thrown when file cannot be found.
82       * @throws ParseException the parse exception if a date could not be parsed.
83       * @throws IOException Signals that an unexpected I/O exception has occurred.
84       */
85      public boolean updateTimes()
86              throws FileNotFoundException, ParseException, IOException {
87          times = fio.readCheckpointData(entrants, courses);
88
89          //Failed to acquire lock or not
90          return (times != null);
91      }
92
93      /**
94       * Check compare the time part of two instances of a date object
95       *
96       * @param time the first time to be compared
97       * @param time2 the second time to be compared
98       * @return true, if the time is valid
99       */
100     public boolean compareTime(Date time, Date time2) {
101         SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
102         return sdf.format(time).compareTo(sdf.format(time2)) >= 0;
103     }
104
105     /**
106      * Check if the supplied time is a valid time.
107      *
108      * @param entrantId the entrant ID
109      * @param time the time to be checked.
110      * @return true, if the time is valid
111      */
112     public boolean checkValidTime(int entrantId, Date time) {
113         Entrant entrant = getEntrant(entrantId);
114         if(entrant.hasStarted()) {
115             if(compareTime(entrant.getLatestTime().getTime(), time)) {
116                 return false;
117             }
118         }
119
120         return true;
121     }
122
123     /**
```

```
124         * Check in entrant.
125         *
126         * @param entrantId the entrant ID
127         * @param chkptId the checkpoint ID
128         * @param arrivalTime the arrival time of the entrant
129         * @param departureTime the departure time of the entrant
130         * @param mcExcluded the flag for if the entrant is exlcuded for medical reasons
131         * @return true, if successful at writing data to file.
132         * @throws FileNotFoundException exception thrown when file cannot be found.
133         * @throws IOException Signals that an unexpected I/O exception has occurred.
134         * @throws ParseException the parse exception if a date could not be parsed.
135         */
136        public boolean checkInEntrant(int entrantId, int chkptId,
137                Date arrivalTime, Date departureTime, boolean mcExcluded)
138                throws FileNotFoundException, IOException, ParseException {
139
140            boolean checkedIn = false;
141            Date checkInTime;
142            Entrant entrant = entrants.get(entrantId);
143            Checkpoint chkpoint = checkpoints.get(chkptId);
144            Course course = courses.get(entrant.getCourse());
145            char updateType = 'T';
146
147            if(!entrant.isExcluded()) {
148                checkInTime = arrivalTime;
149
150                //set arrival time if medical checkpoint
151                if (chkpoint.getType() == CPType.MC) {
152                    checkInTime = departureTime;
153                    addEntrantTime(entrantId, chkptId, arrivalTime, 'A', CPType.MC);
154                    updateType = 'D';
155                }
156
157                CPType type = (updateType == 'D') ? CPType.MC : CPType.CP;
158
159                //exclude entrant if they failed for medical reasons
160                if (mcExcluded) {
161                    entrant.setExcluded(true);
162                    updateType = 'E';
163                }
164
165                //exclude entrant if they came to wrong checkpoint
166                if(willExcludedEntrant(entrant.getId(), chkpoint.getId())) {
167                    entrant.setExcluded(true);
168                    updateType = 'I';
169                }
170
171                //check if the entrant is after this update
172                if(entrant.getPosition() >= course.getLength()-2) {
173                    entrant.setFinished(true);
174                }
175
176                addEntrantTime(entrantId, chkptId, checkInTime, updateType, type);
177                entrant.incrementPosition();
178                checkedIn = fio.writeTimes(times);
179            }
180
181            return checkedIn;
182        }
183
184        /**
185         * Output an update to the log file.
186         * @param output the output to add to the log file.
187         * @return true, if updating the log file was successful
188         * @throws IOException Signals that an unexpected I/O exception has occurred.
189         * @throws FileNotFoundException exception thrown when file cannot be found.
190         */
191        public boolean updateLog(String output) throws FileNotFoundException, IOException {
192            return fio.writeLog(output);
193        }
194
```

```
195     /**
196      * Creates a time update and adds it to the list of times and the entrant's
197      * time list.
198      *
199      * @param entrantId the entrant ID
200      * @param chkptId the checkpoint ID
201      * @param date the time of the update
202      * @param updateType the type of update (T, I, A, D, E)
203      * @param type the type of checkpoint.
204      */
205     private void addEntrantTime(int entrantId, int chkptId, Date date, char updateType, CPType type) {
206         CPTimeData time = new CPTimeData();
207         time.setTime(date);
208         time.setEntrantId(entrantId);
209         time.setType(type);
210         time.setUpdateType(updateType);
211         time.setNode(chkptId);
212         entrants.get(entrantId).addTime(time);
213         times.add(time);
214     }
215 
216     /**
217      * Gets an entrant with the given ID.
218      *
219      * @param id the ID of the entrant
220      * @return the entrant with the given ID
221      */
222     public Entrant getEntrant(int id) {
223         return getEntrants().get(id);
224     }
225 
226     /**
227      * Gets a checkpoint with the given ID
228      *
229      * @param id the ID of the checkpoint
230      * @return the checkpoint with the given ID
231      */
232     public Checkpoint getCheckpoint(int id) {
233         return getCheckpoints().get(id);
234     }
235 
236     /**
237      * Gets the list of entrants.
238      *
239      * @return the entrant list
240      */
241     public HashMap<Integer, Entrant> getEntrants() {
242         return entrants;
243     }
244 
245     /**
246      * Gets the list of checkpoints.
247      *
248      * @return the checkpoint list
249      */
250     public LinkedHashMap<Integer, Checkpoint> getCheckpoints() {
251         return checkpoints;
252     }
253 }
```

Listing 21: Entrant.java

```
1 
2  package checkpoint.manager.datamodel;
3 
4  import java.util.ArrayList;
5 
6  /**
7   * The Class Entrant.
8   * Holds data about a single entrant in the event.
9   * @author Samuel Jackson (slj11@aber.ac.uk)
```

```
10   */
11  public class Entrant {
12
13      /** The name of the entrant. */
14      private String name;
15
16      /** The course the entrant is on. */
17      private char course;
18
19      /** The id of the entrant. */
20      private int id;
21
22      /** The list of time updates an entrant has been checked in on. */
23      private ArrayList<CPTimeData> times;
24
25      /** Whether the entrant has been exlcuded or not. */
26      private boolean excluded;
27
28      /** Whether the entrant has finished or not. */
29      private boolean finished;
30
31      /** The position of the entrant on the course. */
32      private int position;
33
34      /**
35       * Instantiates a new entrant.
36       */
37      public Entrant() {
38          times = new ArrayList<CPTimeData>();
39          excluded = false;
40          finished = false;
41          position = -1;
42      }
43
44      /**
45       * Gets the name of this entrant.
46       *
47       * @return the name
48       */
49      public String getName() {
50          return name;
51      }
52
53      /**
54       * Sets the name of this entrant.
55       *
56       * @param name the name to set
57       */
58      public void setName(String name) {
59          this.name = name;
60      }
61
62      /**
63       * Gets the course the entrant is on.
64       *
65       * @return the course
66       */
67      public char getCourse() {
68          return course;
69      }
70
71      /**
72       * Sets the course the entrant is on.
73       *
74       * @param course the course to set
75       */
76      public void setCourse(char course) {
77          this.course = course;
78      }
79
80      /**
```

```
81          * Gets the id of the entrant.
82          *
83          * @return the id
84          */
85         public int getId() {
86             return id;
87         }
88
89         /**
90          * Sets the id of the entrant.
91          *
92          * @param id the id to set
93          */
94         public void setId(int id) {
95             this.id = id;
96         }
97
98         /**
99          * Gets the times the entrant has been check in at.
100         *
101         * @return the times
102         */
103        public ArrayList<CPTimeData> getTimes() {
104            return times;
105        }
106
107        /**
108         * Sets the times the entrant has been check in at.
109         *
110         * @param times the times to set
111         */
112        public void setTimes(ArrayList<CPTimeData> times) {
113            this.times = times;
114        }
115
116        /**
117         * Adds a time update to the entrant
118         *
119         * @param cpData the cp data
120         */
121        public void addTime(CPTimeData cpData) {
122            this.times.add(cpData);
123        }
124
125        /**
126         * Checks if is excluded.
127         *
128         * @return the excluded
129         */
130        public boolean isExcluded() {
131            return excluded;
132        }
133
134        /**
135         * Sets the as excluded or not.
136         *
137         * @param excluded the excluded to set
138         */
139        public void setExcluded(boolean excluded) {
140            this.excluded = excluded;
141        }
142
143        /**
144         * Gets the position of the entrant.
145         *
146         * @return the position
147         */
148        public int getPosition() {
149            return position;
150        }
151
```

```
152     /**
153      * Reset position of the entrant.
154      */
155     public void resetPosition() {
156         position = -1;
157     }
158 
159     /**
160      * Increment position of the entrant.
161      */
162     public void incrementPosition() {
163         position++;
164     }
165 
166     /**
167      * Check if the entrant has started.
168      *
169      * @return true, if entrant has started
170      */
171     public boolean hasStarted() {
172         return (times.size() > 0);
173     }
174 
175     /**
176      * Gets the latest time currently avalible for the entrant.
177      *
178      * @return the latest time
179      */
180     public CPTimeData getLatestTime() {
181         return times.get(times.size()-1);
182     }
183 
184     /**
185      * Checks if is finished has finished.
186      *
187      * @return the finished
188      */
189     public boolean isFinished() {
190         return finished;
191     }
192 
193     /**
194      * Sets the finished as been finished or not.
195      *
196      * @param finished the finished to set
197      */
198     public void setFinished(boolean finished) {
199         this.finished = finished;
200     }
201 }
```

Listing 22: Course.java

```
1  package checkpoint.manager.datamodel;
2 
3  import java.util.ArrayList;
4 
5  /**
6   * The Class Course.
7   * Holds data about a single course
8   *
9   * @author Samuel Jackson (slj11@aber.ac.uk)
10  */
11 public class Course {
12 
13     /** The id of the course */
14     private char id;
15 
16     /** The nodes in the course */
17     private ArrayList<Integer> nodes;
18 
```

```
19      /**
20       * Gets the id of the course.
21       *
22       * @return the id
23       */
24      public char getId() {
25          return id;
26      }
27
28      /**
29       * Sets the id of the course.
30       *
31       * @param id the id to set
32       */
33      public void setId(char id) {
34          this.id = id;
35      }
36
37      /**
38       * Gets the length.
39       *
40       * @return the length
41       */
42      public int getLength() {
43          return nodes.size();
44      }
45
46      /**
47       * Gets the nodes in the course.
48       *
49       * @return the nodes
50       */
51      public ArrayList<Integer> getNodes() {
52          return nodes;
53      }
54
55      /**
56       * Sets the nodes in the course.
57       *
58       * @param nodes the nodes to set
59       */
60      public void setNodes(ArrayList<Integer> nodes) {
61          this.nodes = nodes;
62      }
63
64      /**
65       * Gets the node.
66       *
67       * @param index the index of the node.
68       * @return the node
69       */
70      public int getNode(int index) {
71          return getNodes().get(index);
72      }
73  }
```

Listing 23: Checkpoint.java

```
1   package checkpoint.manager.datamodel;
2
3   /**
4    * The Class Checkpoint.
5    * Holds data about a single checkpoint (or medical checkpoint) in an event.
6    * @author Samuel Jackson (slj11@aber.ac.uk)
7    */
8   public class Checkpoint {
9
10      /** The id of the checkpoint */
11      private int id;
12
13      /** The type of the checkpoint. */
```

```
14      private CPType type;
15
16      /**
17       * Instantiates a new checkpoint.
18       *
19       * @param id the id of the checkpoint
20       * @param type the type of the checkpoint
21       */
22      public Checkpoint(int id, String type) {
23          this.id = id;
24          this.type = CPType.valueOf(type);
25      }
26
27      /**
28       * Gets the id of the checkpoint.
29       *
30       * @return the id
31       */
32      public int getId() {
33          return id;
34      }
35
36      /**
37       * Gets the type type of the checkpoint.
38       *
39       * @return the type
40       */
41      public CPType getType() {
42          return type;
43      }
44  }
```

Listing 24: CPTimeData.java

```
1   package checkpoint.manager.datamodel;
2
3   import java.text.SimpleDateFormat;
4   import java.util.Calendar;
5   import java.util.Date;
6
7   /**
8    * The Class CPTimeData.
9    * Holds data about a single checkpoint time update.
10   *
11   * @author Samuel Jackson (slj11@aber.ac.uk)
12   */
13  public class CPTimeData implements Comparable<CPTimeData> {
14
15      /** The entrant id of the entrant. */
16      private int entrantId;
17
18      /** The type of checkpoint. */
19      private CPType type;
20
21      /** The update type. One of the 5 types of updates allowed (T, I, A, D, E) . */
22      private char updateType;
23
24      /** The node that the checkpoint update occurred on. */
25      private int node;
26
27      /** The time the update occurred. */
28      private Date time;
29
30      /** The date formatter object. */
31      private final SimpleDateFormat sdf;
32
33      /**
34       * Instantiates a new instance of a checkpoint time data object.
35       */
36      public CPTimeData() {
37          sdf = new SimpleDateFormat("HH:mm");
```

```
38      }
39
40      /**
41       * Gets the entrant's id.
42       *
43       * @return the entrantId
44       */
45      public int getEntrantId() {
46          return entrantId;
47      }
48
49      /**
50       * Sets the entrant id.
51       *
52       * @param entrantId the entrantId to set
53       */
54      public void setEntrantId(int entrantId) {
55          this.entrantId = entrantId;
56      }
57
58      /**
59       * Gets the type.
60       *
61       * @return the type
62       */
63      public CPType getType() {
64          return type;
65      }
66
67      /**
68       * Sets the type of update.
69       *
70       * @param type the type to set
71       */
72      public void setType(CPType type) {
73          this.type = type;
74      }
75
76      /**
77       * Gets the node that the update occurred on.
78       *
79       * @return the cpId
80       */
81      public int getNode() {
82          return node;
83      }
84
85      /**
86       * Sets the node that the update occurred on.
87       *
88       * @param checkpointId the cpId to set
89       */
90      public void setNode(int checkpointId) {
91          this.node = checkpointId;
92      }
93
94      /**
95       * Gets the time as a string.
96       *
97       * @return the time
98       */
99      public String getStringTime() {
100         return sdf.format(time);
101     }
102
103     /**
104      * Gets the time (Date) object.
105      *
106      * @return the time
107      */
108     public Date getTime() {
```

```
109             return time;
110         }
111 
112         /**
113          * Sets the time.
114          *
115          * @param time the new time
116          */
117         public void setTime(Date time) {
118 
119             this.time = time;
120         }
121 
122         /**
123          * Gets the update type. One of the 5 types of updates (T,I,A,D,E)
124          *
125          * @return the updateType
126          */
127         public char getUpdateType() {
128             return updateType;
129         }
130 
131         /**
132          * Sets the update type. One of the 5 types of updates (T,I,A,D,E)
133          *
134          * @param updateType the updateType to set
135          */
136         public void setUpdateType(char updateType) {
137             this.updateType = updateType;
138         }
139 
140         /* (non-Javadoc)
141          * @see java.lang.Comparable#compareTo(java.lang.Object)
142          */
143         @Override
144         public int compareTo(CPTimeData t) {
145             return sdf.format(time).compareTo(sdf.format(t.getTime()));
146         }
147 }
```

Listing 25: CPType.java

```
1  package checkpoint.manager.datamodel;
2 
3  // TODO: Auto-generated Javadoc
4  /**
5   * The Enum CPType.
6   * The used to represent the type of a checkpoint, either regular or medical.
7   * @author Samuel Jackson (slj11@aber.ac.uk)
8   */
9  public enum CPType {
10     CP,
11     MC
12 }
```

Listing 26: FileIO.java

```
1 
2  package checkpoint.manager;
3 
4  import checkpoint.manager.datamodel.CPTimeData;
5  import checkpoint.manager.datamodel.Checkpoint;
6  import checkpoint.manager.datamodel.Course;
7  import checkpoint.manager.datamodel.Entrant;
8  import checkpoint.manager.exceptions.ArgumentParseException;
9  import java.io.File;
10 import java.io.FileNotFoundException;
11 import java.io.FileOutputStream;
12 import java.io.IOException;
13 import java.io.PrintWriter;
```

```
14 import java.io.RandomAccessFile;
15 import java.nio.channels.FileLock;
16 import java.text.ParseException;
17 import java.text.SimpleDateFormat;
18 import java.util.ArrayList;
19 import java.util.Date;
20 import java.util.HashMap;
21 import java.util.LinkedHashMap;
22 import java.util.Map.Entry;
23 import java.util.PriorityQueue;
24 import java.util.Scanner;
25
26 /**
27  * The Class FileIO.
28  * Reads and writes files used during a race event.
29  *
30  * @author Samuel Jackson (slj11@aber.ac.uk)
31  */
32 public class FileIO {
33
34     /** The simple date formatter */
35     private SimpleDateFormat sdf;
36
37     /** The names of each of the files passed as command line arguements. */
38     private HashMap<String, String> filenames;
39
40     /**
41      * Instantiates a new instace of FileIO.
42      *
43      * @param args HashMap of filenames
44      */
45     public FileIO (HashMap<String, String> args) {
46         filenames = args;
47         sdf = new SimpleDateFormat("HH:mm");
48     }
49
50     /**
51      * Parses the command line arguments.
52      *
53      * @param args the command line arguments
54      * @return HashMap of parse arguments
55      * @throws ArgumentParseException the argument parse exception thrown if
56      * arguments array cannot be parsed.
57      */
58     public static HashMap<String, String> parseArgs(String[] args)
59             throws ArgumentParseException {
60         HashMap<String, String> argsList = new HashMap<String, String>();
61
62         if (args.length == 10) { //all arguments are required
63             for (int i = 0; i < args.length; i+=2) {
64                 String key = "";
65                 switch(args[i].charAt(1)) {
66                     case 'E':
67                         key = "entrants";
68                         break;
69                     case 'T':
70                         key = "times";
71                         break;
72                     case 'C':
73                         key = "courses";
74                         break;
75                     case 'K':
76                         key = "checkpoints";
77                         break;
78                     case 'L':
79                         key = "log";
80                         break;
81                     default:
82                         throw new ArgumentParseException();
83                 }
84
```

```
85                  argsList.put(key, args[i+1]);
86              }
87          } else {
88              throw new ArgumentParseException();
89          }
90
91          return argsList;
92      }
93
94      /**
95       * Read in the entrant's file.
96       *
97       * @return the linked HashMap of entrant's, identified by an entrant's ID.
98       * @throws FileNotFoundException exception thrown when file cannot be found.
99       * @throws IOException Signals that an unexpected I/O exception has occurred.
100      */
101     public LinkedHashMap<Integer, Entrant> readEntrants()
102             throws FileNotFoundException, IOException {
103         Scanner in = new Scanner(new File(filenames.get("entrants")));
104         LinkedHashMap<Integer, Entrant> entrants = new LinkedHashMap<Integer, Entrant>();
105
106         while(in.hasNext()) {
107             Entrant e = new Entrant();
108             e.setId(in.nextInt());
109             e.setCourse(in.next().charAt(0));
110             e.setName(in.nextLine());
111             entrants.put(e.getId(),e);
112         }
113
114         in.close();
115
116         return entrants;
117     }
118
119     /**
120      * Read in the courses file.
121      *
122      * @param checkpoints the HashMap of nodes that are checkpoints (or medical checkpoints).
123      * @return HashMap of courses, identified by the course ID.
124      * @throws FileNotFoundException exception thrown when file cannot be found.
125      * @throws IOException Signals that an unexpected I/O exception has occurred.
126      */
127     public HashMap<Character, Course> readCourses(LinkedHashMap<Integer, Checkpoint> checkpoints)
128             throws FileNotFoundException, IOException {
129         Scanner in = new Scanner(new File(filenames.get("courses")));
130
131         HashMap<Character, Course> courses = new HashMap<Character, Course>();
132
133         while (in.hasNext()) {
134             ArrayList<Integer> nodes = new ArrayList<Integer>();
135             Course course = new Course();
136             course.setId(in.next().charAt(0));
137
138             while(in.hasNextInt()) {
139                 int node = in.nextInt();
140                 if(checkpoints.containsKey(node)) {
141                     nodes.add(node);
142                 }
143             }
144             course.setNodes(nodes);
145             courses.put(course.getId(), course);
146         }
147
148         in.close();
149
150         return courses;
151     }
152
153     /**
154      * Read checkpoint data.
155      *
```

```
156         * @param entrants the list of entrants to update.
157         * @param courses the list of all courses.
158         * @return PriorityQueue of CPTimeData objects, ordered by oldest time first.
159         * @throws FileNotFoundException exception thrown when file cannot be found.
160         * @throws ParseException the parse exception thrown when a date cannot be parsed.
161         * @throws IOException Signals that an unexpected I/O exception has occurred.
162         */
163        public PriorityQueue<CPTimeData> readCheckpointData(
164                LinkedHashMap<Integer, Entrant> entrants, HashMap<Character, Course> courses)
165                throws FileNotFoundException, ParseException, IOException {
166            RandomAccessFile fis = new RandomAccessFile(filenames.get("times"), "rw");
167            FileLock fl = fis.getChannel().tryLock();
168            Scanner in = new Scanner(fis.getChannel());
169
170            PriorityQueue<CPTimeData> times = null;
171            Entrant entrant;
172
173            //clear out the entrants times and reset
174            for (Entry<Integer, Entrant> entry : entrants.entrySet()) {
175                entrant = (Entrant) entry.getValue();
176                entrant.setTimes(new ArrayList<CPTimeData>());
177                entrant.resetPosition();
178            }
179
180            //if we have locked the file
181            if(fl != null) {
182                times = new PriorityQueue<CPTimeData>();
183
184                while (in.hasNext()) {
185                    CPTimeData chkpoint = new CPTimeData();
186                    char type = in.next().charAt(0);
187                    int node = in.nextInt();
188                    int entrantNo = in.nextInt();
189                    Date date = sdf.parse(in.next());
190                    entrant = entrants.get(entrantNo);
191
192                    //exclude entrant if necessary
193                    switch(type) {
194                        case 'I':
195                        case 'E':
196                            entrant.setExcluded(true);
197                            break;
198                    }
199
200                    //create checkpoint update data
201                    chkpoint.setUpdateType(type);
202                    chkpoint.setNode(node);
203                    chkpoint.setEntrantId(entrantNo);
204                    chkpoint.setTime(date);
205
206                    Course course = courses.get(entrant.getCourse());
207                    if(entrant.getPosition() >= course.getLength()-2) {
208                        entrant.setFinished(true);
209                    }
210
211                    //update entrant and times list.
212                    entrant.incrementPosition();
213                    entrant.addTime(chkpoint);
214                    times.add(chkpoint);
215                }
216
217                fl.release();
218            }
219
220            in.close();
221            fis.close();
222
223            return times;
224        }
225
226        /**
```

```
227         * Read in the checkpoints file.
228         *
229         * @return the LinkedHashMap of checkpoints (nodes) identified by ID.
230         * @throws FileNotFoundException exception thrown when file cannot be found.
231         * @throws IOException Signals that an unexpected I/O exception has occurred.
232         */
233        public LinkedHashMap<Integer, Checkpoint> readCheckpoints()
234                throws FileNotFoundException, IOException {
235            Scanner in = new Scanner(new File(filenames.get("checkpoints")));
236
237            LinkedHashMap<Integer, Checkpoint> checkpoints = new LinkedHashMap<Integer, Checkpoint>();
238
239            while(in.hasNext()) {
240                int id = in.nextInt();
241                String type = in.next();
242
243                //ignore junctions
244                if(!type.equals("JN")) {
245                    checkpoints.put(id, new Checkpoint(id, type));
246                }
247            }
248
249            in.close();
250
251            return checkpoints;
252        }
253
254        /**
255         * Write out time data to the times file.
256         *
257         * @param writer the PrintWriter to use to output the time
258         * @param data the data to output to file
259         * @throws FileNotFoundException exception thrown when file cannot be found.
260         * @throws IOException Signals that an unexpected I/O exception has occurred.
261         */
262        private void writeTimeData(PrintWriter writer, CPTimeData data) throws FileNotFoundException, IOException {
263            String time = data.getStringTime();
264            String output = data.getUpdateType() + " " + data.getNode() + " " + data.getEntrantId() + " " + time;
265            writer.write(output + "\n");
266            writer.flush();
267        }
268
269        /**
270         * Write out the list of times to file.
271         *
272         * @param times the list of times to output.
273         * @return true, if successful at writing
274         * @throws FileNotFoundException exception thrown when file cannot be found.
275         * @throws IOException Signals that an unexpected I/O exception has occurred.
276         */
277        public boolean writeTimes(PriorityQueue<CPTimeData> times) throws FileNotFoundException, IOException {
278            FileOutputStream fis = new FileOutputStream(new File(filenames.get("times")));
279            FileLock fl = fis.getChannel().tryLock();
280            PrintWriter writer = new PrintWriter(fis);
281            boolean writeSuccess = false;
282
283            //we have file lock
284            if(fl != null) {
285                while (!times.isEmpty()) {
286                    //get times in order of priority (oldest first)
287                    CPTimeData t = times.poll();
288                    writeTimeData(writer, t);
289                }
290                fl.release();
291                writeSuccess = true;
292            }
293
294            fis.close();
295            writer.close();
296
297
```

```
298             return writeSuccess;
299
300         }
301
302         /**
303          * Write to the log file.
304          *
305          * @param updateText the message to output to the log file
306          * @throws FileNotFoundException exception thrown when file cannot be found.
307          * @throws IOException Signals that an unexpected I/O exception has occurred.
308          * @return true, if successful at writing
309          */
310         public boolean writeLog(String updateText) throws FileNotFoundException, IOException {
311             String outputStr;
312             Date time = new Date();
313             FileOutputStream fis = new FileOutputStream(new File(filenames.get("log")), true);
314             FileLock fl = fis.getChannel().tryLock();
315             PrintWriter writer = new PrintWriter(fis);
316             boolean writeSuccess = false;
317             //we have file lock
318             if(fl != null) {
319                 outputStr = sdf.format(time) + " CMP: " + updateText + "\n";
320                 writer.append(outputStr);
321                 writer.flush();
322                 writeSuccess = true;
323             }
324             fis.close();
325             writer.close();
326
327             return writeSuccess;
328         }
329     }
```

Listing 27: ArgumentParseException.java

```
1
2   package checkpoint.manager.exceptions;
3
4   /**
5    * The Class ArguementParseException.
6    * Thrown if the command line arguments could not be parsed.
7    * @author Samuel Jackson (slj11@aber.ac.uk)
8    */
9   @SuppressWarnings("serial")
10  public class ArgumentParseException extends Exception{
11
12      /* (non-Javadoc)
13       * @see java.lang.Throwable#getMessage()
14       */
15      @Override
16      public String getMessage() {
17          return "Could not parse command line arguments";
18      }
19  }
```

2.2 Compilation Output

Below is the compilation output for the jar file for Checkpoint Manager application.

Listing 28: Build log of the C Event Manager Program

```
ant -f "/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager" clean jar
init:
deps-clean:
Updating property file:
```

```
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/built-clean.properties
Deleting directory
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build
clean:
init:
deps-jar:
Created dir:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build
Updating property file:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/built-jar.properties
Created dir:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/classes
Created dir:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/empty
Created dir:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/
generated-sources/ap-source-output
Compiling 10 source files to
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build/classes
warning: [options] bootstrap class path not set in conjunction with -source 1.6
Note:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/src/checkpoint/manager
/gui/CheckpointManagerGUI.java
uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
compile:
Created dir:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/dist
Copying 1 file to /home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/build
Nothing to copy.
Building jar:
/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/dist/Checkpoint_Manager.jar
To run this application from the command line without Ant, try:
java -jar
"/home/samuel/Dropbox/Aber/uni_docs/Paradigms/cs22510-runners-and-riders-redux/Checkpoint Manager/dist/Checkpoint_Manager.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

2.3 Example Run Of Checkpoint Manager

This section shows a run through of the Checkpoint manager application using to demonstrate its usage. This run through is using the event three example files supplied for the C programming assignment.

2.3.1 Adding a Correct Time

For the first example, I am demonstrating how to check in an entrant at with a valid time at a valid node. Below is a screen image showing this:

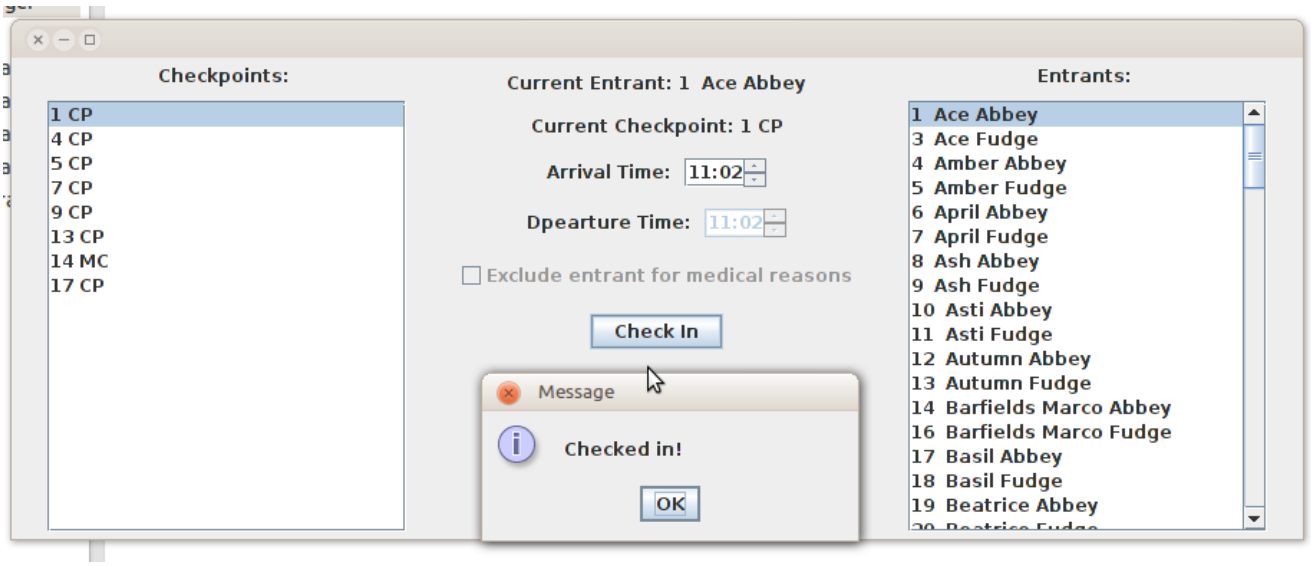

Figure 1: Checking in a valid entrant at a valid time.

2.3.2 Arriving at an Incorrect Node

This example shows attempting to check in the same entrant as in the previous step to the same node directly after signing them in. This results in an a notice that the entrant will be excluded for going the wrong way on their course.

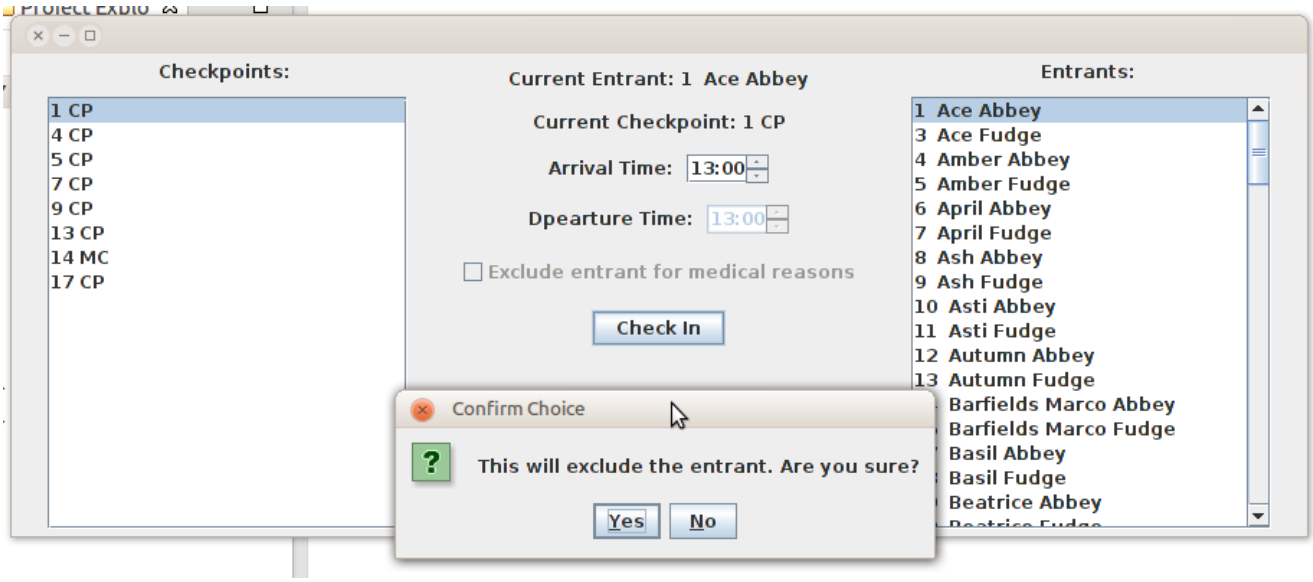

Figure 2: Checking in an entrant at an invalid node.

2.3.3 Arriving at Another Node at an Incorrect Time

The next example shows checking in an entrant to a valid node (node 4) but at an invalid time (e.g. at a time before the last node they we're checked in at).

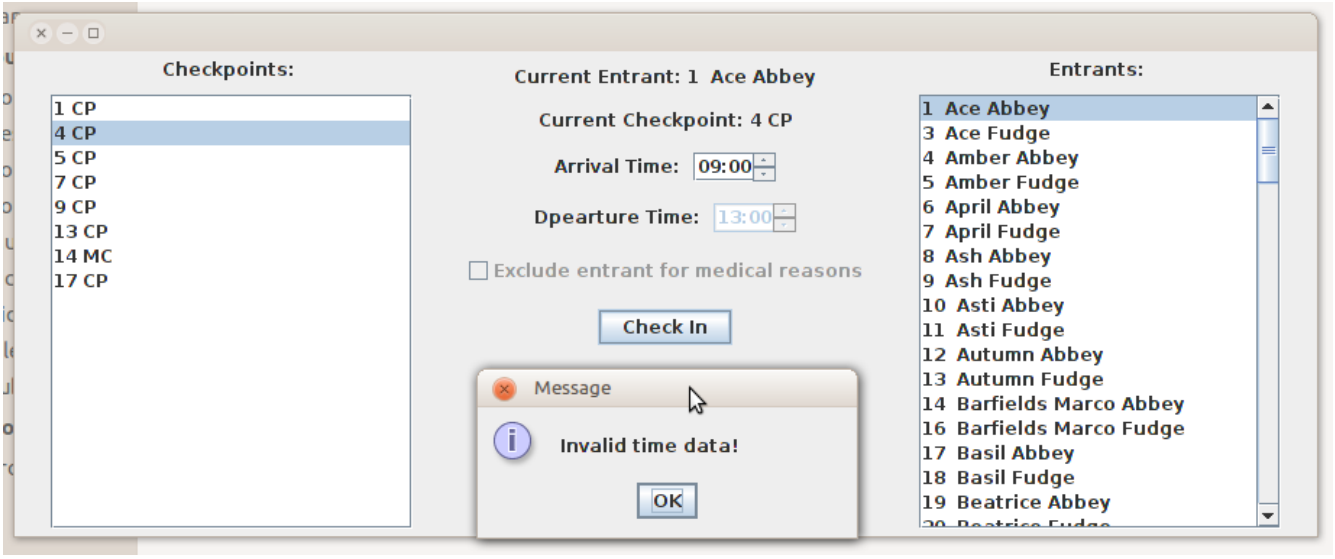

Figure 3: Checking in an entrant at a valid node but at an invalid time.

2.3.4 Checking into a Medical Checkpoint

The following example shows checking in an entrant to a medical checkpoint. Notice how the relevant parts of the form are no longer greyed out when the medical checkpoint is selected.

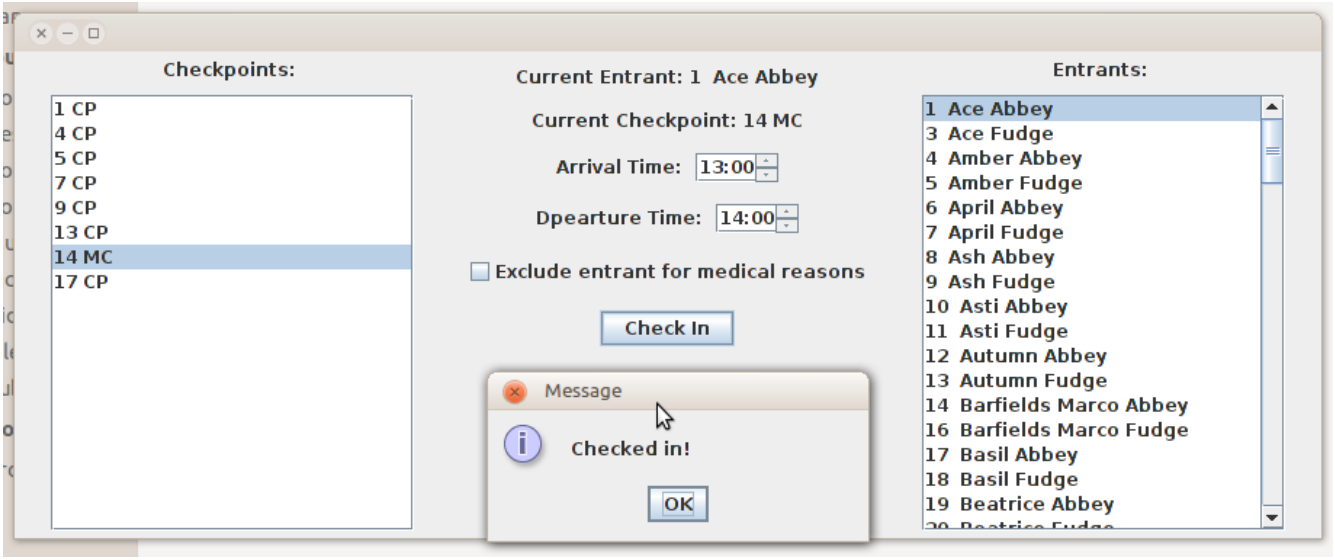

Figure 4: Checking in an entrant at a medical checkpoint using valid times.

2.3.5 Checking into a Medical Checkpoint with Incorrect Times

This example shows attempting to check an entrant into a medical checkpoint with a departure time that is before the arrival time.

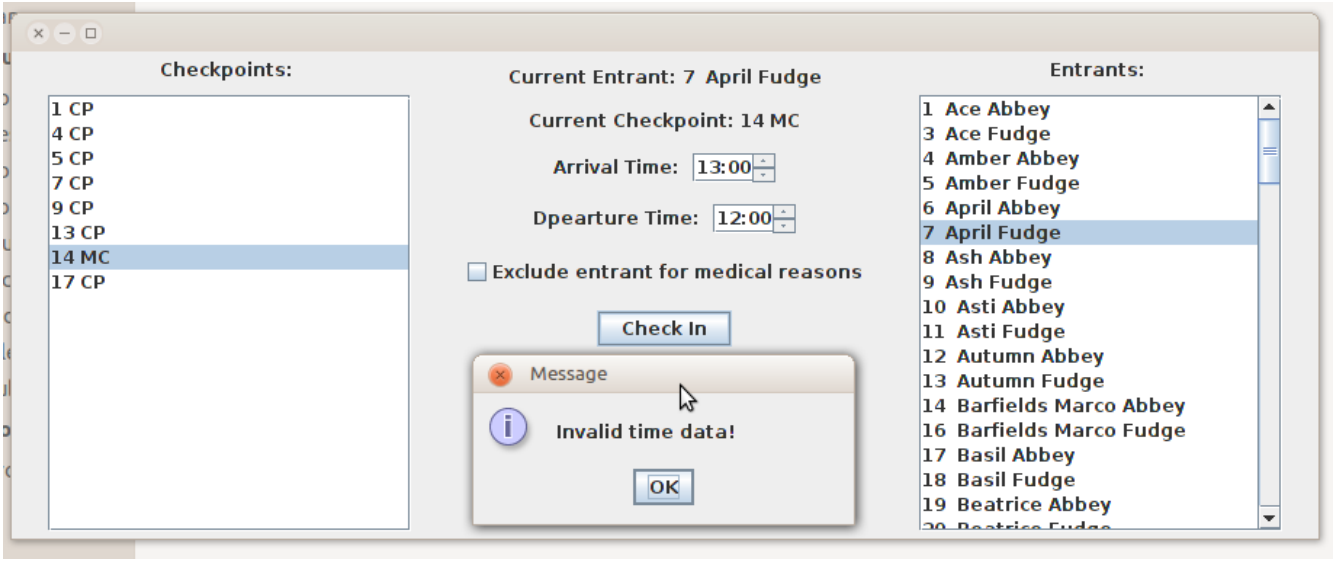

Figure 5: Checking in an entrant at a medical checkpoint but with invalid arrival/departure times.

2.3.6 Checking into a Medical Checkpoint and Excluding the Entrant

The following example demonstrates the user interface when the user chooses to exclude an entrant for medical reasons from the event. Note that excluding an entrant at a checkpoint for going the wrong way but not for medical reasons works the same as at a regular checkpoint.

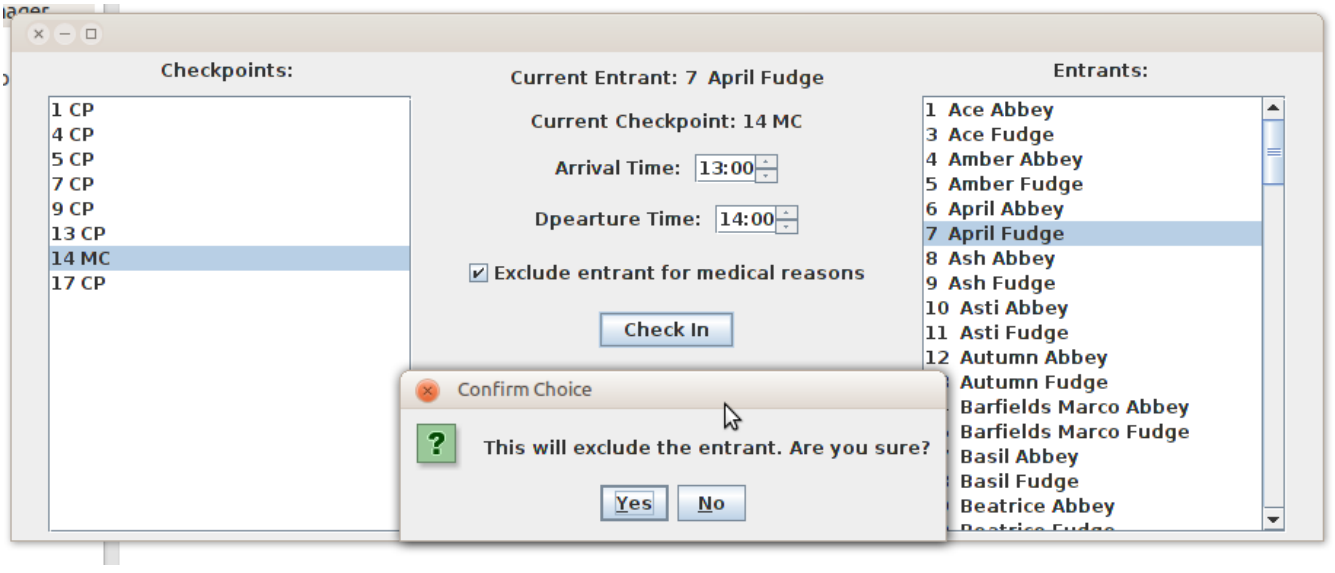

Figure 6: Checking in an entrant at a medical checkpoint and excluding them at the same time.

2.3.7 File Locking

The final screen image shows what happens when the application tries to read the times file when another application has it open for writing. To demonstrate this functionality, I had to have one version of the application running from a jar file and another version running in debug mode with a breakpoint on line 290 of listing 26 to prevent the file lock from releasing the lock on the file.

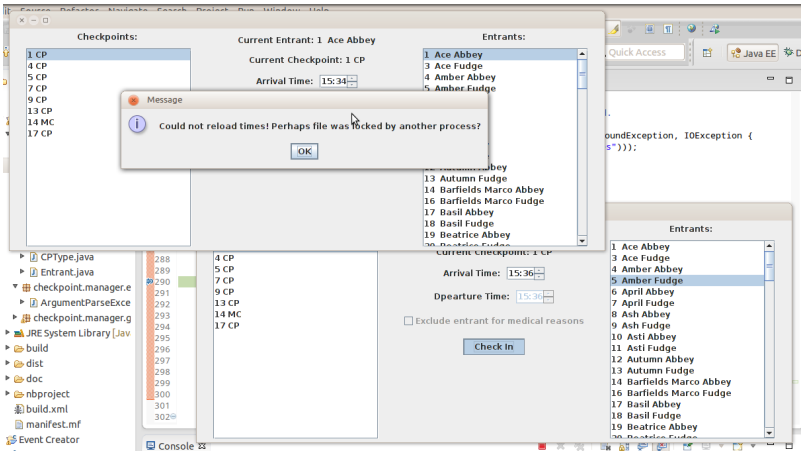

Figure 7: Demonstrating file locking works by putting a break point in the lower right hand side application to prevent the lock from releasing.

3 Event Manager Program Documentation

This section provides the documentation for the Event Manager program, written in C, including build logs and example output.

3.1 Compilation Output

Below is the full build log from the compilation of the Event Manager program.

Listing 29: Build log of the C Event Manager Program

```
12:27:50 **** Build of configuration Debug for project Event Manager ****
make all
Building file: ../fileio.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"fileio.d" -MT"fileio.d" -o "fileio.o" "../fileio.c"
Finished building: ../fileio.c

Building file: ../linked_list.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"linked_list.d" -MT"linked_list.d" -o "linked_list.o" "../linked_list.c"
Finished building: ../linked_list.c

Building file: ../main.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"main.d" -MT"main.d" -o "main.o" "../main.c"
Finished building: ../main.c

Building file: ../util.c
Invoking: GCC C Compiler
gcc -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"util.d" -MT"util.d" -o "util.o" "../util.c"
Finished building: ../util.c

Building target: Event Manager
Invoking: GCC C Linker
gcc -o "Event Manager" ./fileio.o ./linked_list.o ./main.o ./util.o
Finished building target: Event Manager


12:27:50 Build Finished (took 415ms)
```

3.2 Example Run Output

The following section shows the output generated from running the event manager program for each type of query a user can make with the system. This run was carried out using event one example files.

Listing 30: Output of an example run of the C program

```
              |=============================================|
|--------------RUNNERS AND RIDERS--------------|
|---------------------------------------------|
|----------Event tracking application----------|
|=============================================|

Loading data files...
Enter location and name of event details file:
../event_1/name.txt
Enter location and name of the nodes file:
../event_1/nodes.txt
Enter location and name of the tracks file:
../event_1/tracks.txt
Enter location and name of the courses file:
../event_1/courses.txt
Enter location and name of the entrants file:
../entrants.txt
Error opening file:
../entrants.txt
Enter location and name of the entrants file:
../event_1/entrants.txt
Enter location and name of the log file:
../event_1/log.txt
Enter location and name of the times file:
../event_1/cp_times_1.txt
Enter an Option:
0 - Exit
1 - Event Details
2 - Query Competitor
3 - Check how many competitors not yet started
4 - Check how many competitors are out on courses
5 - Check how many competitors have finished
6 - Print table of results
7 - Print entrants excluded at medical checkpoints
8 - Print entrants excluded at regular checkpoints
1
Endurance Horse Race - Beginners Event
26th June 2012
Start Time: 07:30
Enter an Option:
0 - Exit
1 - Event Details
2 - Query Competitor
3 - Check how many competitors not yet started
4 - Check how many competitors are out on courses
5 - Check how many competitors have finished
6 - Print table of results
7 - Print entrants excluded at medical checkpoints
8 - Print entrants excluded at regular checkpoints
2
Enter id for the competitor:
1
COMPETITOR 1:
Name: Donald Duck
Status: Out on track.
Last recorded time: 08:46
Last checkpoint visited: 9
Track Reference No.: 11
Presumed on track between node 3 and node 9
Enter an Option:
0 - Exit
1 - Event Details
```

```
2 − Query Competitor
3 − Check how many competitors not yet started
4 − Check how many competitors are out on courses
5 − Check how many competitors have finished
6 − Print table of results
7 − Print entrants excluded at medical checkpoints
8 − Print entrants excluded at regular checkpoints
3
0 competitors have not yet started
Enter an Option:
0 − Exit
1 − Event Details
2 − Query Competitor
3 − Check how many competitors not yet started
4 − Check how many competitors are out on courses
5 − Check how many competitors have finished
6 − Print table of results
7 − Print entrants excluded at medical checkpoints
8 − Print entrants excluded at regular checkpoints
4
13 competitors are out on a course
Enter an Option:
0 − Exit
1 − Event Details
2 − Query Competitor
3 − Check how many competitors not yet started
4 − Check how many competitors are out on courses
5 − Check how many competitors have finished
6 − Print table of results
7 − Print entrants excluded at medical checkpoints
8 − Print entrants excluded at regular checkpoints
5
0 competitors have completed their course
Enter an Option:
0 − Exit
1 − Event Details
2 − Query Competitor
3 − Check how many competitors not yet started
4 − Check how many competitors are out on courses
5 − Check how many competitors have finished
6 − Print table of results
7 − Print entrants excluded at medical checkpoints
8 − Print entrants excluded at regular checkpoints
6
---------------------------------------------------------------------------------------------------
|Competitor | Course | Status | Start Time | End Time | MC Delay | Total |
|--------------------------------------------------------------------------------------------------
|Donald Duck | D |ON TRACK | 07:30 | N/a | N/a | N/a |
|Mickey Mouse | D |ON TRACK | 07:35 | N/a | N/a | N/a |
|Jemima Julieta Mouse | E |ON TRACK | 07:39 | N/a | N/a | N/a |
|Minnie Duck | F |ON TRACK | 07:43 | N/a | N/a | N/a |
|Minnie Mouse | E |ON TRACK | 07:47 | N/a | N/a | N/a |
|Minnie Mouse Junior | E |ON TRACK | 07:51 | N/a | N/a | N/a |
|Deputy Doug | D |ON TRACK | 07:56 | N/a | N/a | N/a |
|Deputy Duck | D |ON TRACK | 08:01 | N/a | N/a | N/a |
|Bewick Swan | F |ON TRACK | 08:05 | N/a | N/a | N/a |
|Black Swan | F |TIME CHECKPOINT | 08:10 | N/a | N/a | N/a |
|Albert Einstein | E |ON TRACK | 08:14 | N/a | N/a | N/a |
|Albert Mouse | D |ON TRACK | 08:18 | N/a | N/a | N/a |
|Donald Duck Senior | E |ON TRACK | 08:22 | N/a | N/a | N/a |
|Egbert Einstein | F |ON TRACK | 08:26 | N/a | N/a | N/a |
---------------------------------------------------------------------------------------------------
Enter an Option:
0 − Exit
1 − Event Details
2 − Query Competitor
3 − Check how many competitors not yet started
4 − Check how many competitors are out on courses
5 − Check how many competitors have finished
6 − Print table of results
7 − Print entrants excluded at medical checkpoints
```

```
8 - Print entrants excluded at regular checkpoints
7
Competitors Excluded from Medical Checkpoints
-----------------------------------------
|Competitor | Node | Time |
|----------------------------------------|
-----------------------------------------
Enter an Option:
0 - Exit
1 - Event Details
2 - Query Competitor
3 - Check how many competitors not yet started
4 - Check how many competitors are out on courses
5 - Check how many competitors have finished
6 - Print table of results
7 - Print entrants excluded at medical checkpoints
8 - Print entrants excluded at regular checkpoints
8
Competitors Excluded from Regular Checkpoints
-----------------------------------------
|Competitor | Node | Time |
|----------------------------------------|
-----------------------------------------
Enter an Option:
0 - Exit
1 - Event Details
2 - Query Competitor
3 - Check how many competitors not yet started
4 - Check how many competitors are out on courses
5 - Check how many competitors have finished
6 - Print table of results
7 - Print entrants excluded at medical checkpoints
8 - Print entrants excluded at regular checkpoints
0
```

3.3 Example Run Results List

This sections show a listing of the results table for an event, including a selection of competitors in a variety of states (e.g. excluded, completed, not started etc.)

Listing 31: The results table. With competitors in various states

```
-----------------------------------------------------------------------------------------------------
|Competitor | Course | Status | Start Time | End Time | MC Delay | Total |
|----------------------------------------------------------------------------------------------------
|Donald Duck | D |COMPLETED | 15:32 | 19:32 | 00hrs 00mins | 04hrs 00mins |
|Mickey Mouse | D |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Jemima Julieta Mouse | E |COMPLETED | 15:35 | 18:35 | 00hrs 00mins | 03hrs 00mins |
|Minnie Duck | F |COMPLETED | 15:30 | 18:30 | 00hrs 00mins | 03hrs 00mins |
|Minnie Mouse | E |EXCLUDED IR | 00:00 | N/a | N/a | N/a |
|Minnie Mouse Junior | E |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Deputy Doug | D |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Deputy Duck | D |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Bewick Swan | F |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Black Swan | F |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Albert Einstein | E |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Albert Mouse | D |NOT STARTED | 00:00 | N/a | N/a | N/a |
|Donald Duck Senior | E |EXCLUDED IR | 00:00 | N/a | N/a | N/a |
|Egbert Einstein | F |NOT STARTED | 00:00 | N/a | N/a | N/a |
-----------------------------------------------------------------------------------------------------
```

3.4 Output Of Log File

Below is the output of the log file after using the Checkpoint and Event manager applications to create the previous output.

Listing 32: Output from the log file generated when creating the results shown in the listing 31

```
16:30 CMP: Read the times file.
16:31 CMP: Read the times file.
16:31 CMP: Checked in entrant 4 @ node 1
16:31 CMP: Read the times file.
16:31 CMP: Checked in entrant 4 @ node 9
16:31 CMP: Read the times file.
16:31 CMP: Checked in entrant 4 @ node 13
16:31 CMP: Read the times file.
16:31 CMP: Read the times file.
16:31 CMP: Checked in entrant 4 @ node 1
16:32 CMP: Read the times file.
16:32 CMP: Checked in entrant 1 @ node 1
16:32 CMP: Read the times file.
16:32 CMP: Checked in entrant 1 @ node 4
16:32 CMP: Read the times file.
16:32 CMP: Read the times file.
16:32 CMP: Checked in entrant 1 @ node 5
16:32 CMP: Read the times file.
16:32 CMP: Checked in entrant 1 @ node 9
16:32 CMP: Read the times file.
16:32 CMP: Checked in entrant 1 @ node 1
16:34 CMP: Read the times file.
16:34 CMP: Checked in entrant 3 @ node 1
16:34 CMP: Read the times file.
16:34 CMP: Read the times file.
16:34 CMP: Checked in entrant 3 @ node 9
16:34 CMP: Read the times file.
16:34 CMP: Checked in entrant 3 @ node 13
16:35 CMP: Read the times file.
16:35 CMP: Checked in entrant 3 @ node 1
16:43 CMP: Read the times file.
16:43 CMP: Read the times file.
16:43 CMP: Checked in entrant 5 @ node 4
16:44 CMP: Read the times file.
16:44 CMP: Checked in entrant 13 @ node 4
16:45 EMP: Loaded the times file.
16:45 EMP: Viewed a list of results
16:45 EMP: Loaded the times file.
```

4 Outline of Programs

4.1 Event Creation Program

The event creation program is a command line based application written in C++. Its purpose is to create the event, courses and entrants file for each event. The design of the application allows the user to create multiple events at the same time, rather than having to make each event in serial. Because entrants need a course and a course needs an event, an event must be created before a course and a course must be created before an entrant. This includes the functionality to create different course and entrants associated with different events. Each event also expects a nodes file to be given when creating the event, allowing different events to work with different sets of allowed nodes. The user is also able to view an event by selecting the relevant option form the main menu.

Since lists of courses and entrants are associated with each event, I decided that the best approach would be to allow the user to create all the data about an event, then write it to file, rather than creating each of the files one at a time. When the user chooses the option to write an event, a new folder is created with the name of the event as the name of the folder. Inside the folder, the event, entrants and courses files are written.

Note that as the brief made no mention of courses relying on the input of the tracks file, I have assumed that the tracks file is generated from the courses file and not visa versa. Integration of the creation of the tracks file alongside courses might be a nice enhancement in any future development.

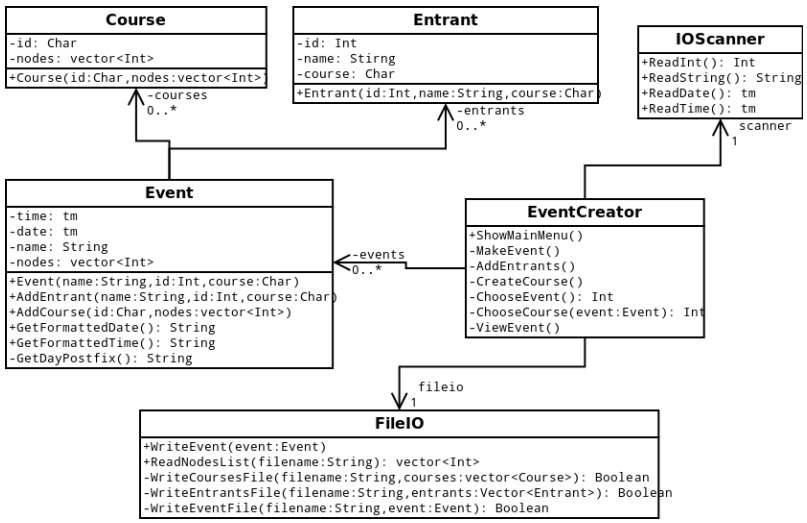

Figure 8: Class diagram of the Event Creator program. Getters/Setters not shown.

4.2 Checkpoint Manager Program

The checkpoint manager program is written in Java and provides a Swing based GUI to allow the user to easily update entrants out in the field as the JVM allows the program to be executed on a variety of platforms. This program accepts the required files (entrants, courses, nodes, time and log files) as command line arguments using flags for each file. Help instructions are printed when no arguments or incorrect arguments are supplied. An example listing of arguments is supplied below:

```
java -jar checkpoint_manager.jar -E ../../event_3/entrants.txt -C ../../event_3/courses.txt -K ../../event_3/nodes.txt
 -T ../../event_3/times.txt -L ../../event_3/log.txt
```

The checkpoint manager program allows a race marshal to update the location of the entrants as they arrive at the various checkpoints on the course. Entrants are automatically excluded if checked into a checkpoint they should not

of visited. The GUI also provides an option for marshals to excluded entrants based on failing a medical checkpoint. When an entrant is excluded, they are automatically removed from the list of available entrants. When an entrant is about to be excluded, the user is asked to confirm the operation, ensuring that they don't accidentally excluded a competitor.

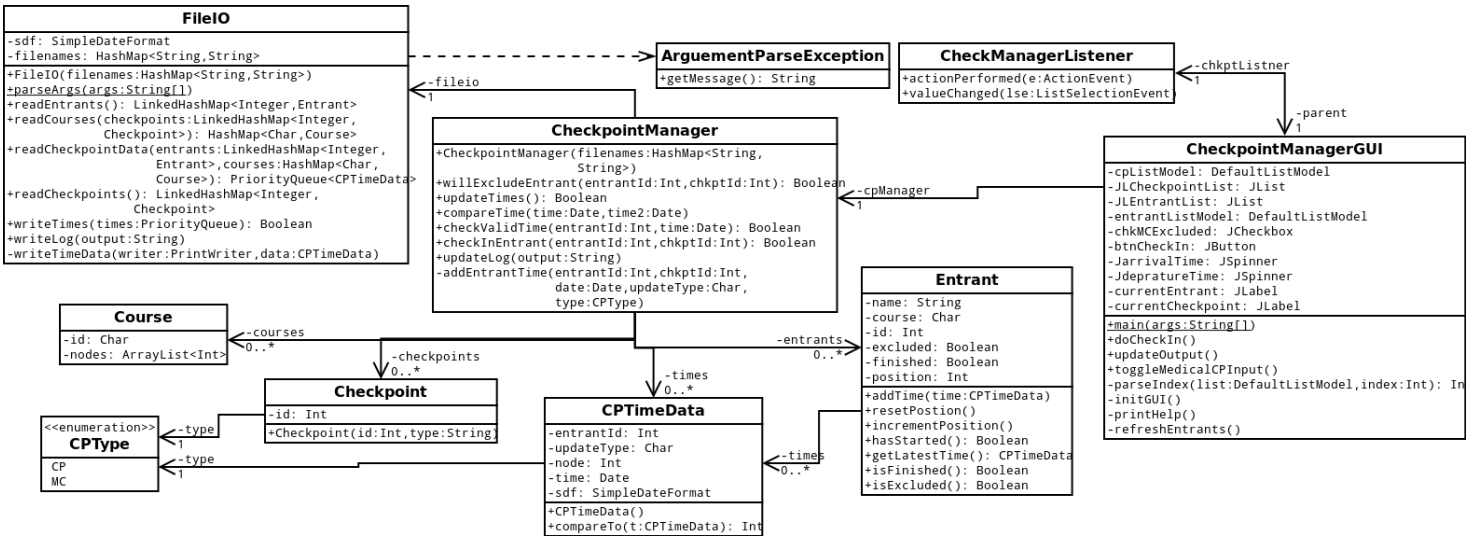

Figure 9: Class diagram of the Checkpoint Manager program. Getters/Setters not shown.

The event manager program allows the user to input the time a competitor arrives and, in the case of medical checkpoints, departs. The program automatically checks that the arrival time is greater than the last time the entrant was checked in. In the case of medical checkpoints, it also checks that the arrival time is not greater than the departure time. Correct order of times is tracked using a priority queue.

One small note on the design of the Java program is that I would of liked to have implemented the GUI to better adhere to the model view controller design principles as I feel this would have made for a cleaner communication model between the GUI and data model.

4.3 Event Manager Program

The event manager program is written in C and handles checking the position and state of entrants as they progress through a course. This application has changed only slightly from the previous C assignment. Changes include removal of manual entry, automatically reloading the times file, file locking and writing to the log file. This includes viewing a list of which entrants have been excluded, finished and are currently out on a track. It also gives the user the ability to query individual competitors and provides an estimate of what track/node they should/are on.

The event manager requires the loading of all the data files for an event. This is done by prompting the user at the start of the application and only needs to be done once. Like the event manager, the application locks the log and times file when reading to prevent multiple applications crashing during file processing.

4.4 Summary

In summary of the three programs, I feel that they all adequately achieve the aims laid out in the brief, although I also strongly feel that there is plenty of room for improvement. One of the largest areas that could be improved would be better design of the file reading and writing, functions to intelligently handle badly formed files when parsing. The C and Java programs could also be improved to better handle the when trying to read/write to a locked file than simply throwing an error message.