CS22510 - C++, C & Java Programming Paradigms

Assignment 1 - Runners & Riders

2012-2013

Connor Luke Goddard

clg11@aber.ac.uk

March 2013

Contents

1 Introduction

1.1 Purpose of this Document

The purpose of this document is to provide a description and supporting evidence of my implemented solution to the CS22510 Assignment 1.

1.2 Scope

This document describes the final state of the implemented solution and contains evidence demonstrating the functionality, compilation, and source code of all three applications that form to produce the final system.

1.3 Objectives

The objectives of this document are:

- To provide the complete source code for the "event creator" application, and evidence of it's compilation and functionality.

- To provide the complete source code for the "checkpoint manager" application, and evidence of it's compilation and functionality.

- To provide evidence of the compilation and functionality of the "event manager" applicaiton.

- To briefly describe the structure and programming language choice of each of the three applications.

2 Design Assumptions

To allow a definitive design for the checkpoint manager to be established, the following assumption has had to be drawn:

1. When logging new time values for **medical checkpoints**, a user will submit arrival and departure times **separatly** (i.e. separate transactions), and not at the same time. This will allow other entrant checkpoint time values to be logged while an entrant is waiting at a medcial checkpoint.

3 Event Creator (C++) - Source Code

This section contains the complete source code for the "event creator" program written in C++.

3.1 Header Files

3.1.1 Menu.h

```
1  /*
2   * File: Menu.h
3   * Description: Defines all variables/methods for the Course class.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #ifndef MENU_H
10 #define MENU_H
11
12 #include "Process.h"
13 #include "Course.h"
14 #include "FileIO.h"
15
16 /**
17  * Used to provide interactive interface with the application through
18  * the use of menus.
19  */
20 class Menu {
21
22 public:
23
24     Menu(Datastore *newData, Process *newProc);
25     virtual ~Menu();
26     void showEventEditor(void);
27     void showCourseEditor(void);
28     void showEntrantEditor(void);
29     void showMainMenu(void);
30     void checkExistingEvent(void);
31
32 private:
33
34     /** Pointer to shared Process class created in "main.cpp".*/
35     Process *proc;
36
37     /** Pointer to shared Datastore class created in "main.cpp".*/
38     Datastore *data;
39
40     /** Allows access to file I/O methods.*/
41     FileIO io;
42 };
43
44 #endif  /* MENU_H */
```

3.1.2   Process.h

```
 1  /*
 2   * File: Process.h
 3   * Description: Defines all variables/methods for the Process class.
 4   * Author: Connor Luke Goddard (clg11)
 5   * Date: March 2013
 6   * Copyright: Aberystwyth University, Aberystwyth
 7   */
 8
 9  #ifndef PROCESS_H
10  #define PROCESS_H
11
12  #include <vector>
13  #include <cstdlib>
14  #include "Entrant.h"
15  #include "Node.h"
16  #include "Course.h"
17  #include "FileIO.h"
18  #include "Datastore.h"
19  #include "Event.h"
20
21  class Process {
22
23  public:
24
25      Process(Datastore *newData);
26      virtual ~Process();
27      void addEntrant(void);
28      void createEvent(void);
29      void getAllNodes(void);
30      void showCourseEditor(void);
31      void createNewCourse(void);
32      Course* getSelectedCourse(void);
33      void addCourseNode(Course *currentCourse);
34      std::string convertDate(std::string &input);
35
36  private:
37
38      /** Allows access to file I/O methods.*/
39      FileIO io;
40
41      /** Pointer to shared Datastore class created in "main.cpp".*/
42      Datastore *data;
43  };
44
45  #endif  /* PROCESS_H */
```

3.1.3 Datastore.h

```
 1  /*
 2   * File: Datastore.h
 3   * Description: Defines all variables/methods for the Datastore class.
 4   * Author: Connor Luke Goddard (clg11)
 5   * Date: March 2013
 6   * Copyright: Aberystwyth University, Aberystwyth
 7   */
 8
 9  #ifndef DATASTORE_H
10  #define DATASTORE_H
11
12  #include <vector>
13  #include <cstdlib>
14  #include "Entrant.h"
15  #include "Node.h"
16  #include "Course.h"
17  #include "Event.h"
18
19  /**
20   * Datastore class used for storing and providing access to all the of
21   * shared data used throughout the application..
22   */
23  class Datastore {
24  public:
25      Datastore();
26      virtual ~Datastore();
27      std::vector<Course*> getCourseList(void);
28      std::vector<Node*> getNodeList(void);
29      std::vector<Entrant*> getEntrantList(void);
30      Event* getEvent(void) const;
31      void addNewCourse (Course* newCourse);
32      void addNewNode (Node* newNode);
33      void addNewEntrant (Entrant* newEntrant);
34      void setNewEvent(Event* newEvent);
35      Course* getInCourse (char courseID);
36      Node* obtainNode (int nodeNo);
37
38  private:
39
40      /** Vector of pointers to all Entrant objects created. */
41      std::vector<Entrant*> entrantList;
42
43      /** Vector of pointers to all Nodes objects read into the system. */
44      std::vector<Node*> nodeList;
45
46      /** Vector of pointers to all Course objects created. */
47      std::vector<Course*> courseList;
48
49      /** Pointer to Event object used to define the race event. */
50      Event *event;
51  };
52
53  #endif  /* DATASTORE_H */
```

3.1.4   FileIO.h

```
1   /*
2    * File: FileIO.h
3    * Description: Defines all variables/methods for the FileIO class.
4    * Author: Connor Luke Goddard (clg11)
5    * Date: March 2013
6    * Copyright: Aberystwyth University, Aberystwyth
7    */
8
9   #ifndef FILEIO_H
10  #define FILEIO_H
11
12  #include "Entrant.h"
13  #include "Course.h"
14  #include "Event.h"
15
16  class FileIO {
17  public:
18      FileIO();
19      virtual ~FileIO();
20      void writeEntrants(std::vector<Entrant*> entrantList);
21      void writeCourses(std::vector<Course*> courseList);
22      void writeEvent(Event *event);
23      std::vector<std::vector<std::string> > getFile(std::string fileName);
24  private:
25
26      /**
27       * Vector of vectors used to store the contents of individual line
28       * that collect to form the entire file.
29       */
30      std::vector<std::vector<std::string> > arrayTokens;
31  };
32
33  #endif  /* FILEIO_H */
```

3.1.5   Event.h

```
 1  /*
 2   * File: Event.h
 3   * Description: Defines all variables/methods for the Entrant class.
 4   * Author: Connor Luke Goddard (clg11)
 5   * Date: March 2013
 6   * Copyright: Aberystwyth University, Aberystwyth
 7   */
 8
 9  #ifndef EVENT_H
10  #define EVENT_H
11
12  #include <string>
13
14  /**
15   * Event class used to define the data model for a particular event.
16   */
17  class Event {
18
19  public:
20      Event();
21      Event(std::string newEventName, std::string newEventDate, std::string
                newEventTime);
22      virtual ~Event();
23      void setEventTime(std::string eventTime);
24      std::string getEventTime(void) const;
25      void setEventDate(std::string eventDate);
26      std::string getEventDate(void) const;
27      void setEventName(std::string eventName);
28      std::string getEventName(void) const;
29
30  private:
31      std::string eventName; /**< The name/description of the event.*/
32      std::string eventDate; /**< The date that the event is to take place.*/
33      std::string eventTime; /**< The starting time of the event.*/
34  };
35
36  #endif  /* EVENT_H */
```

3.1.6 Entrant.h

```
1   /*
2    * File: Entrant.h
3    * Description: Defines all variables/methods for the Entrant class.
4    * Author: Connor Luke Goddard (clg11)
5    * Date: March 2013
6    * Copyright: Aberystwyth University, Aberystwyth
7    */
8
9   #ifndef ENTRANT_H
10  #define ENTRANT_H
11
12  #include <string>
13
14  /**
15   * Entrant class used to define the data model for a particular entrant.
16   */
17  class Entrant {
18
19  public:
20      Entrant();
21      Entrant(const std::string &theName, const int theEnNo, char theCourseID);
22      virtual ~Entrant();
23      void print(void);
24      std::string getEntrantName(void);
25      int getEntrantNo(void);
26      char getCourseID(void);
27  private:
28      std::string entrantName; /**< The name of the entrant.*/
29      const int entrantNo; /**< The unique number for the entrant.*/
30      char courseID; /**< The ID that the entrant is registered for.*/
31  };
32
33  #endif   /* ENTRANT_H */
```

3.1.7   Node.h

```
1   /*
2    * File: Node.h
3    * Description: Defines all variables/methods for the Node class.
4    * Author: Connor Luke Goddard (clg11)
5    * Date: March 2013
6    * Copyright: Aberystwyth University, Aberystwyth
7    */
8
9   #ifndef NODE_H
10  #define NODE_H
11
12  #include <string>
13
14  /**
15   *  Course class used to define the data model for a particular course node.
16   */
17  class Node {
18
19  public:
20      Node();
21      Node(const int newNodeNo, std::string newNodeType);
22      virtual ~Node();
23      void setNodeType(std::string nodeType);
24      std::string getNodeType(void) const;
25      const int getNodeNo(void) const;
26
27  private:
28      const int nodeNo; /**< Unique number that represents a node.*/
29      std::string nodeType; /**< Contains the type of node.*/
30  };
31
32  #endif  /* NODE_H */
```

3.1.8 Course.h

```cpp
/*
 * File: Course.h
 * Description: Defines all variables/methods for the Course class.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include <vector>
#include "Node.h"

#ifndef COURSE_H
#define COURSE_H

/**
 *  Course class used to define the data model for an event course.
 */
class Course {

public:
    Course();
    Course(const char theCourseID);
    virtual ~Course();
    void addCourseNode(Node *newNode);
    void setCourseSize(int courseSize);
    int getCourseSize(void) const;
    std::vector<Node*> getCourseNodes(void) const;
    const char getCourseID(void) const;
private:

    const char courseID; /**< Unique ID for a Course.*/
    std::vector<Node*> courseNodes; /**< Vector of all nodes that make up a course.
         */
    int courseSize; /**< The total number of nodes in the course. */
};

#endif  /* COURSE_H */
```

3.2   Class Files

3.2.1   Main.cpp

```cpp
/*
 * File: main.cpp
 * Description: Bootstrap loader for the application.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include <cstdlib>
#include "Process.h"
#include "Menu.h"

using namespace std;

/**
 * Bootstrap method for the application.
 */
int main(int argc, char** argv) {

    /**
     * New Datastore object created on the heap
     * that is used throughout the program.
     */
    Datastore *data = new Datastore();

    /**
     * New Process object created on the heap
     * that is used throughout the program.
     */
    Process *proc = new Process(data);

    /**
     * New Menu object created on the heap
     * that will display the main menu.
     */
    Menu *menu = new Menu(data, proc);

    //Load course nodes into system from "nodes.txt" file.
    proc->getAllNodes();

    //Display the main program menu.
    menu->showMainMenu();

  return 0;
}
```

3.2.2   Menu.cpp

```cpp
/*
 * File: Menu.cpp
 * Description: Generates system menus to provide a means of interacting with
 * the application.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */


#include <vector>
#include <iostream>
#include <limits>
#include <string.h>
#include "Menu.h"

using namespace std;

/**
 * Constructor for Menu that allows access to Process and Datastore classes
 * created in "main.cpp". This allows Menu to access the same data stored in
 * Datastore as the Process class.
 * @param newData Pointer to the shared Datastore class created in the main method.
 * @param newProc Pointer to the shared Process class created in the main method.
 */
Menu::Menu(Datastore *newData, Process *newProc) {

    data = newData;
    proc = newProc;

}

/**
 * Destructor to be used once object is removed.
 * Removes the objects stored on the heap.
 */
Menu::~Menu() {

    delete data;
    delete proc;
}

/**
 * Provides top-level interactive menu to allow user to interact with the
 * application and utilise its functions.
 */
void Menu::showMainMenu(void) {

    int x;

    while (x != 5) {

        cout << "\n*************************************\n"
                << "Welcome to the Event Creator.\n"
                << "Please select an option:\n"
                << "------------------------------------\n"
                << "1. Event Editor\n"
                << "2. Entrant Editor\n"
                << "3. Course Editor\n"
                << "4. Export ALL files.\n"
                << "5. Exit Program.\n"
                << "*************************************\n";

        cin >> x;
```

```
66          switch (x) {
67
68              case 1:
69
70                  showEventEditor();
71                  break;
72
73              case 2:
74
75                  showEntrantEditor();
76                  break;
77
78              case 3:
79
80                  showCourseEditor();
81                  break;
82
83              case 4:
84
85                  /**
86                   * Export all data to their files.
87                   * As this method writes ALL the data, it has to check
88                   * that at least one instance of each object (Entrant, Event
89                   * and Course) exists before being able to write them all to file.
90                   */
91
92                  cout << "Writing all data to files...\n";
93
94                  //Check if an event has been created.
95                  if (data->getEvent() == NULL) {
96
97                      cout << "ERROR: No event created. Nothing to export.\n";
98
99                      //Check if any entrants have been created.
100                 } else if (data->getEntrantList().size() <= 0) {
101
102                     cout << "ERROR: No entrants created. Nothing to export.\n";
103
104                     //Check if any courses have been created.
105                 } else if (data->getCourseList().size() <= 0) {
106
107                     cout << "ERROR: No courses created. Nothing to export.\n";
108
109                 } else {
110
111                     //If there are no problems, write all the data to file.
112                     io.writeEvent(data->getEvent());
113                     io.writeEntrants(data->getEntrantList());
114                     io.writeCourses(data->getCourseList());
115                 }
116
117                 break;
118
119             case 5:
120                 cout << "Exiting...\n";
121                 break;
122             default:
123                 cout << "Incorrect option. Please try again.\n";
124         }
125     }
126 }
127
128 /**
129  * Provides sub-level interactive menu to allow user to create new
130  * courses and write them to file.
131  */
132 void Menu::showCourseEditor(void) {
```

```cpp
133
134        int x;
135
136        while (x != 4) {
137
138            cout << "\n*************************************\n"
139                    << "Course Editor | Please make a choice:\n"
140                    << "-------------------------------------\n"
141                    << "1. Create a new course.\n"
142                    << "2. Add a new node to existing course.\n"
143                    << "3. Export courses to file.\n"
144                    << "4. Return to main menu.\n"
145                    << "*************************************\n";
146
147            cin >> x;
148
149            switch (x) {
150
151                case 1:
152
153                    proc->createNewCourse();
154
155                    break;
156
157                case 2:
158                {
159                    Course *newCourse = NULL;
160
161                    //Prompt user for the ID of the course they wish to edit.
162                    newCourse = proc->getSelectedCourse();
163
164                    //If the specified course does not exist..
165                    if (newCourse == NULL) {
166
167                        //.. inform the user.
168                        cout << "ERROR: Course does not exist. Please try again";
169
170                    //Otherwise if the course does exist..
171                    } else {
172
173                        //Prompt the user for the node they wish to add and add it.
174                        proc->addCourseNode(newCourse);
175
176                    }
177
178                    break;
179                }
180                case 3:
181
182                    cout << "Exporting all courses to file.\n";
183
184                    //Check if any courses have been created.
185                    if (data->getCourseList().size() > 0) {
186                        io.writeCourses(data->getCourseList());
187                    } else {
188                        cout << "\nERROR: No courses created. Nothing to export.\n";
189                    }
190
191                    break;
192
193                case 4:
194                    cout << "Returning to main menu...\n";
195                    break;
196                default:
197                    cout << "Incorrect option. Please try again.\n";
198            }
199        }
```

```cpp
200  }
201
202  /**
203   * Provides sub-level interactive menu to allow user to create new
204   * entrants and write them to file.
205   */
206  void Menu::showEntrantEditor(void) {
207
208      int x;
209
210      while (x != 3) {
211
212          cout << "\n*************************************\n"
213                  << "Entrant Editor | Please make a choice:\n"
214                  << "---------------------------------\n"
215                  << "1. Create a new entrant.\n"
216                  << "2. Export entrants to file.\n"
217                  << "3. Return to main menu.\n"
218                  << "*************************************\n";
219
220          cin >> x;
221
222          switch (x) {
223
224              case 1:
225
226                  //Flush the input buffer to prevent skipping on "getline()".
227                  cin.ignore(numeric_limits<streamsize>::max(), '\n');
228
229                  //Run method to add a new entrant.
230                  proc->addEntrant();
231                  break;
232
233              case 2:
234
235                  cout << "Exporting all entrants to file.\n";
236
237                  //Check is any entrants have been created.
238                  if (data->getEntrantList().size() > 0) {
239                      io.writeEntrants(data->getEntrantList());
240                  } else {
241                      cout << "\nERROR: No entrants created. Nothing to export.\n";
242                  }
243
244                  break;
245
246              case 3:
247                  cout << "Returning to main menu...\n";
248                  break;
249              default:
250                  cout << "Incorrect option. Please try again.\n";
251          }
252      }
253  }
254
255  /**
256   * Provides sub-level interactive menu to allow user to create a new
257   * event and write it's details to file.
258   */
259  void Menu::showEventEditor(void) {
260
261      int x;
262
263      while (x != 3) {
264
265          cout << "\n*************************************\n"
266                  << "Event Editor | Please make a choice:\n"
```

```
267                        << "------------------------------------\n"
268                        << "1. Create new event.\n"
269                        << "2. Write event to file.\n"
270                        << "3. Return to main menu.\n"
271                        << "************************************\n";
272
273            cin >> x;
274
275            switch (x) {
276
277                case 1:
278
279                    //Flush the input buffer to prevent skipping on "getline()".
280                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
281
282                    //Perform check to see if an event has already been created.
283                    checkExistingEvent();
284                    break;
285
286                case 2:
287
288                    cout << "Exporting event to file.\n";
289
290                    //Check to see if an event had been created.
291                    if (data->getEvent() != NULL) {
292                        io.writeEvent(data->getEvent());
293                    } else {
294                        cout << "\nERROR: No event created. Nothing to export.\n";
295                    }
296
297                    break;
298
299                case 3:
300                    cout << "Returning to main menu...\n";
301                    break;
302                default:
303                    cout << "Incorrect option. Please try again.\n";
304            }
305        }
306 }
307
308 /**
309  * Checks to see if an existing event has already been created in this session,
310  * and provides suitable prompting and error checking as required.
311  */
312 void Menu::checkExistingEvent(void) {
313
314     char input;
315
316     //Check to see if the 'event' pointer in Datastore has been set to an Event
            object.
317     if (data->getEvent() != NULL) {
318
319         //If an event has already been created, prompt user for confirmation.
320         while (!((input == 'y') || (input == 'n') || (input == 'Y') || (input == 'N
                '))) {
321
322             cout << "WARNING: An event has already been created.\n";
323             cout << "Do you wish to create a new event? (Y/N)\n";
324             cin >> input;
325
326             switch (input) {
327
328                 case 'Y':
329                 case 'y':
330
331                     //Flush the input buffer to prevent skipping on "getline()".
```

```
332                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
333
334                        //Prompt user for event information and create the new event.
335                        proc->createEvent();
336                        break;
337
338                    //If the answer is no, nothing needs to happen.
339                    //Case is left in to prevent system thinking 'n/N' keys are
                            incorrect.
340                    case 'N':
341                    case 'n':
342                        break;
343                    default:
344                        cout << "Not a valid option. Please try again.\n";
345                        break;
346                }
347            }
348        } else {
349
350            //Otherwise if no event has been created as of yet, create a new one.
351            proc->createEvent();
352        }
353 }
```

3.2.3 Process.cpp

```
1   /*
2    * File: Process.cpp
3    * Description: Provides all core functionality and data processing for the
4    * application.
5    * Author: Connor Luke Goddard (clg11)
6    * Date: March 2013
7    * Copyright: Aberystwyth University, Aberystwyth
8    */
9
10  #include <vector>
11  #include <iostream>
12  #include <limits>
13  #include <ctime>
14  #include <sstream>
15  #include <algorithm>
16  #include "Process.h"
17
18  using namespace std;
19
20  /**
21   * Constructor for Process that allows access to the shared Datastore class
22   * created in "main.cpp".
23   * @param newData Pointer to the shared Datastore class created in the main method.
24   */
25  Process::Process(Datastore *newData) {
26
27      data = newData;
28
29  }
30
31  /**
32   * Destructor to be used once object is removed.
33   * Removes the objects stored on the heap.
34   */
35  Process::~Process() {
36
37      delete data;
38  }
39
40  /**
41   * Prompts user for input to define an event before creating a new
42   * 'Event' object and storing it's pointer in the shared Datastore class.
43   */
44  void Process::createEvent(void) {
45
46      string inputName, inputDate, inputTime, convertedDate;
47
48      cout << "Please enter an event name/description: ";
49
50      //Obtain all inputted characters including white space.
51      getline(cin, inputName);
52
53      cout << "Please enter the date of the event: (DD/MM/YYYY) ";
54      getline(cin, inputDate);
55
56      cout << "Please enter the time of the event: ";
57      getline(cin, inputTime);
58
59      //Convert inputted date string into format for writing to file.
60      convertedDate = convertDate(inputDate);
61
62      //Create a new Event object on the heap using the inputted information.
63      Event *newEvent = new Event(inputName, convertedDate, inputTime);
64
65      //Check if an Event object already exists on the heap.
```

```
66        if (data->getEvent() != NULL) {
67
68            //If it does remove it to prevent a memory leak.
69            delete data->getEvent();
70
71        }
72
73        //Set a pointer to the new object in the shared Datastore class.
74        data->setNewEvent(newEvent);
75
76        cout << "\nEvent (" << inputName << ") created successfully.\n";
77 }
78
79 /**
80  * Prompts user for input to define an entrant before creating a new
81  * 'Entrant' object and adding it's pointer to the
82  * vector of Entrant pointers contained in the shared Datastore class.
83  */
84 void Process::addEntrant(void) {
85
86        string entrantName;
87        char courseID, input;
88        int entrantNo;
89
90        cout << "Please enter a name: ";
91        getline(cin, entrantName);
92
93        //Prompt user to ask if they wish to specify their own entrant number.
94        while (!((input == 'y') || (input == 'n') || (input == 'Y') || (input == 'N')))
             {
95
96            cout << "Do you wish to set a manual entrant no? (Y/N)";
97            cin >> input;
98
99            switch (input) {
100
101                case 'Y':
102                case 'y':
103                {
104                    bool notExists = false;
105
106                    //Flush the input buffer to prevent input skipping.
107                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
108
109                    if (data->getEntrantList().size() > 0) {
110
111                        while (!notExists) {
112
113                            cout << "Please enter an entrant no: ";
114                            cin >> entrantNo;
115
116                            //Obtain a vector of ALL the entrants stored in Datastore
                                   node vector.
117                            vector<Entrant*> allEntrants = data->getEntrantList();
118
119                            //Loop through all the entrants.
120                            for (vector<Entrant*>::iterator it = allEntrants.begin();
                                   it != allEntrants.end(); ++it) {
121
122                                //Check to see if another entrant already has the
                                       entered value
123                                if ((*it)->getEntrantNo() == entrantNo) {
124
125                                    //If so break out of the loop as there should not
                                           be ANY matches.
126                                    notExists = false;
```

```
127                                    cout << "\nERROR: This entrant already exists.
                                           Please enter another value.\n";
128                                    break;
129
130                                } else {
131
132                                    //Otherwise if there is no match, then we can
                                           continue.
133                                    notExists = true;
134
135                                }
136                            }
137
138                        }
139
140                    } else {
141
142                        cout << "Please enter an entrant no: ";
143                        cin >> entrantNo;
144
145                    }
146
147                    break;
148
149                }
150                case 'N':
151                case 'n':
152
153                    cout << "Setting automatic entrant number\n";
154
155                    //Set entrant number to total numbeer of entrants + 1 (increment).
156                    entrantNo = (data->getEntrantList().size() + 1);
157                    break;
158                default:
159                    cout << "Not a valid option. Please try again.\n";
160                    break;
161            }
162        }
163
164        //Perform error checking to confirm entered course ID is a letter.
165        while (!isalpha(courseID)) {
166
167            cout << "Please enter a course ID: ";
168            cin >> courseID;
169
170            //If the user has not entered a letter, they must enter another value.
171            if (!isalpha(courseID)) {
172
173                cout << "ERROR: Course ID's can contain letters only. Please try again\
                       n";
174
175            }
176
177        }
178
179        cin.ignore(numeric_limits<streamsize>::max(), '\n');
180
181        //Create a new Entrant object on the heap using the inputted information.
182        Entrant *emp = new Entrant(entrantName, entrantNo, courseID);
183
184        //Add a pointer to the new object in the entrant vector stored in Datastore.
185        data->addNewEntrant(emp);
186
187        cout << "\nEntrant(" << entrantNo << ") created successfully.\n";
188
189    }
190
```

```cpp
191   /**
192    * Prompts user for the file path to the file that contains all the course
193    * node information, before processing and storing these nodes in a vector
194    * contained in the shared Datastore class.
195    */
196   void Process::getAllNodes(void) {
197
198       string fileName;
199
200       //Obtain the file path from the user.
201       cout << "Welcome. Please enter the file path for course nodes:\n";
202       getline(cin, fileName);
203
204       //Read-in all the node data from the file and store it all in a vector.
205       vector<vector<string> > fileContents = io.getFile(fileName);
206
207       //Check if the data has been successfully parsed and read-in.
208       if (fileContents.size() <= 0) {
209
210           cout << "ERROR: Nodes file (" << fileName << ") could not be located.\n\n"
211                   << "Please check the file path and try again. Exiting...\n";
212
213           //If the node data could not be loaded, terminate the program.
214           exit(EXIT_FAILURE);
215
216       } else {
217
218           //Loop through every line read-in from the file.
219           for (vector<vector<string> >::iterator it = fileContents.begin(); it !=
220               fileContents.end(); ++it) {
221
222               //Convert the first value on the line (node number) to an int.
223               int value = atoi((*it).at(0).c_str());
224
225               //Create a new Node object on the heap using the inputted information.
226               Node *tempNode = new Node(value, (*it).at(1));
227
228               //Add a pointer to the new object in the node vector stored in
229                   Datastore
230               data->addNewNode(tempNode);
231           }
232           cout << "Course nodes loaded successfully.\n" << "Loading program...\n\n";
233       }
234
235   }
236
237   /**
238    * Prompts user for input to define a course before creating a new
239    * 'Course' object and adding it's pointer to the
240    * vector of Course pointers contained in the shared Datastore class.
241    */
242   void Process::createNewCourse(void) {
243
244       char cid;
245       bool notExists = false;
246
247       if (data->getCourseList().size() > 0) {
248
249           while (!notExists) {
250
251               cid = 0;
252
253               //Check that the ID inputted by the user is a letter.
254               while (!isalpha(cid)) {
255
256                   //Prompt user for a course ID.
257                   cout << "Please enter a new course ID: ";
```

```
256                    cin >> cid;
257
258                    if (!isalpha(cid)) {
259
260                        cout << "ERROR: Course ID's can contain letters only. Please
                               try again\n";
261
262                    }
263
264                }
265
266            //Obtain a vector of ALL the courses stored in Datastore node vector.
267            vector<Course*> allCourses = data->getCourseList();
268
269            //Loop through all the stored courses.
270            for (vector<Course*>::iterator it = allCourses.begin(); it !=
                   allCourses.end(); ++it) {
271
272                //Check to see if another course already has the entered value.
273                if ((*it)->getCourseID() == cid) {
274
275                    notExists = false;
276
277                    //If so break out of the loop as there should not be ANY
                           matches.
278                    cout << "\nERROR: This course already exists. Please enter
                           another value.\n";
279                    break;
280
281                } else {
282
283                    //Otherwise if there is no match, then we can continue.
284                    notExists = true;
285
286                }
287            }
288        }
289
290    } else {
291
292        //Check that the ID inputted by the user is a letter.
293        while (!isalpha(cid)) {
294
295            //Prompt user for a course ID.
296            cout << "Please enter a new course ID: ";
297            cin >> cid;
298
299            if (!isalpha(cid)) {
300
301                cout << "ERROR: Course ID's can contain letters only. Please try
                       again\n";
302
303            }
304
305        }
306    }
307
308    //Create a new Course object on the heap using the inputted information.
309    Course *newCourse = new Course(cid);
310
311    //Add a pointer to the new object in the course vector stored in Datastore.
312    data->addNewCourse(newCourse);
313
314    cout << "\nCourse (" << cid << ") created successfully.\n";
315 }
316
317 /**
```

```
318    * Allows a user to specify a new node (loaded in from "nodes.txt") that
319    * that will form part of a particular course.
320    * @param currentCourse The course that a user wishes to add a new node to.
321    */
322   void Process::addCourseNode(Course *currentCourse) {
323
324       int nodeNo;
325
326       cout << "Please select a node to add: \n";
327
328       //Obtain a vector of ALL the course nodes stored in Datastore node vector.
329       vector<Node*> allNodes = data->getNodeList();
330
331       //Print all the nodes to screen to provide user with a list to select from.
332       for (vector<Node*>::iterator it = allNodes.begin(); it != allNodes.end(); ++it)
               {
333           cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << "), ";
334       }
335
336       cout << endl;
337
338       //Prompt user for the number of the node they wish to add.
339       cin >> nodeNo;
340
341       //Attempt to fetch the specified node from the vector of nodes in Datastore.
342       Node *tempNode = data->obtainNode(nodeNo);
343
344       //Check to see if a matching node was located.
345       if (tempNode != NULL) {
346
347           /**
348            * Add a pointer to the located node object in the course's vector
349            * of nodes.
350            */
351           currentCourse->addCourseNode(tempNode);
352           cout << "\nNode (" << tempNode->getNodeNo() << ") added successfully.\n";
353
354
355           //Obtain the vector of all nodes contained within the course.
356           vector<Node*> currentCourseNodes = currentCourse->getCourseNodes();
357
358           //Display a list of all the nodes that make up the course on screen.
359           cout << "\nCurrent nodes contained in Course (" << currentCourse->
                   getCourseID() << "):\n";
360
361           for (vector<Node*>::iterator it = currentCourseNodes.begin(); it !=
                   currentCourseNodes.end(); ++it) {
362               cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << ")\n";
363           }
364
365       } else {
366
367           //Otherwise if no matching node can be found, inform the user.
368           cout << "\nERROR: Node " << nodeNo << " not found.\n";
369
370       }
371
372   }
373
374   /**
375    * Attempts to locate a course from Datastore that matches
376    * the ID entered by the user.
377    * @returns The pointer to a matching course or NULL.
378    */
379   Course* Process::getSelectedCourse(void) {
380
381       char selectedID;
```

```
382
383        //Check that the ID inputted by the user is a letter.
384        while (!isalpha(selectedID)) {
385
386            //Prompt user for a course ID.
387            cout << "Please enter an existing course ID: ";
388            cin >> selectedID;
389
390            if (!isalpha(selectedID)) {
391
392                cout << "ERROR: Course ID's can contain letters only. Please try again\
                         n";
393
394            }
395
396        }
397
398        //Return the matching course fetched from the course vector in Datastore.
399        return data->getInCourse(selectedID);
400
401 }
402
403 /**
404  * Converts an inputted date string from "DD/MM/YYYY" to correct format
405  * "%d %b %Y" (e.g. 05 July 1993) required to write event details to file.
406  * @param input The original inputted event date in format "DD/MM/YYYY".
407  * @returns A string containing the same date in a modified format.
408  */
409 string Process::convertDate(string &input) {
410
411     string convertDate, result;
412
413     //Resize the conversion string to the same sized as the original.
414     convertDate.resize(input.size());
415
416     //Remove any '/' characters from the original string and pass to new string.
417     remove_copy(input.begin(), input.end(), convertDate.begin(), '/');
418
419     //Create new string stream and input modified date string into it.
420     ostringstream date1;
421     date1 << convertDate;
422
423     //Create new time structure used to process date conversion.
424     struct tm tm;
425     strptime(date1.str().c_str(), "%d%m%Y", &tm);
426
427     char date2[30];
428
429     //Re-format the date into the correct format.
430     strftime(date2, sizeof (date2), "%d %b %Y", &tm);
431
432     //Set a resulting string to the output of the re-arranged time struct.
433     result = string(date2);
434
435     //Return the re-formatted date string.
436     return result;
437
438 }
```

3.2.4 Datastore.cpp

```
1  /*
2   * File: Datastore.cpp
3   * Description: Contains and stores all the persistent data used
4   * by the application to allow data to be accessed by multiple classes.
5   * Author: Connor Luke Goddard (clg11)
6   * Date: March 2013
7   * Copyright: Aberystwyth University, Aberystwyth
8   */
9
10 #include "Datastore.h"
11
12 using namespace std;
13
14 /**
15  * Default constructor for Datastore.
16  * Sets the initial value of the 'event' pointer to NULL for
17  * error checking purposes.
18  */
19 Datastore::Datastore() {
20
21     event = NULL;
22 }
23
24 /**
25  * Destructor to be used once object is removed.
26  */
27 Datastore::~Datastore() {
28     delete event;
29 }
30
31 /**
32  * Fetches the vector of all the courses created for an event.
33  * @return A vector that contains pointers to all the Course objects created.
34  */
35 vector<Course*> Datastore::getCourseList(void){
36     return courseList;
37 }
38
39 /**
40  * Fetches the vector of all the nodes read in from "nodes.txt".
41  * @return A vector that contains pointers to all the Node objects.
42  */
43 vector<Node*> Datastore::getNodeList(void) {
44     return nodeList;
45 }
46
47 /**
48  * Fetches the vector of all the entrants created for an event.
49  * @return A vector that contains pointers to all the Entrant objects created.
50  */
51 vector<Entrant*> Datastore::getEntrantList(void){
52     return entrantList;
53 }
54
55 /**
56  * Fetches the Event object created to define the race event.
57  * @return A a pointer to the created Event object.
58  */
59 Event* Datastore::getEvent(void) const {
60     return event;
61 }
62
63 /**
64  * Adds a new Course object to the end of the 'courseList' vector.
65  * @param newCourse Pointer to the new Course object to be added to the vector.
```

```cpp
66    */
67   void Datastore::addNewCourse (Course *newCourse) {
68
69       courseList.push_back(newCourse);
70
71   }
72
73   /**
74    * Adds a new Node object to the end of the 'nodeList' vector.
75    * @param newNode Pointer to the new Node object to be added to the vector.
76    */
77   void Datastore::addNewNode (Node *newNode) {
78
79       nodeList.push_back(newNode);
80
81   }
82
83   /**
84    * Adds a new Entrant object to the end of the 'courseEntrant' vector.
85    * @param newEntrant Pointer to the new Entrant object to be added to the vector.
86    */
87   void Datastore::addNewEntrant (Entrant *newEntrant) {
88
89       entrantList.push_back(newEntrant);
90
91   }
92
93   /**
94    * Sets the 'event' pointer to a newly created Event object.
95    * @param newEvent A pointer to the new Event object created.
96    */
97   void Datastore::setNewEvent(Event *newEvent) {
98
99       this->event = newEvent;
100
101  }
102
103  /**
104   * Determines if a course with the inputted ID exists in the vector of
105   * courses ('courseList') and if so returns the pointer to that Course object.
106   * @param selectedID The course ID inputted by the user.
107   * @return Either the located course or NULL.
108   */
109  Course* Datastore::getInCourse (char selectedID) {
110
111      //Loop through the entire vector of courses.
112      for (vector<Course*>::iterator it = courseList.begin(); it != courseList.end();
              ++it) {
113
114          //If the ID of the current course matches the inputted ID...
115          if ((*it)->getCourseID() == selectedID) {
116
117              //... return the pointer to that Course object.
118              return (*it);
119          }
120      }
121
122      //Otherwise if no matches are found, return NULL.
123      return NULL;
124  }
125
126  /**
127   * Determines if a node with the inputted number exists in the vector of
128   * nodes ('nodeList') and if so returns the pointer to that Node object.
129   * @param nodeNo The node number inputted by the user.
130   * @return Either a pointer to the located Node object or NULL.
131   */
```

```
132  Node* Datastore::obtainNode (int nodeNo) {
133
134      //Loop through the entire vector of courses.
135      for (vector<Node*>::iterator it = nodeList.begin(); it != nodeList.end(); ++it)
             {
136
137          //If the number of the current node matches the inputted number...
138          if ((*it)->getNodeNo() == nodeNo) {
139
140              //... return the pointer to that Node object.
141              return (*it);
142          }
143
144      }
145
146      //Otherwise if no matches are found, return NULL.
147      return NULL;
148  }
```

3.2.5   FileIO.cpp

```cpp
/*
 * File: FileIO.cpp
 * Description: Provides file input/output and parsing facilities.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include <cstdlib>
#include <vector>
#include <fstream>
#include <iostream>
#include <sstream>  //for std::istringstream
#include <iterator> //for std::istream_iterator
#include "FileIO.h"
#include "Event.h"

using namespace std;

/**
 * Default constructor for FileIO.
 */
FileIO::FileIO() {
}

/**
 * Destructor to be used once object is removed.
 */
FileIO::~FileIO() {
}

/**
 * Writes all the created entrants for a particular event to file using a
 * specified format. Entrant information is obtained from the Entrant pointers
     vector
 * stored within the Datastore class.
 * @param entrantList Vector of all the Entrant pointers contained
 * within Datastore class.
 */
void FileIO::writeEntrants(vector<Entrant*> entrantList) {

    //Create a new file stream.
    ofstream myfile;

    /**
     * "Load" or create a new file with the given file name.
     * Flags: ios::out = Output to file, ios::app = Append to existing file
     * or create a new one.
     */
    myfile.open("../files/exampleentrants.txt", ios::out | ios::app);

    //Loop through all the created entrants.
    for (vector<Entrant*>::iterator it = entrantList.begin(); it != entrantList.end
        (); ++it) {

        //Write the entrant details to the file using specific format.
        myfile << (*it)->getEntrantNo() << " " << (*it)->getCourseID() << " " << (*
            it)->getEntrantName() << "\n";
    }

    //Close the file stream once completed.
    myfile.close();

}
```

```cpp
63  /**
64   * Writes all the created courses for a particular event to file using a
65   * specified format. Course information is obtained from the Entrant pointers
          vector
66   * stored within the Datastore class.
67   * @param entrantList Vector of all the Course pointers contained
68   * within Datastore class.
69   */
70  void FileIO::writeCourses(vector<Course*> courseList) {
71
72      ofstream myfile;
73      myfile.open("../files/examplecourses.txt", ios::out | ios::app);
74
75      //Loop through all the the courses in the vector.
76      for (vector<Course*>::iterator it = courseList.begin(); it != courseList.end();
            ++it) {
77
78          //Write the current course ID and total course size to file.
79          myfile << (*it)->getCourseID() << " " << (*it)->getCourseSize() << " ";
80
81          //Create a temporary array of current course nodes.
82          vector<Node*> currentCourseNodes = (*it)->getCourseNodes();
83
84          //Loop through all nodes that make up the current course.
85          for (vector<Node*>::iterator jt = currentCourseNodes.begin(); jt !=
                currentCourseNodes.end(); ++jt) {
86
87              //Write the node number to the file.
88              myfile << (*jt)->getNodeNo() << " ";
89          }
90
91          myfile << "\n";
92      }
93
94      myfile.close();
95
96  }
97
98  /**
99   * Writes the details of a particular event to file using a
100  * specified format. Event information is obtained from the 'event' pointer
101  * stored within the Datastore class.
102  * @param event Pointer to the stored Event class.
103  */
104 void FileIO::writeEvent(Event *event) {
105
106     ofstream myfile;
107     myfile.open("../files/exampleevent.txt", ios::out | ios::trunc);
108
109     myfile << (*event).getEventName() << "\n" << (*event).getEventDate() << "\n" <<
            (*event).getEventTime() << "\n";
110
111     myfile.close();
112
113 }
114
115 /**
116  * Accesses a specified file and returns the contents as a vector.
117  * @param fileName The file path of the specified file.
118  * @return A vector of vectors that each contain the contents of each line
119  * of the file that was read in.
120  */
121 vector<vector<string > > FileIO::getFile(string fileName) {
122
123     string line;
124
125     //Create a new file stream.
```

```
126        ifstream myfile(fileName.c_str());
127
128        //Check the file has been successfully opened.
129        if (myfile.is_open()) {
130
131            //Read the entire contents of the file.
132            while (std::getline(myfile, line)) {
133
134                //Split the current line by white space into separate tokens.
135                istringstream ss(line);
136                istream_iterator<string> begin(ss), end;
137
138                //Place all the tokens into a new vector (For the line).
139                vector<string> allStrings(begin, end);
140
141                //Add this vector to the parent vector for the whole file.
142                arrayTokens.push_back(allStrings);
143
144            }
145
146            myfile.close();
147        }
148
149        //Return the vector. If the loading was un-successful, it will be empty.
150        return arrayTokens;
151 }
```

3.2.6 Event.cpp

```
1  /*
2   * File: Event.cpp
3   * Description: Provides a data model for a particular event.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include "Event.h"
10
11 using namespace std;
12
13 /**
14  * Default constructor for Event.
15  */
16 Event::Event() {
17
18 }
19
20 /**
21  * Constructor that allows the characteristics of an event to be specified.
22  * @param newEventName The inputted name/description of the event.
23  * @param newEventDate The inputted date of the event.
24  * @param newEventTime The inputted start time of the event.
25  */
26 Event::Event(string newEventName, string newEventDate, string newEventTime) {
27
28     eventName = newEventName;
29     eventDate = newEventDate;
30     eventTime = newEventTime;
31
32 }
33
34 /**
35  * Destructor to be used once object is removed.
36  */
37 Event::~Event() {
38 }
39
40 /**
41  * Updates the start time of the event to an inputted value.
42  * @param eventTime Recently inputted start time value.
43  */
44 void Event::setEventTime(string eventTime) {
45     this->eventTime = eventTime;
46 }
47
48 /**
49  * Fetches the start time of the event.
50  * @return The value of the 'eventTime' string variable.
51  */
52 string Event::getEventTime(void) const {
53     return eventTime;
54 }
55
56 /**
57  * Updates the date of the event to an inputted value.
58  * @param eventDate Recently inputted event date value.
59  */
60 void Event::setEventDate(string eventDate) {
61     this->eventDate = eventDate;
62 }
63
64 /**
65  * Fetches the date of the event.
```

```
66    * @return The value of the 'eventDate' string variable.
67    */
68   string Event::getEventDate(void) const {
69       return eventDate;
70   }
71
72   /**
73    * Updates the name/description of the event to an inputted value.
74    * @param eventTime Recently inputted event name/description..
75    */
76   void Event::setEventName(string eventName) {
77       this->eventName = eventName;
78   }
79
80   /**
81    * Fetches the name/description of the event.
82    * @return The value of the 'eventName' string variable.
83    */
84   string Event::getEventName(void) const {
85       return eventName;
86   }
```

3.2.7   Entrant.cpp

```cpp
/*
 * File: Entrant.cpp
 * Description: Provides a data model for an entrant in an event.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include "Entrant.h"
#include <iostream>

using namespace std;


/**
 * Constructor that allows the constant 'entrant_name' and 'entrant_no' variables
 * to be specified. Also specifies the ID of the course the entrant is registered
      for.
 * @param theName The inputted name of the entrant.
 * @param theEnNo The inputted unique entrant number.
 * @param theCourseID The new course ID value to be set.
 */
Entrant::Entrant(const string &theName, const int theEnNo, char theCourseID) :
    entrantName(theName), entrantNo(theEnNo){

    courseID = theCourseID;
}

/**
 * Destructor to be used once object is removed.
 */
Entrant::~Entrant() {

}

/**
 * Fetches the name of the entrant.
 * @return A string containing the name of the entrant.
 */
string Entrant::getEntrantName(void) {
    return entrantName;
}

/**
 * Fetches the ID number of the entrant.
 * @return An integer containing the entrant number.
 */
int Entrant::getEntrantNo(void) {
    return entrantNo;
}

/**
 * Fetches the ID of the course the entrant is registered for.
 * @return An char containing the course ID.
 */
char Entrant::getCourseID(void) {
    return courseID;
}
```

3.2.8   Node.cpp

```cpp
/**
 * File: Node.cpp
 * Description: Provides the data model for a particular course node.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include <iostream>
#include "Node.h"

using namespace std;

/**
 * Constructor that allows the characteristics of a course node to be specified.
 * Constant variable 'nodeNo' is set it's value here.
 * @param newNodeNo The unique identifier of a particular node. (constant)
 * @param newNodeType The course node type to be set.
 */
Node::Node(const int newNodeNo, string newNodeType) : nodeNo(newNodeNo){

    setNodeType(newNodeType);

}

/**
 * Destructor to be used once object is removed.
 */
Node::~Node() {
}


/**
 * NOTE: setNodeNo() cannot be used due to 'nodeNo'
 * being a CONSTANT value. It therefore cannot be changed
 * once created. Making the variable MUTABLE however would allow ]
 * it to be changed.
 */

/**
 * Updates the type value of the node.
 * @param nodeType The new node type value.
 */
void Node::setNodeType(string nodeType) {
    this->nodeType = nodeType;
}

/**
 * Fetches the node type of the current node.
 * @return The type of the current node.
 */
string Node::getNodeType(void) const {
    return nodeType;
}

/**
 * Fetches the unique number of the node.
 * @return The number representing the node.
 */
const int Node::getNodeNo(void) const {
    return nodeNo;
}
```

3.2.9   Course.cpp

```
 1  /*
 2   * File: Course.cpp
 3   * Description: Provides a data model for an event course.
 4   * Author: Connor Luke Goddard (clg11)
 5   * Date: March 2013
 6   * Copyright: Aberystwyth University, Aberystwyth
 7   */
 8
 9  #include "Course.h"
10
11  using namespace std;
12
13  /**
14   * Constructor that allows the constant 'courseID' variable
15   * to be specified. Also defaults the size of a course to 0.
16   * @param theCourseID The new course ID value to be set.
17   */
18  Course::Course(const char theCourseID) : courseID(theCourseID) {
19      courseSize = 0;
20  }
21
22  /**
23   * Destructor to be used once object is removed.
24   */
25  Course::~Course() {
26
27  }
28
29  /**
30   * Adds a new node to the end of the 'courseNode' vector.
31   * @param newNode Pointer to the new node to be added to the vector.
32   */
33  void Course::addCourseNode(Node *newNode) {
34
35      this->courseNodes.push_back(newNode);
36      this->setCourseSize(courseNodes.size());
37
38  }
39
40  /**
41   * Updates the total size of the course. (i.e. vector size).
42   * @param courseSize The new size value.
43   */
44  void Course::setCourseSize(int courseSize) {
45      this->courseSize = courseSize;
46  }
47
48  /**
49   * Fetches the value of 'courseSize'.
50   * @return The total number of nodes in the course.
51   */
52  int Course::getCourseSize(void) const {
53      return courseSize;
54  }
55
56  /**
57   * Fetches a vector of all the nodes in the course.
58   * @return A vector of nodes that make up the course.
59   */
60  std::vector<Node*> Course::getCourseNodes(void) const {
61      return courseNodes;
62  }
63
64  /**
65   * Fetches the ID of the course.
```

```
66    * @return The ID of the course.
67    */
68   const char Course::getCourseID(void) const {
69       return courseID;
70   }
```

4 Event Creator - Build/Compilation Log

The listing below contains the build/compilation log for the "event creator" application. Extra warning flags have been used with the C++ compiler (**g++**) to ensure that no errors/warnings occur when compiling the application.

Listing 1: Compilation log built within Netbeans IDE 7.3 on Ubuntu 12.04

```
 1  "/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-conf
 2  make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
 3  rm -f -r build/Debug
 4  rm -f dist/Debug/GNU-Linux-x86/event-creator
 5  make[1]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
 6
 7
 8  CLEAN SUCCESSFUL (total time: 57ms)
 9
10  "/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
11  make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
12  "/usr/bin/make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/event-creator
13  make[2]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
14  mkdir -p build/Debug/GNU-Linux-x86
15  rm -f build/Debug/GNU-Linux-x86/Course.o.d
16  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Course.o.d -o build/Debug/GNU-Linux-
        x86/Course.o Course.cpp
17  mkdir -p build/Debug/GNU-Linux-x86
18  rm -f build/Debug/GNU-Linux-x86/Datastore.o.d
19  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Datastore.o.d -o build/Debug/GNU-
        Linux-x86/Datastore.o Datastore.cpp
20  mkdir -p build/Debug/GNU-Linux-x86
21  rm -f build/Debug/GNU-Linux-x86/Entrant.o.d
22  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Entrant.o.d -o build/Debug/GNU-Linux-
        x86/Entrant.o Entrant.cpp
23  mkdir -p build/Debug/GNU-Linux-x86
24  rm -f build/Debug/GNU-Linux-x86/Event.o.d
25  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Event.o.d -o build/Debug/GNU-Linux-
        x86/Event.o Event.cpp
26  mkdir -p build/Debug/GNU-Linux-x86
27  rm -f build/Debug/GNU-Linux-x86/FileIO.o.d
28  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/FileIO.o.d -o build/Debug/GNU-Linux-
        x86/FileIO.o FileIO.cpp
29  mkdir -p build/Debug/GNU-Linux-x86
30  rm -f build/Debug/GNU-Linux-x86/Menu.o.d
31  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Menu.o.d -o build/Debug/GNU-Linux-x86
        /Menu.o Menu.cpp
32  mkdir -p build/Debug/GNU-Linux-x86
33  rm -f build/Debug/GNU-Linux-x86/Node.o.d
34  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Node.o.d -o build/Debug/GNU-Linux-x86
        /Node.o Node.cpp
35  mkdir -p build/Debug/GNU-Linux-x86
36  rm -f build/Debug/GNU-Linux-x86/Process.o.d
37  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/Process.o.d -o build/Debug/GNU-Linux-
        x86/Process.o Process.cpp
38  mkdir -p build/Debug/GNU-Linux-x86
39  rm -f build/Debug/GNU-Linux-x86/main.o.d
40  g++     -c -g -Wall -MMD -MP -MF build/Debug/GNU-Linux-x86/main.o.d -o build/Debug/GNU-Linux-x86
        /main.o main.cpp
41  mkdir -p dist/Debug/GNU-Linux-x86
42  g++      -o dist/Debug/GNU-Linux-x86/event-creator build/Debug/GNU-Linux-x86/Course.o build/
        Debug/GNU-Linux-x86/Datastore.o build/Debug/GNU-Linux-x86/Entrant.o build/Debug/GNU-Linux-
        x86/Event.o build/Debug/GNU-Linux-x86/FileIO.o build/Debug/GNU-Linux-x86/Menu.o build/
        Debug/GNU-Linux-x86/Node.o build/Debug/GNU-Linux-x86/Process.o build/Debug/GNU-Linux-x86/
        main.o
43  make[2]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
44  make[1]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
45
46
47  BUILD SUCCESSFUL (total time: 3s)
```

5   Event Creator - Example Usage

This section demonstrates the "event creator" application running using test input data to ensure that expected functionality and suitable error checking is taking place correctly.

Listing 2: Example output of functionality testing of the event creator application.

```
Welcome. Please enter the file path for course nodes:
../files/idontknow.txt
ERROR: Nodes file (../files/idontknow.txt) could not be located.

Please check the file path and try again. Exiting...

RUN FINISHED; exit value 1; real time: 11s; user: 0ms; system: 0ms


Welcome. Please enter the file path for course nodes:
../files/nodes.txt
Course nodes loaded successfully.
Loading program...


***********************************
Welcome to the Event Creator.
Please select an option:
-----------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
***********************************
1


***********************************
Event Editor | Please make a choice:
-----------------------------------
1. Create new event.
2. Write event to file.
3. Return to main menu.
***********************************
2
Exporting event to file.

ERROR: No event created. Nothing to export.

***********************************
Event Editor | Please make a choice:
-----------------------------------
1. Create new event.
2. Write event to file.
3. Return to main menu.
***********************************
1
Please enter an event name/description: the test running event
Please enter the date of the event: (DD/MM/YYYY) 15/06/2004
Please enter the time of the event: 18:00

Event (the test running event) created successfully.

***********************************
Event Editor | Please make a choice:
-----------------------------------
1. Create new event.
2. Write event to file.
3. Return to main menu.
***********************************
1
WARNING: An event has already been created.
Do you wish to create a new event? (Y/N)
Y
Please enter an event name/description: the test horse event
Please enter the date of the event: (DD/MM/YYYY) 08/07/2013
Please enter the time of the event: 09:45

Event (the test horse event) created successfully.

***********************************
Event Editor | Please make a choice:
```

```
------------------------------------
1. Create new event.
2. Write event to file.
3. Return to main menu.
************************************
2
Exporting event to file.


************************************
Event Editor | Please make a choice:
------------------------------------
1. Create new event.
2. Write event to file.
3. Return to main menu.
************************************
3
Returning to main menu...


************************************
Welcome to the Event Creator.
Please select an option:
------------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
************************************
4
Writing all data to files...
ERROR: No entrants created. Nothing to export.


************************************
Welcome to the Event Creator.
Please select an option:
------------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
************************************
3


************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
2
Please enter an existing course ID: U
ERROR: Course does not exist. Please try again
************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
1
Please enter a new course ID: U

Course (U) created successfully.


************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
```

```
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
45

ERROR: Node 45 not found.

**********************************
Course Editor | Please make a choice:
-----------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
**********************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
1

Node (1) added successfully.

Current nodes contained in Course (U):
1 (CP)

**********************************
Course Editor | Please make a choice:
-----------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
**********************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
3

Node (3) added successfully.

Current nodes contained in Course (U):
1 (CP)
3 (JN)

**********************************
Course Editor | Please make a choice:
-----------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
**********************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
11

Node (11) added successfully.

Current nodes contained in Course (U):
1 (CP)
3 (JN)
11 (JN)

**********************************
Course Editor | Please make a choice:
-----------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
**********************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
```

```
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
18

Node (18) added successfully.

Current nodes contained in Course (U):
1 (CP)
3 (JN)
11 (JN)
18 (JN)

************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
1
Please enter a new course ID: 6
ERROR: Course ID's can contain letters only. Please try again
Please enter a new course ID: C

Course (C) created successfully.

************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
2
Please enter an existing course ID: C
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
5

Node (5) added successfully.

Current nodes contained in Course (C):
5 (CP)

************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
11

Node (11) added successfully.

Current nodes contained in Course (U):
1 (CP)
3 (JN)
11 (JN)
18 (JN)
11 (JN)

************************************
Course Editor | Please make a choice:
------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
2
Please enter an existing course ID: C
```

```
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
7

Node (7) added successfully.

Current nodes contained in Course (C):
5 (CP)
7 (CP)

************************************
Course Editor | Please make a choice:
-------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
************************************
4
Returning to main menu...

************************************
Welcome to the Event Creator.
Please select an option:
-------------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
************************************
2

************************************
Entrant Editor | Please make a choice:
-------------------------------------
1. Create a new entrant.
2. Export entrants to file.
3. Return to main menu.
************************************
1
Please enter a name: cONNOR Goddard
Do you wish to set a manual entrant no? (Y/N)N
Setting automatic entrant number
Please enter a course ID: 4
ERROR: Course ID's can contain letters only. Please try again
Please enter a course ID: U

Entrant(1) created successfully.

************************************
Entrant Editor | Please make a choice:
-------------------------------------
1. Create a new entrant.
2. Export entrants to file.
3. Return to main menu.
************************************
1
Please enter a name: David Ash
Do you wish to set a manual entrant no? (Y/N)Y
Please enter an entrant no: 1

ERROR: This entrant already exists. Please enter another value.
Please enter an entrant no: 13
Please enter a course ID: C

Entrant(13) created successfully.

************************************
Entrant Editor | Please make a choice:
-------------------------------------
1. Create a new entrant.
2. Export entrants to file.
3. Return to main menu.
************************************
1
Please enter a name: Charlie Sheen
Do you wish to set a manual entrant no? (Y/N)N
Setting automatic entrant number
```

```
Please enter a course ID: U

Entrant(3) created successfully.

************************************
Entrant Editor | Please make a choice:
------------------------------------
1. Create a new entrant.
2. Export entrants to file.
3. Return to main menu.
************************************
3
Returning to main menu...

************************************
Welcome to the Event Creator.
Please select an option:
------------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
************************************
4
Writing all data to files...

************************************
Welcome to the Event Creator.
Please select an option:
------------------------------------
1. Event Editor
2. Entrant Editor
3. Course Editor
4. Export ALL files.
5. Exit Program.
************************************
5
Exiting...

RUN FINISHED; exit value 0; real time: 4m 28s; user: 0ms; system: 0ms
```

6   Event Creator - File Output

This section lists the contents of the three external files that the event creator application has produced from the user input provided from the previous test run.

Listing 3: Output of 'event.txt' file produced by the "event creator" application.

```
the test horse event
08 Jul 2013
09:45
```

Listing 4: Output of 'courses.txt' file produced by the "event creator" application.

```
U 5 1 3 11 18 11
C 2 5 7
```

Listing 5: Output of 'entrants.txt' file produced by the "event creator" application.

```
1 U cONNOR Goddard
13 C David Ash
3 U Charlie Sheen
```

7 Checkpoint Manager (Java) - Source Code

This section contains the complete source code for the "checkpoint manager" program written in Java (JVM 7).

7.1 'Driver' Package

7.1.1 CMDriver.java

```java
package aber.dcs.cs22510.clg11.driver;

import aber.dcs.cs22510.clg11.util.LoadData;
import aber.dcs.cs22510.clg11.gui.GUIFrame;
import aber.dcs.cs22510.clg11.model.Datastore;
import aber.dcs.cs22510.clg11.model.Datatype;
import aber.dcs.cs22510.clg11.util.FileIO;

/**
 * Bootstrap class - Initialises the application.
 *
 * @author Connor Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class CMDriver {

    /**
     * The main method used to initialise the main application.
     *
     * @param args The file names for the data files.
     */
    public static void main(String[] args) {

        //Instantiate new Datastore object that will be shared by other classes.
        Datastore data = new Datastore();

        //Instantiate new Datastore object that will be shared by other classes.
        LoadData load = new LoadData(data);


        //Load input files into Datastore class (nodes, tracks and courses).
        try {

            load.loadFiles(Datatype.NODE, args[0]);
            load.loadFiles(Datatype.COURSE, args[1]);
            load.loadFiles(Datatype.ENTRANT, args[2]);

            //Once loading via textual interface is complete, display GUI.
            new GUIFrame(data);

        } catch (IndexOutOfBoundsException eX) {

            System.out.println("ERROR: File parameters missing.");
            System.out.println("Parameter format = <node path> <courses path> <
                entrants path>");
        }

    }
}
```

7.2 'Util' Package

7.2.1 ProcessData.java

```java
package aber.dcs.cs22510.clg11.util;

import aber.dcs.cs22510.clg11.model.Course;
import aber.dcs.cs22510.clg11.model.Datastore;
import aber.dcs.cs22510.clg11.model.Entrant;
import aber.dcs.cs22510.clg11.model.Node;
import java.io.File;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Responsible for updating the internal record of entrant progress (based on
 * data read in from "times.txt") and for processing new time logs submitted by
 * a user. Has access to the shared
 * {@link aber.dcs.cs22510.clg11.model.Datastore} class to allow processing and
 * manipulation of the data collections.
 *
 * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class ProcessData {

    private Datastore data;
    private FileIO fileIO = new FileIO();
    private String lastLoggedTime = null;


    /**
     * Constructor to instantiate a new ProcessData. Takes the shared data store
     * object created in {@link aber.dcs.cs22510.clg11.driver.CMDriver} as a
     * parameter to allow accessed to the lists of nodes/entrants/courses loaded
     * in.
     *
     * @param newData Datastore object created in CMDriver.
     */
    public ProcessData(Datastore newData) {

        this.data = newData;

    }

    /**
     * Returns the time value of the last read-in time log.
     *
     * @return The last time value of the "times.txt" file.
     */
    public String getLastLoggedTime() {
        return lastLoggedTime;
    }

    /**
     * Attempts to fetch a specified entrant from the internal collection of
     * entrants.
     *
     * @param requiredEntrant The number of the required entrant.
     * @return The specified Entrant object, or NULL.
     */
    public Entrant obtainEntrant(int requiredEntrant) {

```

```java
 64            for (Entrant e : data.getEntrants()) {
 65
 66                if (e.getNumber() == requiredEntrant) {
 67
 68                    return e;
 69                }
 70
 71            }
 72
 73            return null;
 74        }
 75
 76        /**
 77         * Attempts to fetch the collection of course nodes that make up the course
 78         * that a specified entrant is registered for.
 79         *
 80         * @param selectedEntrant The number of the required entrant.
 81         * @return The collection of course nodes, or NULL.
 82         */
 83        public ArrayList<Node> obtainEntrantCourseNodes(Entrant selectedEntrant) {
 84
 85            //Loop through all the stored courses.
 86            for (Course c : data.getCourses()) {
 87
 88                //If the current course matches the entrant's course.
 89                if (c.getCourseID() == selectedEntrant.getCourseID()) {
 90
 91                    //Return the collection of nodes for that course.
 92                    return c.getCourseNodes();
 93                }
 94
 95            }
 96
 97            //Otherwise if nothing is found, return NULL.
 98            return null;
 99        }
100
101        /**
102         * Processes each line read in from the "times.txt" file to update the
103         * internal record of entrant's progress. This method is crucial to ensure
104         * that any time log updates created by any other running versions of the
105         * checkpoint manager are recorded in the internal entrant record within
106         * this application.
107         *
108         * @param timeDelimiter The character symbol used to represent the status of
109         * the particular time log.
110         * @param nodeNo The number of the node the time log was recorded at.
111         * @param entrantNo The number of the entrant that was recorded.
112         * @throws IndexOutOfBoundsException
113         */
114        public void processNewTime(String timeDelimiter, int nodeNo, int entrantNo)
                throws IndexOutOfBoundsException {
115
116            //Boolean used to determine whether this particular time log has been
                    processed.
117            boolean isUpdated = false;
118
119            //Obtain the required entrant from the internal collection of entrants.
120            Entrant currentEntrant = obtainEntrant(entrantNo);
121
122            //Check if the time log dictates that the entrant should be excluded.
123            if (timeDelimiter.equals("I") || timeDelimiter.equals("E")) {
124
125                //If so exclude the entrant.
126                excludeEntrant(entrantNo);
127
128                //Log this activity in the log file ("log.txt");
```

```java
129                fileIO.addActivityLog("Entrant no: " + entrantNo + " successfully
                       excluded.");
130
131            /*
132             * Otherwise if they shouldn't be excluded,
133             * check to see if the entrant has already been excluded.
134             */
135        } else if (!currentEntrant.getIsExcluded()) {
136
137            ArrayList<Node> courseNodes = obtainEntrantCourseNodes(currentEntrant);
138
139            //Loop through all the nodes that make up the course the entrant is on.
140            for (int i = 0; i < courseNodes.size(); i++) {
141
142                /*
143                 * Check that the current progress of the entrant < the index of
144                 * the current node in the array (to prevent nodes the entrant has
145                 * already passed being used again), and the current node equals
146                 * the node number of the current time log.
147                 */
148                if (i > (currentEntrant.getCurrentProgress() - 1) && courseNodes.
                       get(i).getNumber() == nodeNo && !isUpdated) {
149
150                    /*
151                     * If the entrant has ARRIVED at a medical checkpoint,
152                     * their progress should not be incremented as they are
153                     * now waiting at the MC
154                     */
155                    if (timeDelimiter.equals("A")) {
156
157                        //Just prevent this particular time log being processed any
                               further.
158                        currentEntrant.setAtMC(true);
159                        isUpdated = true;
160
161                        /*
162                         * Otherwise, if they are DEPARTING from a MC or they
163                         * have just arrived at a normal checkpoint, then their
164                         * progress needs to be recorded and incremented.
165                         */
166                    } else {
167
168                        /*
169                         * If the read in node from time file is further along
170                         * the course than the current progress,
171                         * update the current progress.
172                         */
173                        currentEntrant.setCurrentProgress((i + 1));
174                        currentEntrant.setAtMC(false);
175
176
177                        /*
178                         * Check to see if the entrant has now completed
179                         * their course.
180                         */
181                        if (currentEntrant.getCurrentProgress() >= courseNodes.size
                               ()) {
182
183                            //Log that they have finished.
184                            currentEntrant.setIsFinished(true);
185
186                            //Log this activity in the log file ("log.txt");
187                            fileIO.addActivityLog("Entrant no: " + currentEntrant.
                                   getNumber() + " has sucessfully finished the course
                                   .");
188                        }
189
```

```
190                         isUpdated = true;
191
192                     }
193                 }
194             }
195         }
196     }
197
198     /**
199      * Updates a particular Entrant object to log the fact that they have been
200      * excluded from their race.
201      *
202      * @param entrantNo The number of the required entrant.
203      */
204     public void excludeEntrant(int entrantNo) {
205
206         for (Entrant e : data.getEntrants()) {
207
208             if (e.getNumber() == entrantNo) {
209
210                 e.setIsExcluded(true);
211             }
212
213         }
214
215     }
216
217     /**
218      * Processes a new time log submitted by the user by determining whether the
219      * entrant is on the correct path or not and updates the "times.txt" file
220      * with the resulting time log.
221      *
222      * @param courseNodes The collection of nodes that make up the course the
223      * current entrant is registered for.
224      * @param selectedEntrant The current entrant being processed.
225      * @param newNode The newly submitted node that the entrant has arrived at.
226      * @param time The inputted time of the entrant's arrival at the CP.
227      * @return
228      */
229     public String processTimeLog(ArrayList<Node> courseNodes, Entrant
230         selectedEntrant, int newNode, String time) {
231         //Obtain the current progress of the entrant (i.e. the index of the array).
232         int nextNodeIndex = selectedEntrant.getCurrentProgress();
233         String result = null;
234         boolean timesNotLocked = true;
235         boolean logNotLocked = true;
236
237         //Check that the entrant has not already finished, or been excluded.
238         if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
239
240             result = " Entrant " + selectedEntrant.getNumber() + " successfully
241                 completed their course.";
242
243         } else if (selectedEntrant.getIsExcluded()) {
244
245             result = " Entrant " + selectedEntrant.getNumber() + " has been
246                 excluded from their course.";
247
248         } else {
249
250             /*
251              * Check whether the next node in the array (i.e. the next node that
252                  the
253              * entrant SHOULD have reached) is actually the node submitted.
254              */
255             if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
```

```
253
254                          /*
255                           * If they do not match, the entrant has gone the wrong way.
256                           * Append this new time log with the 'I' time delimiter to the
257                           * times file ("times.txt").
258                           */
259                          timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "
                                 I " + newNode + " " + selectedEntrant.getNumber() + " " + time
                                 + "\n");
260
261                          logNotLocked = fileIO.addActivityLog("Submitted checkpoint " +
                                 newNode + " incorrect for course. (Entrant No: " +
                                 selectedEntrant.getNumber() + ")");
262
263                          result = "Entrant " + selectedEntrant.getNumber()
264                                      + " has gone the INCORRECT way. (Expected node: " +
                                         courseNodes.get(nextNodeIndex).getNumber() + ")";
265
266                  } else {
267
268                          /*
269                           * Otherwise if they do  match, the entrant has gone the right way.
270                           * Append this new time log with the 'T' time delimter to the
271                           * times file ("times.txt").
272                           */
273                          timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "
                                 T " + newNode + " " + selectedEntrant.getNumber() + " " + time
                                 + "\n");
274
275                          logNotLocked = fileIO.addActivityLog("Submitted checkpoint " +
                                 newNode + " incorrect for course. (Entrant No: " +
                                 selectedEntrant.getNumber() + ")");
276
277                          result = "Entrant " + selectedEntrant.getNumber()
278                                      + " has gone the CORRECT way. (Expected node: " +
                                         courseNodes.get(nextNodeIndex).getNumber() + ")";
279                  }
280
281          }
282
283          /*
284           * If any of the output files are locked by another process/application,
285           * inform the user.
286           */
287          if (!logNotLocked) {
288
289              result = " ERROR: System log file locked - Cannot write to file.";
290
291          }
292
293          if (!timesNotLocked) {
294
295              result = " ERROR: Times log file locked - Cannot write to file. Please
                     try again.";
296
297          }
298
299          if (!timesNotLocked && !logNotLocked) {
300
301              result = " ERROR: Cannot write to time log or log file. - Both files
                     locked.";
302          }
303
304          return result;
305
306      }
307
```

```
308        /**
309         * Processes a new time log submitted by the user by determining whether the
310         * entrant is on the correct path or not and updates the "times.txt" file
311         * with the resulting time log (Overloaded method for processing medical
312         * checkpoints).
313         *
314         * @param courseNodes The collection of nodes that make up the course the
315         * current entrant is registered for.
316         * @param selectedEntrant The current entrant being processed.
317         * @param newNode The newly submitted node that the entrant has arrived at.
318         * @param mcType Whether the entrant was arriving or departing from the MC.
319         * @param time The inputted time of the entrant's arrival at the CP.
320         * @param isExcluded
321         * @return String containing the result of processing the time log.
322         */
323        public String processTimeLog(ArrayList<Node> courseNodes, Entrant
               selectedEntrant, int newNode, String mcType, String time, boolean
               isExcluded) {
324
325            int nextNodeIndex = selectedEntrant.getCurrentProgress();
326            String result = null;
327            boolean timesNotLocked = true;
328            boolean logNotLocked = true;
329
330            //Check that the entrant has not already finished, or been excluded.
331            if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
332
333                result = " Entrant " + selectedEntrant.getNumber() + " successfully
                   completed their course.";
334
335            } else if (selectedEntrant.getIsExcluded()) {
336
337                result = " Entrant " + selectedEntrant.getNumber() + " has been
                   excluded from their course.";
338
339            } else {
340
341                if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
342
343                    logNotLocked = fileIO.addActivityLog("Submitted checkpoint
                       incorrect for course. (Entrant No: " + selectedEntrant.
                       getNumber() + ")");
344
345                    timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "
                       I " + newNode + " " + selectedEntrant.getNumber() + " " + time
                       + "\n");
346
347                    result = "Entrant " + selectedEntrant.getNumber()
348                            + " has gone the wrong way. (Expected node: " + courseNodes
                               .get(nextNodeIndex).getNumber() + ")";
349
350                } else if (isExcluded) {
351
352                    logNotLocked = fileIO.addActivityLog("Entrant excluded for medical
                       reasons. (Entrant No: " + selectedEntrant.getNumber() + ")");
353
354                    timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "
                       E " + newNode + " " + selectedEntrant.getNumber() + " " + time
                       + "\n");
355
356                    result = "Entrant " + selectedEntrant.getNumber()
357                            + " has been excluded for medical reasons.";
358
359                } else {
360
361                    /*
362                     * If they do match, the entrant has gone the right way.
```

```
363                              * Determine whether the entrant was arriving at, or departing
364                              * from the MC and update the time log file ("times.txt")
365                              * accordingly.
366                              */
367                          if (mcType.equals("Arriving")) {
368
369                              logNotLocked = fileIO.addActivityLog("New MC arrival time
                                     submitted. (Entrant No: " + selectedEntrant.getNumber() + "
                                     )");
370
371                              timesNotLocked = fileIO.writeFile(new File("../files/times.txt"
                                     ), "A " + newNode + " " + selectedEntrant.getNumber() + " "
                                     + time + "\n");
372
373                              result = "Entrant " + selectedEntrant.getNumber()
374                                      + " has successfully arrived at MC " + courseNodes.get(
                                         nextNodeIndex).getNumber() + ".";
375
376                          } else {
377
378                              logNotLocked = fileIO.addActivityLog("New MC departure time
                                     submitted. (Entrant No: " + selectedEntrant.getNumber() + "
                                     )");
379                              timesNotLocked = fileIO.writeFile(new File("../files/times.txt"
                                     ), "D " + newNode + " " + selectedEntrant.getNumber() + " "
                                     + time + "\n");
380
381                              result = "Entrant " + selectedEntrant.getNumber()
382                                      + " has successfully departed from MC " + courseNodes.
                                         get(nextNodeIndex).getNumber() + ".";
383                          }
384                      }
385
386                  }
387
388              /*
389               * If any of the output files are locked by another process/application,
390               * inform the user.
391               */
392              if (!timesNotLocked) {
393
394                  result = " ERROR: Times log file locked - Cannot write to file. Please
                         try again.";
395
396              }
397
398              if (!logNotLocked) {
399
400                  result = " ERROR: System log file locked - Cannot write to file. Please
                         try again.";
401
402              }
403
404              if (!timesNotLocked && !logNotLocked) {
405
406                  result = " ERROR: Cannot write to time log file or system log file. -
                         Both files locked.";
407
408              }
409
410              return result;
411
412          }
413
414          /**
415           * Obtains all the times from the time log file ("times.txt") before
416           * processing each time log.
```

```java
417          * @return A boolean determining if the file was successfully loaded or not.
418          */
419         public boolean getTimes() {
420
421             //Obtain a collection of ALL the time logs read in from the "times.txt"
                     file.
422             File timesFile = new File("../files/times.txt");
423
424             if (timesFile.exists()) {
425
426                 ArrayList<String[]> times = fileIO.readIn(timesFile, true);
427
428                 //For every time log read in from the file...
429                 for (String[] newTime : times) {
430
431                     //... process this time log and update the internal record of
                             entrants.
432                     processNewTime(newTime[0], Integer.parseInt(newTime[1]), Integer.
                             parseInt(newTime[2]));
433
434                     this.lastLoggedTime = newTime[3];
435
436                 }
437
438                 //Log this activity in the log file ("log.txt");
439                 fileIO.addActivityLog("Time logs file loaded successfully (times.txt)")
                         ;
440
441                 return true;
442
443             }
444
445             return false;
446         }
447
448         /**
449          * Compares the time of the last read-in time log, with the new time being
450          * submitted to check that the user is not entering a time in the past.
451          *
452          * @param oldTimeString The last time value read-in from "times.txt".
453          * @param newTimeString The new time value being submitted by the user.
454          * @return A boolean determining if the new time value is in the past.
455          */
456         public boolean compareTimes(String oldTimeString, String newTimeString) {
457
458             SimpleDateFormat df = new SimpleDateFormat("HH:mm");
459
460             Date lastRecordedTime;
461             Date newTime;
462
463             try {
464
465                 //Create new Date objects using the last logged, and new time values.
466                 lastRecordedTime = df.parse(oldTimeString);
467                 newTime = df.parse(newTimeString);
468
469                 //Check if the new time entered is before the last read-in time.
470                 if (df.format(lastRecordedTime).compareTo(df.format(newTime)) > 0) {
471
472                     //If so, then this cannot be allowed.
473
474                     //Log this activity in the log file ("log.txt");
475                     fileIO.addActivityLog("User attempted to enter new time value in
                             the past. (New time: " + df.format(newTime) + ")");
476
477                     return true;
478                 }
```

```
479
480            } catch (ParseException ex) {
481                Logger.getLogger(ProcessData.class.getName()).log(Level.SEVERE, null,
                        ex);
482            }
483
484            return false;
485        }
486  }
```

7.2.2 FileIO.java

```java
package aber.dcs.cs22510.clg11.util;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.channels.FileLock;
import java.nio.channels.OverlappingFileLockException;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;

/**
 * Provides file I/O facilities to allow data files to be read into the system,
 * and the time file to be updated/appended to as required.
 *
 * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class FileIO {

    /**
     * The last read-in line number from "times.txt".
     */
    private int timesFilePosition = 0;

    /**
     * Default constructor for FileIO.
     */
    public FileIO() {

    }

    /**
     * Reads in the contents of specified data files and places the contents
     * into an Arraylist which is then returned, and used to update the internal
     * data collections used by the application.
     *
     * @param fileName The directory of the file to be parsed.
     * @param isTimesFile
     * @return Arraylist of String arrays containing the contents of the parsed
     * file.
     */
    public ArrayList<String[]> readIn(File fileName, boolean isTimesFile) {

        ArrayList<String[]> values = new ArrayList<>();

        try {

            //Create File IO objects
            FileReader fileReader;
            BufferedReader bufferedReader;

            //Initialise the File IO objects, passing in the selected file path
            fileReader = new FileReader(fileName);
            bufferedReader = new BufferedReader(fileReader);


            /*
             * Check if the current file being read in is the times file,
             * and if so whether or not the file has been read-in previously.
             */
```

```java
66                if (isTimesFile && this.timesFilePosition > 0) {
67
68                    /*
69                     * Read down to the last logged line read-in file
70                     * without processing any of the lines (used to "skip" down
71                     * to any lines that could have been added after the last time
72                     * the file was read in by this application).
73                     */
74                    for (int i = 0; i < this.timesFilePosition; i++) {
75                        bufferedReader.readLine();
76                    }
77                }
78
79                //Initialise local variable used to store the current line being read
                        in
80                String line;
81
82                //While there are still lines to read in from the file (i.e. read in
                        every line in the file)
83                while ((line = bufferedReader.readLine()) != null) {
84
85                    //As there is multiple data on each line, split the values up.
86                    String[] details = line.split(" ");
87
88                    //Add these broken down values to the larger collection of lines.
89                    values.add(details);
90
91                    /*
92                     * If the current file being read in is "times.txt", updated the
93                     * last line to be read in by the system. (Used for when the
94                     * file is re-"readin" by the system).
95                     */
96                    if (isTimesFile) {
97                        timesFilePosition++;
98                    }
99                }
100
101                //Once completed, safely close the file reader
102                bufferedReader.close();
103
104                return values;
105
106                //If any IO exceptions occur...
107            } catch (IOException iOE) {
108
109                return null;
110            }
111
112    }
113
114    /**
115     * Writes output data to specified files, as these files are shared, file
116     * locking has to be used to prevent corruption of data/files.
117     *
118     * @param writeFile The file that is to be written to.
119     * @param output The output data string.
120     * @return A boolean determining if the file was successfully written to.
121     */
122    public boolean writeFile(File fileName, String output) {
123
124        FileOutputStream fos;
125        FileLock fl = null;
126
127        try {
128
129            //If the file does not exist, create a new file.
130            if (!fileName.exists()) {
```

```java
131                    fileName.createNewFile();
132                }
133
134                //Create a new output stream that will append to the file.
135                fos = new FileOutputStream(fileName.getAbsoluteFile(), true);
136
137                //Attempt to lock the file to allow the data to be written.
138                try {
139
140                    fl = fos.getChannel().tryLock();
141
142                } catch (OverlappingFileLockException flE) {
143
144                    /*
145                     * If there is already a process within the same JVM locking
146                     * the file, inform the user.
147                     */
148                    System.out.println("ERROR: File <" + fileName.getName() + "> cannot
                        be accessed. File lock still in place.");
149                }
150
151                //Check if the lock was successfull.
152                if (fl != null) {
153
154                    try (FileWriter fw = new FileWriter(fos.getFD())) {
155
156                        fw.write(output);
157
158                        //Once the data has been successfully written, release the lock
                            .
159                        fl.release();
160                    }
161
162                    return true;
163
164                }
165
166        } catch (IOException e) {
167
168        }
169
170        return false;
171    }
172
173    /**
174     * Adds a new log message to the "logs.txt" file. Called when a major
175     * activity occurs in the application.
176     *
177     * @param logMessage Message describing the activity.
178     * @return Boolean determining if the log was successfully written to file.
179     */
180    public boolean addActivityLog(String logMessage) {
181
182        //Obtain the current date/time and format it for use in the log file.
183        DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
184        Calendar cal = Calendar.getInstance();
185
186        //Build the log message using predefined output template.
187        String logOutput = "LOG - CM: " + logMessage + " - " + dateFormat.format(
                cal.getTime()) + "\n";
188
189        //Write the log message to the log file.
190        return writeFile(new File("../files/log.txt"), logOutput);
191
192    }
193 }
```

7.2.3   LoadData.java

```java
package aber.dcs.cs22510.clg11.util;

import aber.dcs.cs22510.clg11.model.*;
import java.io.File;
import java.util.ArrayList;

/**
 * Responsible for loading crucial, preliminary data files into the system using
 * a textual interface before the GUI is loaded.
 *
 * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class LoadData {

    private Datastore data;
    private FileIO fileIO = new FileIO();

    /**
     * Constructor to instantiate a new LoadData. Takes the shared data store
     * object created in {@link aber.dcs.cs22510.clg11.driver.CMDriver} as a
     * parameter to allow accessed to the lists of nodes/entrants/courses loaded
     * in.
     *
     * @param comp Shared Datastore object created within CMDriver.
     */
    public LoadData(Datastore comp) {

        this.data = comp;

    }

    /**
     * Prompts user for the file path of a specified file before attempting to
     * load the data into it's respective data collection.
     *
     * @param type ENUM denoting the type of data file (Node, Course or
     * Entrant).
     * @param fileName The path of the file to be loaded.
     */
    public void loadFiles(Datatype type, String fileName) {

        File f = new File(fileName);
        ArrayList<String[]> readValues;

        //Check if the file exists.
        if (!f.exists()) {

            //If it does not exist, inform the user.
            if (type == Datatype.NODE) {

                System.out.println("ERROR: Nodes file <" + fileName + "> does not
                    exist.");

            } else if (type == Datatype.COURSE) {

                System.out.println("ERROR: Courses file <" + fileName + "> does not
                    exist.");

            } else {

                System.out.println("ERROR: Entrants file <" + fileName + "> does
                    not exist.");

            }
```

```
63
64              System.out.println("Parameter format = <node path> <courses path> <
                    entrants path>");
65              System.exit(0);
66          }
67
68
69          //If the file does exist, read in the data from the file.
70          readValues = fileIO.readIn(f, false);
71
72          //Determine the type of data being loaded.
73          if (type.equals(Datatype.NODE)) {
74
75              for (String[] newItem : readValues) {
76
77                  //Load all the nodes from the read-in data.
78                  loadNodes(newItem);
79
80              }
81
82              System.out.println("Nodes file loaded successfully (nodes.txt)");
83
84              //Log this activity in the log file ("log.txt");
85              fileIO.addActivityLog("Nodes file loaded successfully (nodes.txt)");
86
87          } else if (type.equals(Datatype.COURSE)) {
88
89              for (String[] newItem : readValues) {
90
91                  loadCourses(newItem);
92
93              }
94
95              System.out.println("Courses file loaded successfully (courses.txt)");
96              fileIO.addActivityLog("Courses file loaded successfully (courses.txt)")
                    ;
97
98          } else {
99
100             for (String[] newItem : readValues) {
101
102                 loadEntrants(newItem);
103
104             }
105
106             System.out.println("Entrants file loaded successfully (entrants.txt)");
107             fileIO.addActivityLog("Entrants file loaded successfully (entrants.txt)
                    ");
108         }
109
110     }
111
112     /**
113      * Parses the data read-in from the "courses.txt" file and creates a new
114      * {@link aber.dcs.cs22510.clg11.model.Course} object populated with the
115      * read-in characteristics. This new Course object is then added to the
116      * internal collection of Courses.
117      *
118      * @param courseData Collection of all course characteristics data read in
119      * from "courses.txt".
120      */
121     public void loadCourses(String[] courseData) {
122
123         try {
124
125             //Create a new empty Course object.
126             Course newCourse = new Course();
```

```
127
128                  //Set the course ID to the first element in the course data array.
129                  newCourse.setCourseID(courseData[0].charAt(0));
130
131                  //Set the course length to the second element in the course data array.
132                  newCourse.setCourseLength(Integer.parseInt(courseData[1]));
133
134                  //Loop through the REST (i=2) of the "read-in" course data array..
135                  for (int i = 2; i < (courseData.length); i++) {
136
137                      //Loop through all the course nodes stored internally.
138                      for (Node n : data.getNodes()) {
139
140                          int origNodeNo = n.getNumber();
141
142                          //Obtain the node number currently being parsed from the read
143                          //   in data.
                             int courseNodeNo = Integer.parseInt(courseData[i]);
144
145                          /*
146                           * If the node number read-in from file matches the current
                             *    node,
147                           * and the node is NOT a junction, add this node to the
                             *    collection
148                           * of nodes within the new Course object.
149                           */
150                          if (origNodeNo == courseNodeNo && (n.getType().equals("CP") ||
                                 n.getType().equals("MC"))) {
151
152                              newCourse.addNewNode(n);
153                          }
154                      }
155
156                  }
157
158                  /*
159                   * Once the new Course object has been populated with data,
160                   * add it to the collection of courses in Datastore.
161                   */
162                  data.getCourses().add(newCourse);
163
164                  //If an error occurs...
165              } catch (Exception e) {
166
167                  //... log the error in the "log.txt" file.
168                  fileIO.addActivityLog("ERROR - Cannot create new course object (" +
                         courseData[0] + ")");
169              }
170
171      }
172
173      /**
174       * Parses the data read-in from the "nodes.txt" file and creates a new
175       * {@link aber.dcs.cs22510.clg11.model.Node} object populated with the
176       * read-in characteristics. This new Node object is then added to the
177       * internal collection of Nodes.
178       *
179       * @param nodeData Collection of all node characteristics data read in from
180       * "nodes.txt".
181       */
182      public void loadNodes(String[] nodeData) {
183
184          try {
185
186              Node newNode = new Node();
187
188              newNode.setNumber(Integer.parseInt(nodeData[0]));
```

```
189                 newNode.setType(nodeData[1]);
190
191             data.getNodes().add(newNode);
192
193         } catch (Exception e) {
194
195             fileIO.addActivityLog("ERROR - Cannot create new node object (" +
                    Integer.parseInt(nodeData[0]) + " / " + nodeData[1] + ")");
196         }
197
198     }
199
200     /**
201      * Parses the data read-in from the "entrants.txt" file and creates a new
202      * {@link aber.dcs.cs22510.clg11.model.Entrant} object populated with the
203      * read-in characteristics. This new Entrant object is then added to the
204      * internal collection of Entrants.
205      *
206      * @param entrantData Collection of all node characteristics data read in
207      * from "nodes.txt".
208      */
209     public void loadEntrants(String[] entrantData) {
210
211         try {
212
213             Entrant newEntrant = new Entrant();
214
215             newEntrant.setNumber(Integer.parseInt(entrantData[0]));
216             newEntrant.setCourseID(entrantData[1].charAt(0));
217             newEntrant.setFirstName(entrantData[2]);
218             newEntrant.setLastName(entrantData[3]);
219
220             data.getEntrants().add(newEntrant);
221
222         } catch (Exception e) {
223
224             fileIO.addActivityLog("ERROR - Cannot create new entrant object (" +
                    Integer.parseInt(entrantData[0]) + " / " + entrantData[1] + " / " +
                    entrantData[2] + " " + entrantData[3] + ")");
225         }
226
227     }
228 }
```

7.3 'Model' Package

7.3.1 Datastore.java

```java
package aber.dcs.cs22510.clg11.model;

import java.util.ArrayList;

/**
 * Stores all internal data used by the system to process existing and new
 * race time logs (Nodes, Courses and Entrants).
 *
 * @author Connor Luke Goddard (clg11)
 * Copyright: Aberystwyth University, Aberystwyth.
 */
public class Datastore {

    /** Arraylist of all courses in an event. */
    private ArrayList<Course> courses = new ArrayList<>();

    /** Arraylist of all nodes in an event. */
    private ArrayList<Node> nodes = new ArrayList<>();

    /** Arraylist of all entrants registered to an event. */
    private ArrayList<Entrant> entrants = new ArrayList<>();

    /**
     * Default constructor for a Course.
     */
    public Datastore() {


    }

    /**
     * Fetches all courses that are stored for a particular event.
     * @return The collection of courses.
     */
    public ArrayList<Course> getCourses() {
        return courses;
    }


    /**
     * Fetches all the nodes that are stored for a particular event.
     * @return The collection of nodes.
     */
    public ArrayList<Node> getNodes() {
        return nodes;
    }


    /**
     * Fetches all the entrants that are stored for a particular event.
     * @return The collection of entrants.
     */
    public ArrayList<Entrant> getEntrants() {
        return entrants;
    }

}
```

7.3.2 Entrant.java

```
 1  package aber.dcs.cs22510.clg11.model;
 2
 3  /**
 4   * Defines the data model for an entrant registered for an event.
 5   * Allows the setting and retrieval of data about a particular entrant.
 6   *
 7   * @author Connor Luke Goddard (clg11)
 8   * Copyright: Aberystwyth University, Aberystwyth.
 9   */
10  public class Entrant {
11
12      private String firstName;
13      private String lastName;
14
15      /** Entrant number used for tracking of entrant. */
16      private int number;
17
18      /** The current progress of the entrant along their registered course. */
19      private int currentProgress;
20
21      /** The ID character of the course the entrant is registered for. */
22      private char courseID;
23
24      /** Defines if the entrant is excluded or not. */
25      private boolean isExcluded = false;
26
27      /** Defines if the entrant has finished or not. */
28      private boolean isFinished = false;
29
30      /** Defines if the entrant is currently at a medical checkpoint. */
31      private boolean atMC = false;
32
33      /**
34       * Default constructor for an Entrant.
35       * Sets the current progress to 0 as a new entrant will
36       * not have started the race.
37       */
38      public Entrant() {
39
40          this.currentProgress = 0;
41
42      }
43
44      /**
45       * Constructor for an Entrant that allows their characteristics to be set upon
46       * instantiation.
47       * @param firstName The first name of the new entrant.
48       * @param lastName The last name of the new entrant.
49       * @param courseID The ID of the course the new entrant is registered for.
50       * @param enNumber The race number of the new entrant.
51       */
52      public Entrant(String firstName, String lastName, char courseID, int enNumber)
             {
53
54          this.firstName = firstName;
55          this.lastName = lastName;
56          this.courseID = courseID;
57          this.number = enNumber;
58          this.currentProgress = 0;
59
60      }
61
62      /**
63       * Fetches the full name (both first and last) name of the entrant.
64       * @return The full name of the entrant.
```

```
65          */
66         public String getFullName() {
67             return getFirstName() + " " + getLastName();
68         }
69
70         /**
71          * Sets the full name of the entrant by splitting the full
72          * name on a space and setting the separate first, and last names.
73          * @param name The inputted full name to be set.
74          */
75         public void setFullName(String name) {
76
77             //Split the inputted name by a space.
78             String[] tempName = name.split(" ");
79             this.setFirstName(tempName[0]);
80             this.setLastName(tempName[1]);
81         }
82
83         /**
84          * Returns the race number of the entrant.
85          * @return The race number of the entrant.
86          */
87         public int getNumber() {
88             return number;
89         }
90
91         /**
92          * Sets the race number of the entrant.
93          * @param number The race number to be set.
94          */
95         public void setNumber(int number) {
96             this.number = number;
97         }
98
99         /**
100         * Fetches the current progress of the entrant along their course.
101         * @return The current progress of the entrant on their course.
102         */
103        public int getCurrentProgress() {
104            return currentProgress;
105        }
106
107        /**
108         * Updates the current progress of the entrant along their course.
109         * @param currentProgress The incremented progress of the entrant.
110         */
111        public void setCurrentProgress(int currentProgress) {
112            this.currentProgress = currentProgress;
113        }
114
115        /**
116         * Returns the first name of the entrant.
117         * @return The first name of the entrant.
118         */
119        public String getFirstName() {
120            return firstName;
121        }
122
123        /**
124         * Sets the first name only of the entrant.
125         * @param firstName The first name of the entrant to be set.
126         */
127        public void setFirstName(String firstName) {
128            this.firstName = firstName;
129        }
130
131        /**
```

```
132        * Returns the last name of the entrant.
133        * @return The last name of the entrant.
134        */
135       public String getLastName() {
136           return lastName;
137       }
138
139       /**
140        * Sets the last name only of the entrant.
141        * @param lastName The last name of the entrant to be set.
142        */
143       public void setLastName(String lastName) {
144           this.lastName = lastName;
145       }
146
147       /**
148        * Fetches the course ID that the entrant is registered for.
149        * @return The ID of the registered course.
150        */
151       public char getCourseID() {
152           return courseID;
153       }
154
155       /**
156        * Sets the course ID of the entrant.
157        * @param courseID The ID of the course that the entrant is registered for.
158        */
159       public void setCourseID(char courseID) {
160           this.courseID = courseID;
161       }
162
163       /**
164        * Returns whether the entrant is excluded or not.
165        * @return Boolean determining if the entrant is excluded.
166        */
167       public boolean getIsExcluded() {
168           return isExcluded;
169       }
170
171       /**
172        * Sets whether or not the entrant is excluded.
173        * @param isExcluded Whether the entrant is excluded or not.
174        */
175       public void setIsExcluded(boolean isExcluded) {
176           this.isExcluded = isExcluded;
177       }
178
179       /**
180        * Returns whether the entrant has finished their race or not.
181        * @return Boolean determining if the entrant has finished.
182        */
183       public boolean getIsFinished() {
184           return isFinished;
185       }
186
187       /**
188        * Sets whether or not the entrant has finished their race or not.
189        * @param isFinished
190        */
191       public void setIsFinished(boolean isFinished) {
192           this.isFinished = isFinished;
193       }
194
195       /**
196        * Returns if the entrant is currently at a medical checkpoint.
197        * @return Boolean determining if the entrant is at an MC.
198        */
```

```
199     public boolean getAtMC() {
200         return atMC;
201     }
202
203     /**
204      * Sets if an entrant is at a medical checkpoint.
205      * @param atMC Whether the entrant is currently at an MC or not.
206      */
207     public void setAtMC(boolean atMC) {
208         this.atMC = atMC;
209     }
210
211 }
```

7.3.3 Course.java

```java
package aber.dcs.cs22510.clg11.model;

import java.util.ArrayList;

/**
 * Defines the data model for an event course.
 * Allows the setting and retrieval of data about a particular course.
 *
 * @author Connor Luke Goddard (clg11)
 * Copyright: Aberystwyth University, Aberystwyth.
 */
public class Course {

    /** Arraylist of Nodes that make up the Course. */
    private ArrayList<Node> courseNodes = new ArrayList<>();

    /** The total length of the course (i.e. the size of the course array).*/
    private int courseLength;

    /** The unique ID of a particular course. */
    private char courseID;

    /**
     * Default constructor for a Course.
     */
    public Course() {

    }

    /**
     * Adds a new {@link aber.dcs.cs22510.clg11.model.Node} to the collection
     * of nodes that make up the course.
     * @param newNode The new node to be added to the course.
     */
    public void addNewNode(Node newNode) {

        getCourseNodes().add(newNode);

    }

    /**
     * Fetches the collection of {@link aber.dcs.cs22510.clg11.model.Node}s that
     * make up the course.
     * @return The collection of nodes in the course.
     */
    public ArrayList<Node> getCourseNodes() {
        return courseNodes;
    }

    /**
     * Fetches the total size of the course.
     * @return The total size of the Arraylist of Nodes.
     */
    public int getCourseLength() {
        return courseLength;
    }

    /**
     * Sets the 'courseLength' value of the course.
     * @param courseLength The new courselength value.
     */
    public void setCourseLength(int courseLength) {
        this.courseLength = courseLength;
    }

```

```
66      /**
67       * Sets the Arraylist of course nodes.
68       * @param courseNodes The collection of course nodes to be set.
69       */
70      public void setCourseNodes(ArrayList<Node> courseNodes) {
71          this.courseNodes = courseNodes;
72      }
73
74      /**
75       * Fetches the ID character of the course.
76       * @return The ID of the current course.
77       */
78      public char getCourseID() {
79          return courseID;
80      }
81
82      /**
83       * Sets the ID of the current course.
84       * @param courseID The course ID to be set.
85       */
86      public void setCourseID(char courseID) {
87          this.courseID = courseID;
88      }
89
90  }
```

7.3.4 Node.java

```
 1  package aber.dcs.cs22510.clg11.model;
 2
 3  /**
 4   * Defines the data model for a course node within an event.
 5   * Allows the setting and retrieval of data about a particular course node.
 6   *
 7   * @author Connor Luke Goddard (clg11)
 8   * Copyright: Aberystwyth University, Aberystwyth.
 9   */
10  public class Node {
11
12      /** Type of the node. (CP, MC, JN) */
13      private String type;
14
15      /** The unique node number. */
16      private int number;
17
18      /**
19       * Default constructor for a Node.
20       */
21      public Node() {
22
23      }
24
25      /**
26       * Constructor for a Node that allows their characteristics to be set upon
27       * instantiation.
28       *
29       * @param cpNumber The new node number to be set.
30       * @param cpType The node type of the new node
31       */
32      public Node(int cpNumber, String cpType) {
33
34          this.number = cpNumber;
35          this.type = cpType;
36
37      }
38
39      /**
40       * Returns the node type of the current node.
41       * @return The type of the current node.
42       */
43      public String getType() {
44          return type;
45      }
46
47      /**
48       * Set the current node type.
49       * @param type The new node type to be set.
50       */
51      public void setType(String type) {
52          this.type = type;
53      }
54
55      /**
56       * Returns the ID number of the node.
57       * @return The number of the current node.
58       */
59      public int getNumber() {
60          return number;
61      }
62
63      /**
64       * Sets the ID number of the current node.
65       * @param number the number to set
```

```
66          */
67      public void setNumber(int number) {
68          this.number = number;
69      }
70
71  }
```

7.3.5 Datatype.java

```
package aber.dcs.cs22510.clg11.model;

/**
 * Public enumeration used to define the type of data that is to be read into
 * the system to allow the correct file read/parse methods to be used for the
 * type of data being read in.
 *
 * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public enum Datatype {

    /**
     *
     */
    COURSE,
    /**
     *
     */
    ENTRANT,
    /**
     *
     */
    NODE
};
```

7.4 'GUI' Package

7.4.1 GUIFrame.java

```java
package aber.dcs.cs22510.clg11.gui;

import aber.dcs.cs22510.clg11.model.Datastore;
import aber.dcs.cs22510.clg11.util.FileIO;
import aber.dcs.cs22510.clg11.util.LoadData;
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;

/**
 * Main JFrame for displaying program GUI. Responsible displaying main GUI
 * window and for instantiating the GUI sub-panel
 * {@link aber.dcs.cs22510.clg11.gui.GUIPanel}. Passes the new {@link aber.dcs.
 *     cs22510.clg11.model.Datastore} &
 * {@link aber.dcs.cs22510.clg11.util.LoadData} classes received from
 * {@link aber.dcs.cs22510.clg11.driver.CMDriver}, to the base panel as a
 * parameter to allow access to the data model from the sub-panel.
 *
 * @author Connor Goddard (clg11) Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class GUIFrame extends JFrame {

    /**
     * The new GUIPanel component.
     */
    private GUIPanel panel;

    /**
     * Constructor to instantiate a new GUIFrame. Takes the two classes created
     * in CMDriver as parameters to pass onto GUI sub-panel.
     *
     * @param newData Datastore class created in main method.
     */
    public GUIFrame(Datastore newData) {

        //Initialise and set up GUI frame (window).
        this.setTitle("Checkpoint Manager | Connor Goddard (clg11)");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Prevent user resizing frame.
        this.setResizable(false);

        //Initialise the sub-panel, passing the two shared components.
        panel = new GUIPanel(newData);

        //Add this panel to the whole of the frame (No layout set).
        this.add(panel);

        //Fit frame to ensure all panels/components are visible.
        this.pack();

        //Determine centre of user's screen and position frame accordingly.
        Toolkit k = Toolkit.getDefaultToolkit();
        Dimension d = k.getScreenSize();
        this.setLocation(d.width / 2 - this.getWidth() / 2, d.height / 2 - this.
            getHeight() / 2);

        //Display frame on screen.
        this.setVisible(true);
    }
}
```

7.4.2 GUIPanel.java

```java
package aber.dcs.cs22510.clg11.gui;

import aber.dcs.cs22510.clg11.model.Datastore;
import aber.dcs.cs22510.clg11.model.Entrant;
import aber.dcs.cs22510.clg11.model.Node;
import aber.dcs.cs22510.clg11.util.ProcessData;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import javax.swing.*;
import javax.swing.border.BevelBorder;

/**
 * Contains the GUI elements accessed by the user to interact with the
 * application. Allows the user to select entrants and nodes. Enter a new time
 * (or use system time) and then submit this new time to the log file. Any
 * problems that occur will be displayed to the user.
 *
 * @author Connor Goddard (clg11)
 * Copyright: Aberystwyth University,
 * Aberystwyth.
 */
public class GUIPanel extends JPanel implements ActionListener {

    /**
     * Buttons that represent system-wide operations.
     */
    private JButton submitTime, setCurrentTime;
    private JLabel nodeTitle, entrantTitle, mcTypeTitle, timeTitle, statusBar;

    /**
     * The layout manager used by the panel.
     */
    private SpringLayout layout = new SpringLayout();

    /**
     * Drop-down selected boxes used to select entrants and nodes.
     */
    private JComboBox<String> entrantList;
    private JComboBox<String> nodeList;

    /**
     * Determines whether an entrant is arriving or leaving medical checkpoint.
     */
    private JComboBox<String> mcTypeList;
    private String[] mcArriveDepart = {"Arriving", "Departing"};

    /**
     * Spinner used to allow the user to select a time value.
     */
    private JSpinner timeSpinner;
    private SpinnerDateModel sm;


    private JCheckBox mcExclude;


    /**
     * Enables the GUI to access the methods used for processing times.
     */
```

```
66    private ProcessData proc;
67    private Datastore data;
68
69    /**
70     * Constructor to instantiate a new GUIPanel. Takes the two classes passed
71     * from GUIFrame as parameters to allow panel to acccess shared data store
72     * and loading facilities.
73     *
74     * @param newData Datastore object passed down from GUIFrame.
75     */
76    public GUIPanel(Datastore newData) {
77
78        data = newData;
79        proc = new ProcessData(data);
80
81        //Set the size of the panel
82        this.setPreferredSize(new Dimension(500, 250));
83
84        //Set the bespoke layout manager.
85        this.setLayout(layout);
86
87        //Initialise and add all of the panel GUI components.
88        initComponents();
89
90        setUpLayout();
91
92    }
93
94    /**
95     * Initialises the panel components (including linking components to
96     * listeners where required) before adding the components to the panel.
97     */
98    private void initComponents() {
99
100        String[] comboValues;
101
102        /*
103         * Instantiate new ProcessData class to allow access to data processing
104         * facilties.
105         */
106
107        //Create new instance of JLabel with specified display text
108        entrantTitle = new JLabel("Entrant List:");
109        nodeTitle = new JLabel("Checkpoint List:");
110        mcTypeTitle = new JLabel("Medical CP Type:");
111        timeTitle = new JLabel("Log Time:");
112
113        statusBar = new JLabel("Welcome to the Checkpoint Manager.");
114        statusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
115        statusBar.setPreferredSize(new Dimension(500, 20));
116
117        //Load all entrant names into entrant drop-down GUI box component.
118        comboValues = Arrays.copyOf(getAllEntrants().toArray(), getAllEntrants().
                toArray().length, String[].class);
119
120        entrantList = new JComboBox<>(comboValues);
121        entrantList.setSelectedIndex(0);
122
123        //Load all node numbers into node drop-down GUI box component.
124        comboValues = Arrays.copyOf(getAllCheckpoints().toArray(),
                getAllCheckpoints().toArray().length, String[].class);
125
126        nodeList = new JComboBox<>(comboValues);
127        nodeList.setSelectedIndex(0);
128
129        //Add local action listener (Required for determining a MC).
130        nodeList.addActionListener(this);
```

```java
131
132                //Load the MC "arrive/depart" options into drop-down GUI box.
133                mcTypeList = new JComboBox<>(mcArriveDepart);
134                mcTypeList.setSelectedIndex(0);
135                mcTypeList.setEnabled(false);
136
137
138                //Create new instance of JButton with specified button text
139                submitTime = new JButton("Submit Checkpoint Time");
140                submitTime.addActionListener(this);
141
142                setCurrentTime = new JButton("Set to Current Time");
143                setCurrentTime.addActionListener(this);
144
145
146                //Create new JSpinner model that will access the current system time.
147                sm = new SpinnerDateModel();
148                sm.setCalendarField(Calendar.MINUTE);
149
150                //Create a new Spinner object and set the above model to it.
151                timeSpinner = new JSpinner();
152                timeSpinner.setModel(sm);
153
154                //Set the time format to be diplayed in the JSpinner.
155                JSpinner.DateEditor de = new JSpinner.DateEditor(timeSpinner, "HH:mm");
156                timeSpinner.setEditor(de);
157
158                mcExclude = new JCheckBox("Exclude Entrant");
159                mcExclude.setSelected(false);
160                mcExclude.setEnabled(false);
161
162                //Add all the components to the GUI panel.
163                this.add(nodeTitle);
164                this.add(entrantTitle);
165                this.add(mcTypeTitle);
166                this.add(timeTitle);
167                this.add(statusBar);
168                this.add(timeSpinner);
169                this.add(mcExclude);
170                this.add(entrantList);
171                this.add(nodeList);
172                this.add(mcTypeList);
173                this.add(submitTime);
174                this.add(setCurrentTime);
175
176        }
177
178        /**
179         * Sets up the 'SpringLayout' layout manager to organise all components on
180         * the panel.
181         */
182        private void setUpLayout() {
183
184                //Set the NORTH edge of the label to be 10 pixels down from the NORTH edge
185                //    of the panel.
                 layout.putConstraint(SpringLayout.NORTH, nodeTitle, 10, SpringLayout.NORTH,
                     this);
186
187                //Set the WEST edge of the label to be 10 pixels left of the WEST edge of
                 //    the panel.
188                layout.putConstraint(SpringLayout.WEST, nodeTitle, 10, SpringLayout.WEST,
                     this);
189
190                layout.putConstraint(SpringLayout.NORTH, nodeList, 10, SpringLayout.NORTH,
                     this);
191                layout.putConstraint(SpringLayout.WEST, nodeList, 10, SpringLayout.EAST,
                     nodeTitle);
```

```
192
193            layout.putConstraint(SpringLayout.NORTH, entrantTitle, 10, SpringLayout.
                   SOUTH, nodeTitle);
194            layout.putConstraint(SpringLayout.WEST, entrantTitle, 10, SpringLayout.WEST
                   , this);
195
196            layout.putConstraint(SpringLayout.NORTH, entrantList, 10, SpringLayout.
                   SOUTH, nodeTitle);
197            layout.putConstraint(SpringLayout.WEST, entrantList, 10, SpringLayout.EAST,
                    entrantTitle);
198
199            layout.putConstraint(SpringLayout.NORTH, mcTypeTitle, 10, SpringLayout.
                   SOUTH, entrantTitle);
200            layout.putConstraint(SpringLayout.WEST, mcTypeTitle, 10, SpringLayout.WEST,
                    this);
201
202            layout.putConstraint(SpringLayout.NORTH, mcTypeList, 10, SpringLayout.SOUTH
                   , entrantTitle);
203            layout.putConstraint(SpringLayout.WEST, mcTypeList, 10, SpringLayout.EAST,
                   mcTypeTitle);
204
205            layout.putConstraint(SpringLayout.NORTH, mcExclude, 10, SpringLayout.SOUTH,
                    mcTypeTitle);
206            layout.putConstraint(SpringLayout.WEST, mcExclude, 10, SpringLayout.WEST,
                   this);
207
208            layout.putConstraint(SpringLayout.NORTH, timeTitle, 10, SpringLayout.SOUTH,
                    mcExclude);
209            layout.putConstraint(SpringLayout.WEST, timeTitle, 10, SpringLayout.WEST,
                   this);
210
211            layout.putConstraint(SpringLayout.NORTH, timeSpinner, 10, SpringLayout.
                   SOUTH, mcExclude);
212            layout.putConstraint(SpringLayout.WEST, timeSpinner, 10, SpringLayout.EAST,
                    timeTitle);
213
214            layout.putConstraint(SpringLayout.NORTH, setCurrentTime, 10, SpringLayout.
                   SOUTH, timeTitle);
215            layout.putConstraint(SpringLayout.WEST, setCurrentTime, 10, SpringLayout.
                   WEST, this);
216
217            layout.putConstraint(SpringLayout.NORTH, submitTime, 10, SpringLayout.SOUTH
                   , setCurrentTime);
218            layout.putConstraint(SpringLayout.WEST, submitTime, 10, SpringLayout.WEST,
                   this);
219
220            layout.putConstraint(SpringLayout.SOUTH, statusBar, 0, SpringLayout.SOUTH,
                   this);
221        }
222
223        /**
224         * Obtains a list of all the entrant names to populate selection box.
225         * Accesses them from the array list of entrants contained within
226         * {@link aber.dcs.cs22510.clg11.model.Datastore}.
227         *
228         * @return Arraylist of all the entrant's names.
229         */
230        public ArrayList<String> getAllEntrants() {
231
232            ArrayList<String> entrantList = new ArrayList<>();
233
234            for (Entrant e : data.getEntrants()) {
235
236                entrantList.add(e.getFullName());
237
238            }
239
```

```
240          return entrantList;
241
242      }
243
244      /**
245       * Obtains a list of all the checkpoints ONLY to populate the CP selection
246       * box. Accesses them from the array list of nodes contained within
247       * {@link aber.dcs.cs22510.clg11.model.Datastore}.
248       *
249       * @return Arraylist of all the nodes that are CHECKPOINTS.
250       */
251      public ArrayList<String> getAllCheckpoints() {
252
253          ArrayList<String> checkpointList = new ArrayList<>();
254
255          //Loop through all the nodes.
256          for (Node cp : data.getNodes()) {
257
258              //If the current node is a checkpoint, and not a junction, add it.
259              if (cp.getType().equals("CP") || cp.getType().equals("MC")) {
260
261                  checkpointList.add(Integer.toString(cp.getNumber()));
262
263              }
264
265          }
266
267          return checkpointList;
268
269      }
270
271      /**
272       * Attempts to fetch a specific node denoted by the node number selected
273       * from the drop-down GUI box. If such a node cannot be found, NULL us
274       * returned.
275       *
276       * @param nodeNo The number of the selected node.
277       * @return The located node or NULL.
278       */
279      public Node getNode(int nodeNo) {
280
281          for (Node n : data.getNodes()) {
282
283              if (n.getNumber() == nodeNo) {
284
285                  return n;
286              }
287          }
288
289          return null;
290
291      }
292
293      /**
294       * Submits a new time log based on user input within the GUI and determines
295       * if an time file currently exists, or if a new one has to be created.
296       */
297      public void submitCheckpoint() {
298
299          try {
300
301
302              //Display question dialog
303              int shouldProcess = JOptionPane.YES_OPTION;
304
305              shouldProcess = JOptionPane.showConfirmDialog(null, "Are you sure you
                      wish to submit this time log?",
```

```java
306                         "CM Manager | Submit Time Log", JOptionPane.YES_NO_OPTION);
307
308             //If user selects "yes"
309             if (shouldProcess == JOptionPane.YES_OPTION) {
310
311                 //Create a formatter for the time value entered by the user.
312                 SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
313
314                 String newTimeValue = sdf.format(timeSpinner.getValue());
315
316                 //Check if a times file currently exists.
317                 if (proc.getTimes()) {
318
319                     updateStatus(" Times file loaded successfully.", false);
320
321                     //Check to see if the time entered by the user is in the past.
322                     if (proc.compareTimes(proc.getLastLoggedTime(), newTimeValue))
                        {
323
324                         updateStatus(" ERROR: Time entered is before last recorded
                            time. Please try again.", true);
325
326                     } else {
327
328                         /*
329                          * Re-read in the "times" file to allow any new times
                             logged by other
330                          * running versions of the checkpoint manager to update the
                             information
331                          * contained within this version of the CM.
332                          */
333                         updateTimeLog();
334                     }
335
336                     //If a time file does not currently exist, create a new one.
337                 } else {
338
339                     updateStatus(" ALERT: Times file (times.txt) not found.
                        Creating new time log file.", true);
340                     updateTimeLog();
341                 }
342
343             }
344
345         } catch (IndexOutOfBoundsException iOB) {
346
347             updateStatus(" ERROR: Cannot parse times file. Problem reading file.",
                true);
348
349         }
350     }
351
352     /**
353      * Takes the user input and processes the new time log entry before adding
354      * it to the times log file.
355      */
356     public void updateTimeLog() {
357
358         String result = null;
359
360         //Obtain the selected entrant from the arraylist of entrants.
361         Entrant currentEntrant = data.getEntrants().get(entrantList.
                getSelectedIndex());
362
363         //Obtain the arraylist of course nodes that make up the course that entrant
                is registered for.
```

```
364            ArrayList<Node> entrantNodes = proc.obtainEntrantCourseNodes(currentEntrant
                   );
365
366            //Create a formatter for the time value entered by the user.
367            SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
368
369            //Obtain the number of the node selected by the user.
370            int nodeNumber = Integer.parseInt((String) nodeList.getSelectedItem());
371
372            //Format the time value entered by the user.
373            String newTimeValue = sdf.format(timeSpinner.getValue());
374
375            //Check to see if the node selected was a MC.
376            if (mcTypeList.isEnabled()) {
377
378                //If so determine whether they were arriving or departing.
379                String mcSelection = (String) mcTypeList.getSelectedItem();
380
381                /*
382                 * Check if the user is attempting to log an entrant arriving
383                 * at a MC while the entrant is currently at an MC.
384                 */
385                if (currentEntrant.getAtMC() && mcSelection.equals("Arriving")) {
386
387                    updateStatus(" ERROR: Entrant " + currentEntrant.getNumber() + "
                           already at medical checkpoint.", true);
388
389                } else if (!(currentEntrant.getAtMC()) && mcSelection.equals("Departing
                       ")) {
390
391                    updateStatus(" ERROR: Entrant must be at MC before they can depart.
                           ", true);
392
393                } else {
394
395                    result = proc.processTimeLog(entrantNodes, currentEntrant,
                           nodeNumber, mcSelection, newTimeValue, mcExclude.isSelected());
396
397                    updateStatus(result, false);
398                }
399
400            } else {
401
402                //The checkpoint is not a MC, and so just process the new logged time.
403                result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber,
                       newTimeValue);
404                updateStatus(result, false);
405            }
406        }
407
408        /**
409         * Displays system status/error messages to the user via the GUI.
410         * @param updateMessage The message to be displayed.
411         * @param isError Determines whether the message is an error or not.
412         */
413        public void updateStatus(String updateMessage, boolean isError) {
414
415            statusBar.setText(updateMessage);
416
417            if (isError) {
418
419                statusBar.setForeground(Color.RED);
420
421            } else {
422
423                statusBar.setForeground(Color.BLACK);
424            }
```

```java
425
426        }
427
428        /**
429         * Listener for actions from sub-panel components, to allow operations to be
430         * run when components are interacted with.
431         *
432         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.
                  ActionEvent)
433         * @param evt - ActionEvent called from components in the panels that
434         * require an action to be performed.
435         */
436        @Override
437        public void actionPerformed(ActionEvent evt) {
438
439            String actionCommand = evt.getActionCommand();
440
441
442            //Switch statement used to capture action commands from buttons.
443            switch (actionCommand) {
444
445                case "Submit Checkpoint Time":
446
447                    //Submit the entered time values.
448                    submitCheckpoint();
449                    break;
450
451                case "Set to Current Time":
452
453                    //Obtain the current system time from the Calendar class.
454                    Calendar currentTime = Calendar.getInstance();
455
456                    //Update the value of the time spinner to the current time.
457                    timeSpinner.setValue(currentTime.getTime());
458
459                    updateStatus(" Current time updated successfully.", false);
460                    break;
461
462            }
463
464
465            //Listen for events on the nodes drop-down box component.
466            if (evt.getSource() == nodeList) {
467
468                //Obtain the selected Node object from the collection of nodes.
469                Node n = getNode(Integer.parseInt((String) nodeList.getSelectedItem()))
                        ;
470
471                //Determine whether the selected node is a medical checkpoint.
472                if (n.getType().equals("MC")) {
473
474                    //If it is, allow the "arrive/depart" selection box to be used.
475                    mcTypeList.setEnabled(true);
476                    mcExclude.setEnabled(true);
477
478                } else {
479
480                    mcTypeList.setEnabled(false);
481                    mcExclude.setEnabled(false);
482                }
483            }
484        }
485 }
```

8   Checkpoint Manager - Build/Compilation Log

The listing below contains the build/compilation log for the "checkpoint manager" application. Extra warning flags (`-Xlint:unchecked`) have been used with the JVM compiler to ensure that no errors/warnings occur when compiling the application.

Listing 6: Compilation log built within Netbeans IDE 7.3 on Ubuntu 12.04

```
 1  ant -f /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager clean jar
 2  init:
 3  deps-clean:
 4  Updating property file: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/built-
        clean.properties
 5  Deleting directory /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
 6  clean:
 7  init:
 8  deps-jar:
 9  Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
10  Updating property file: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/built-
        jar.properties
11  Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/classes
12  Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/empty
13  Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/generated-sources
        /ap-source-output
14  Compiling 11 source files to /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/
        classes
15  compile:
16  Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist
17  Copying 1 file to /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
18  Nothing to copy.
19  Building jar: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist/Checkpoint-
        Manager.jar
20  To run this application from the command line without Ant, try:
21  java -jar "/home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist/Checkpoint-Manager.
        jar"
22  jar:
23  BUILD SUCCESSFUL (total time: 0 seconds)
```

9 Checkpoint Manager - Example Usage

This section demonstrates the "checkpoint manager" application running using test input data to ensure that expected functionality and suitable error checking is taking place correctly.

9.1 Loading External Data Files

9.1.1 Correct File Parameters

Parameter Input: `"../files/nodes.txt" "../files/courses.txt" "../files/entrants.txt"`

Output:

Listing 7: Textual output produced when correct file names are supplied.

```
run:
Nodes file loaded successfully (nodes.txt)
Courses file loaded successfully (courses.txt)
Entrants file loaded successfully (entrants.txt)
```
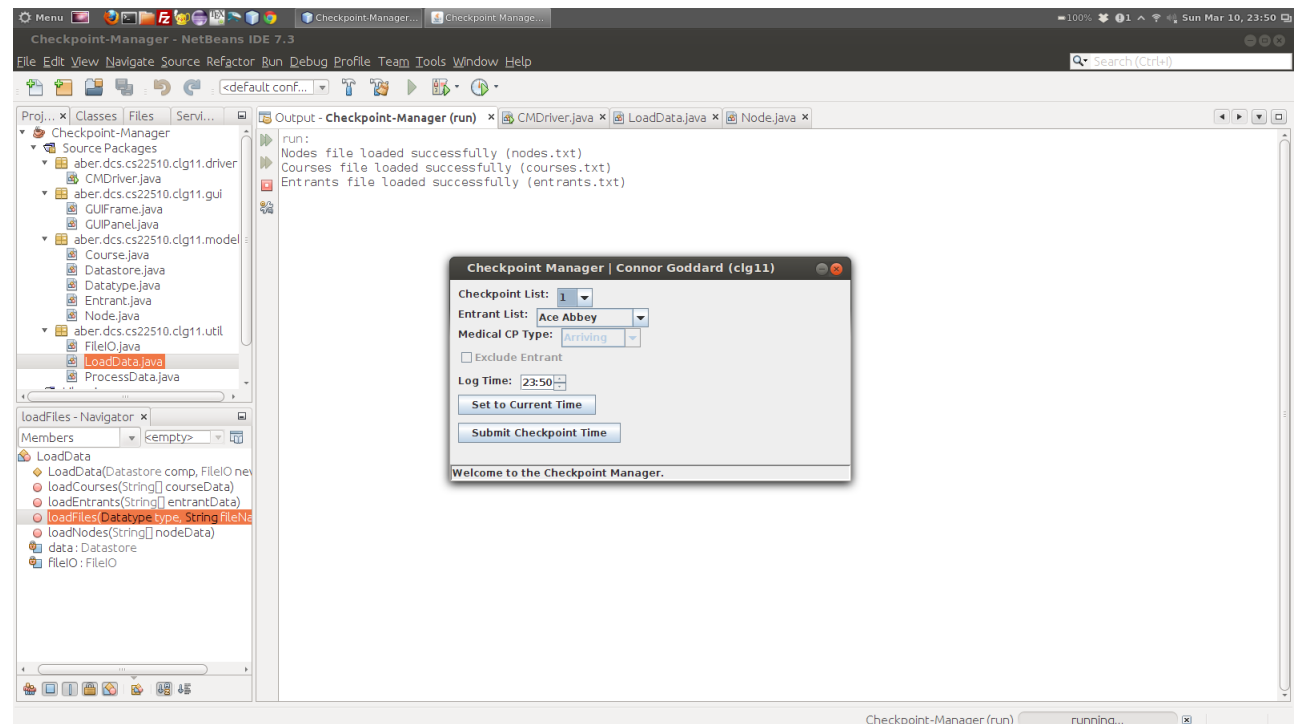

Figure 1: Screenshot displaying the GUI display after successfully loading the external data files.

9.1.2   Incorrect File Parameters

Parameter Input: `"../files/nodes.txt" "../files/idontknow.txt" "../files/entrants.txt""`

Output:

Listing 8: Textual warning output produced when incorrect file names parameters are supplied.

```
run:
Nodes file loaded successfully (nodes.txt)
ERROR: Courses file <../files/idontknow.txt> does not exist.
Parameter format = <node path> <courses path> <entrants path>
BUILD SUCCESSFUL (total time: 0 seconds)
```

9.2   Submit Correct Time Entry

Input: `Entrant 1 - Checkpoint 1 (Starting checkpoint for course).`
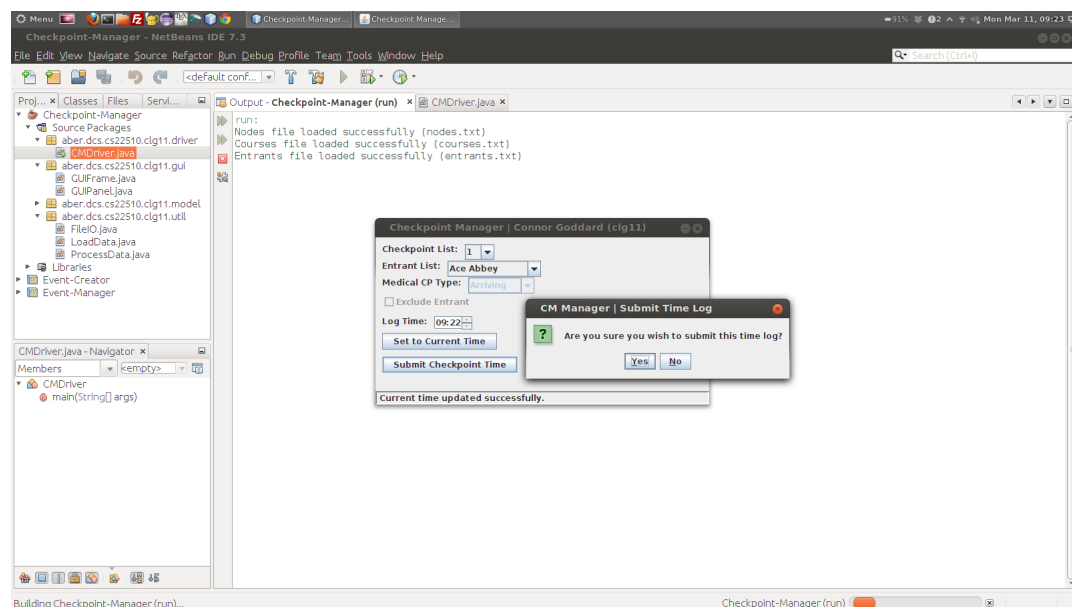
Figure 2: Screenshot displaying attempt to submit a correct time entry for an entrant.

Output:
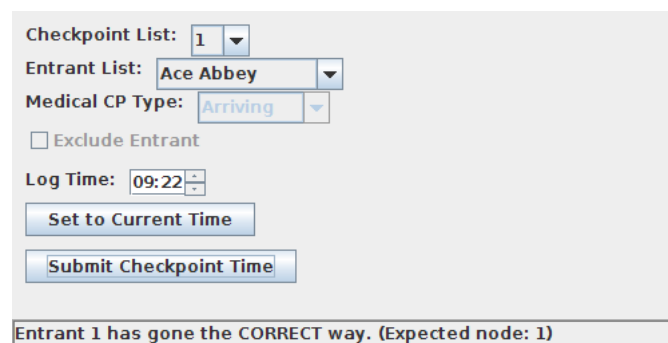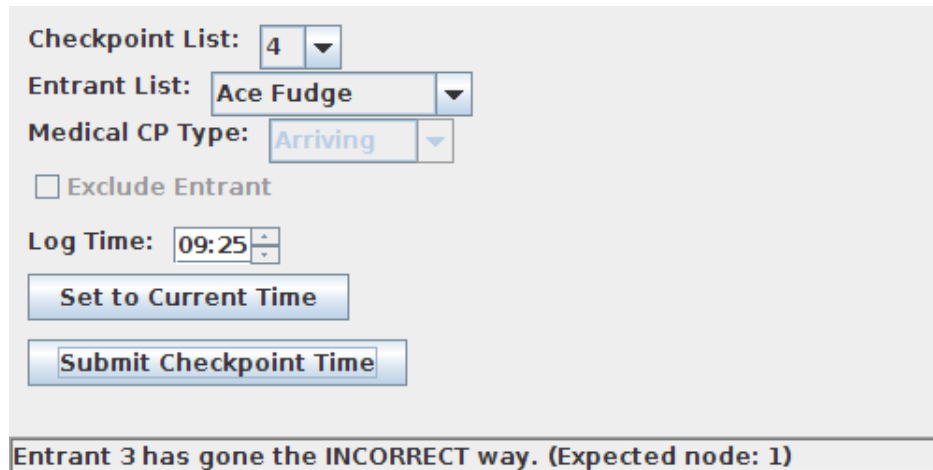
Figure 3: Checkpoint Manager GUI confirming successful submission of time entry.

9.3 Submit Incorrect Time Entry

Input: `Entrant 3 - Checkpoint 4 (Incorrect - should be CP 1).`

Output:
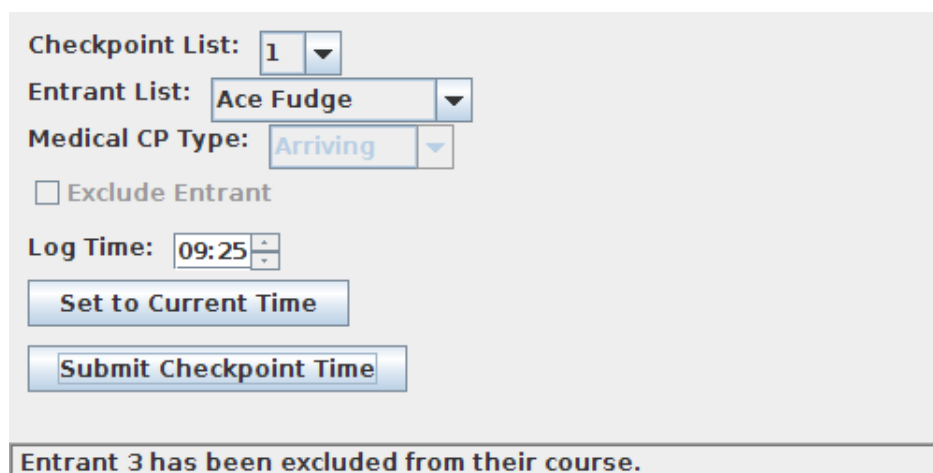
Figure 4: Checkpoint Manager GUI informing user that entrant has been logged at an incorrect checkpoint.

9.4 Entrant Exclusion (Incorrect Node)

Input: `Entrant 3 - Checkpoint 1.`

Output:
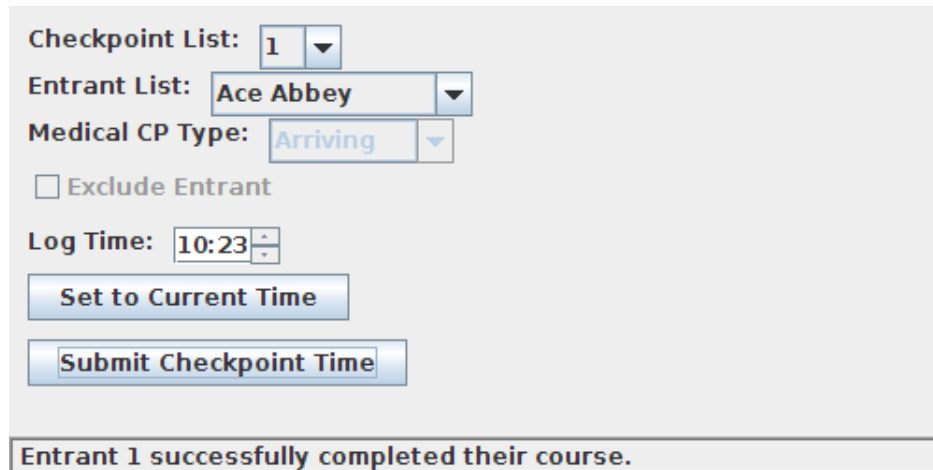
Figure 5: Checkpoint Manager GUI informing user that entrant 3 has already been excluded from the event.

9.5 Entrant Course Completion

Input: `Entrant 1 - Checkpoint 1 (After entrant has finished course).`

Output:
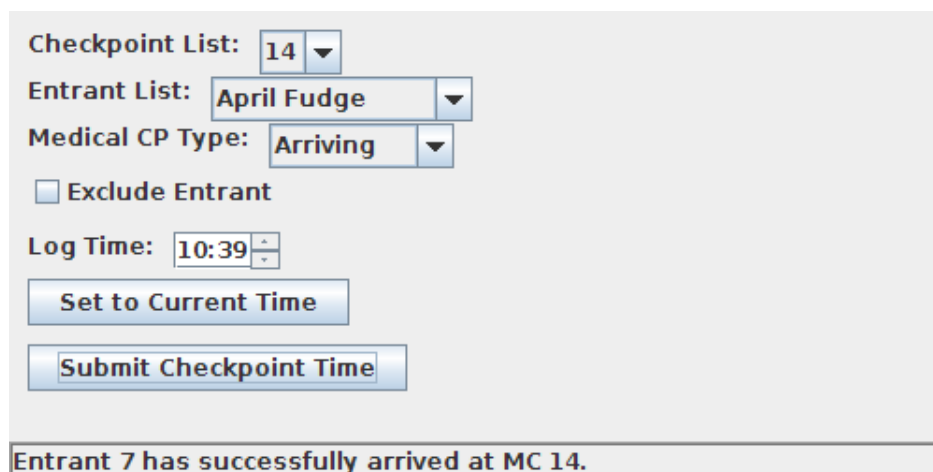
Figure 6: Checkpoint Manager GUI informing user that entrant 1 has successfully completed their course.

9.6 Medical Checkpoint - Successful Arrival

Input: `Entrant 7 - MC 14 (Arriving).`

Output:
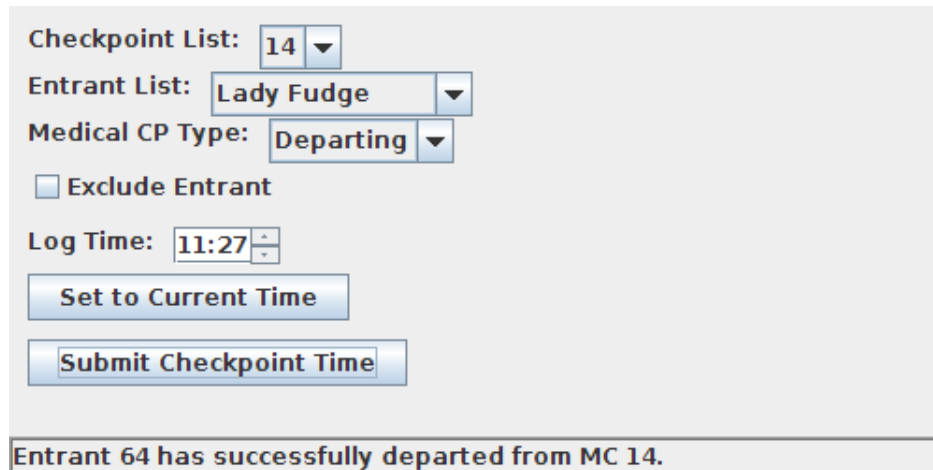
Figure 7: Checkpoint Manager GUI informing user that entrant 7 has successfully arrived (correctly) at medical checkpoint 14.

9.7 Medical Checkpoint - Successful Departure

Input: `Entrant 64 - MC 14 (Departing).`

Output:
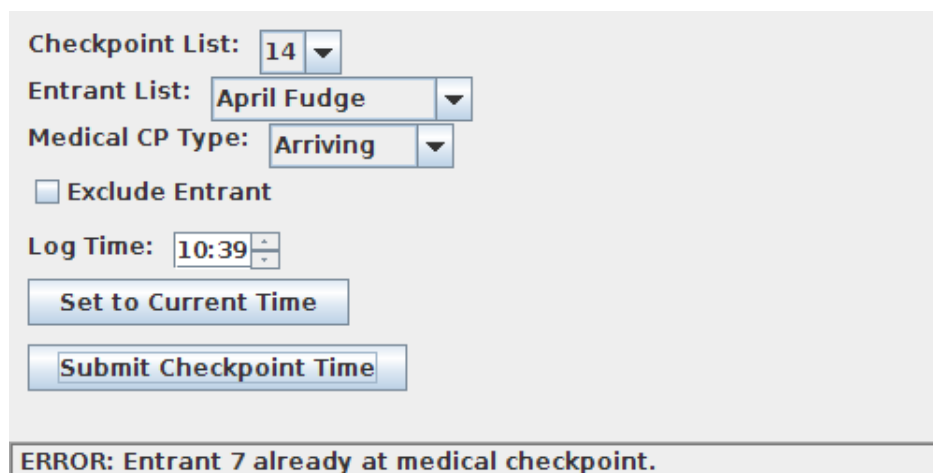
Figure 8: Checkpoint Manager GUI informing user that entrant 64 has successfully departed (correctly) from medical checkpoint 14.

9.8 Medical Checkpoint - Incorrect Arrival

Input: `Entrant 7 - MC 14 (Arriving - Already at MC 14).`

Output:

Figure 9: Checkpoint Manager GUI informing user that entrant 7 is already logged at MC 14 and so cannot have arrived again.

9.9 Medical Checkpoint - Incorrect Departure

Input: `Entrant 98 - MC 14 (Departing - Entrant yet to begin course).`

Output:

Figure 10: Checkpoint Manager GUI informing user that entrant 98 must have arrived at a MC before they can depart.

9.10 Entrant Exclusion (Medical Reasons)

Input: `Entrant 7 - MC 14.`

Output:

Figure 11: Checkpoint Manager GUI informing user that entrant 7 has been successfully excluded for medical reasons.

9.11 Submission of Invalid Time Value

Input: `Submitted Time: 08:46. Latest Logged Time: 09:37.`

Output:

Figure 12: Checkpoint Manager GUI informing user that submitted time cannot be before the latest logged time (**Ensures time log file remains sequential**).

9.12 System Activity Logging

Input:

```
Variety of system activity. (Correct/incorrect node submissions,
automatic loading of data files etc...)
```

Output:

Listing 9: Example of log file produced by CM application of activity detailed above.

```
LOG - CM: Nodes file loaded successfully (nodes.txt) - 11/03/2013 10:24:42
LOG - CM: Courses file loaded successfully (courses.txt) - 11/03/2013 10:24:42
LOG - CM: Entrants file loaded successfully (entrants.txt) - 11/03/2013 10:24:42
LOG - CM: Entrant no: 3 successfully excluded. - 11/03/2013 10:24:46
LOG - CM: Entrant no: 1 has sucessfully finished the course. - 11/03/2013 10:24:46
LOG - CM: Entrant no: 7 successfully excluded. - 11/03/2013 10:24:46
LOG - CM: Entrant no: 64 has sucessfully finished the course. - 11/03/2013 10:24:46
LOG - CM: Nodes file loaded successfully (nodes.txt) - 11/03/2013 10:28:05
LOG - CM: Courses file loaded successfully (courses.txt) - 11/03/2013 10:28:05
LOG - CM: Entrants file loaded successfully (entrants.txt) - 11/03/2013 10:28:05
LOG - CM: Entrant no: 3 successfully excluded. - 11/03/2013 10:28:09
LOG - CM: Entrant no: 1 has sucessfully finished the course. - 11/03/2013 10:28:09
LOG - CM: Entrant no: 7 successfully excluded. - 11/03/2013 10:28:09
LOG - CM: Entrant no: 64 has sucessfully finished the course. - 11/03/2013 10:28:09
LOG - CM: Nodes file loaded successfully (nodes.txt) - 11/03/2013 10:37:41
LOG - CM: Courses file loaded successfully (courses.txt) - 11/03/2013 10:37:41
LOG - CM: Entrants file loaded successfully (entrants.txt) - 11/03/2013 10:37:41
LOG - CM: Entrant no: 3 successfully excluded. - 11/03/2013 10:37:53
LOG - CM: Entrant no: 1 has sucessfully finished the course. - 11/03/2013 10:37:53
LOG - CM: Entrant no: 7 successfully excluded. - 11/03/2013 10:37:53
LOG - CM: Entrant no: 64 has sucessfully finished the course. - 11/03/2013 10:37:53
LOG - CM: Time logs file loaded successfully (times.txt) - 11/03/2013 10:37:53
LOG - CM: User attempted to enter new time value in the past. (New time: 10:37) - 11/03/2013
    10:37:53
LOG - CM: Time logs file loaded successfully (times.txt) - 11/03/2013 10:37:59
LOG - CM: User attempted to enter new time value in the past. (New time: 11:37) - 11/03/2013
    10:37:59
LOG - CM: Time logs file loaded successfully (times.txt) - 11/03/2013 10:38:08
```

9.13   File Lock Access Prevention

Input:

Two instances of the Checkpoint Manager application deployed.

One application modified to prevent the release of the file lock.

Listing 10: File export code modified to prevent a file lock being released once accessed.

```
//Check if the lock was successfull.
         if (fl != null) {

             try (FileWriter fw = new FileWriter(fos.getFD())) {

                 fw.write(output);

                 //Once the data has been successfully written, release the lock
                     .

                 //FILE LOCK RELEASE PREVENTED.
                 //fl.release();
             }

             return true;

         }
```

Output:
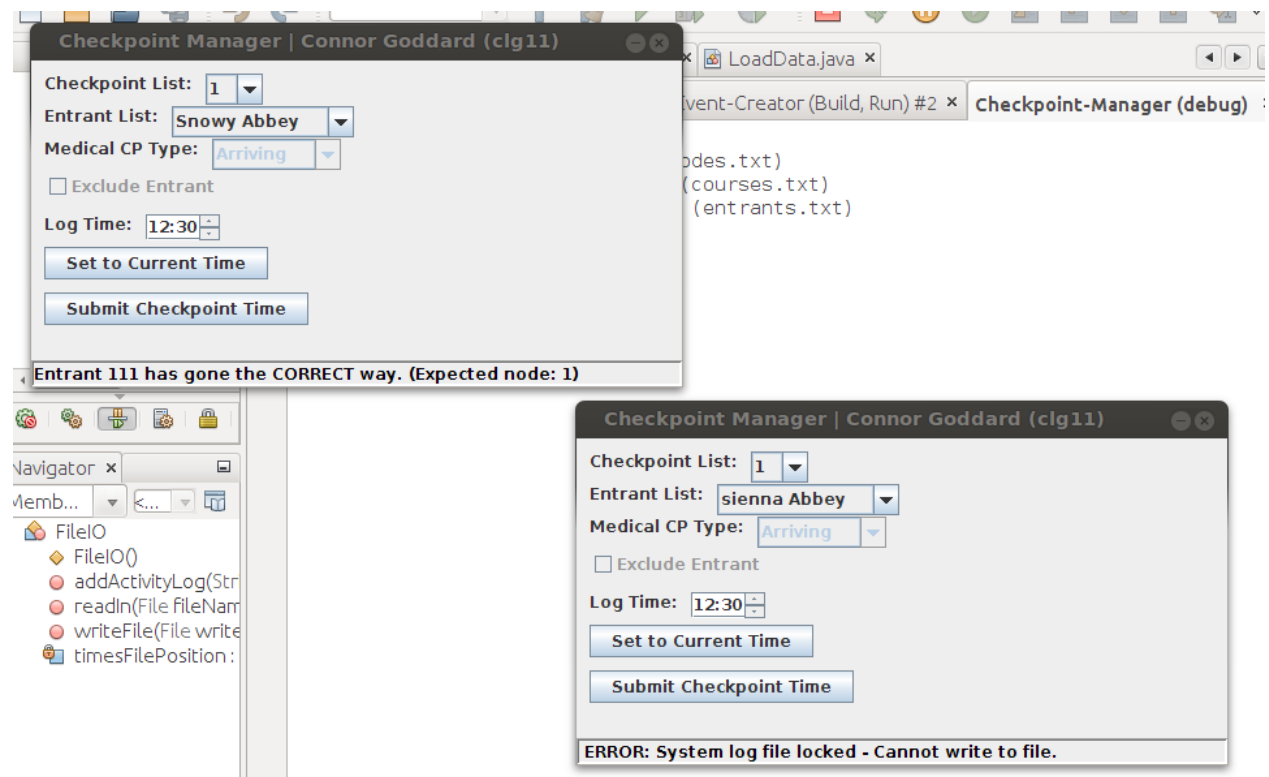
Figure 13: Screenshot demonstrating modified CM application (top-left) preventing original CM application (bottom-right) from accessing log file "log.txt".

9.14 Entrant Times File Generation

This sub-section contains the times file ("times.txt") generated by the checkpoint manager application from the example functionality testing detailed above.

Listing 11: Example of times file produced by CM application from the activity detailed above.

```
T 1 1 09:22
I 4 3 09:25
T 9 1 09:37
T 1 7 09:38
T 13 1 09:45
T 1 1 10:23
T 4 7 10:23
T 1 64 10:25
T 5 7 10:28
T 4 64 10:29
T 7 7 10:35
A 14 7 10:39
E 14 7 10:42
T 5 64 10:50
T 7 64 10:57
A 14 64 11:09
D 14 64 11:27
T 13 64 11:36
T 1 64 11:56
T 1 89 12:30
T 1 103 12:30
T 1 111 12:30
```

10   Event Manager (C) - Build/Compilation Log

The listing below contains the build/compilation log for the "event manager" application. Extra warning flags (`-wall`, `-ansi`, `-std=c89`) have been used with the C compiler (**gcc**) to ensure that any possible errors/warnings are detected during compilation.

Listing 12: Event manager compilation log built within Netbeans IDE 7.3 on Ubuntu 12.04

```
 1  "/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-conf
 2  make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
 3  rm -f -r build/Debug
 4  rm -f dist/Debug/GNU-Linux-x86/event-manager
 5  make[1]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
 6
 7
 8  CLEAN SUCCESSFUL (total time: 59ms)
 9
10  "/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
11  make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
12  "/usr/bin/make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/event-manager
13  make[2]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
14  mkdir -p build/Debug/GNU-Linux-x86
15  rm -f build/Debug/GNU-Linux-x86/course.o.d
16  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/course.o.d -o
        build/Debug/GNU-Linux-x86/course.o course.c
17  mkdir -p build/Debug/GNU-Linux-x86
18  rm -f build/Debug/GNU-Linux-x86/display.o.d
19  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/display.o.d -o
        build/Debug/GNU-Linux-x86/display.o display.c
20  mkdir -p build/Debug/GNU-Linux-x86
21  rm -f build/Debug/GNU-Linux-x86/entrant.o.d
22  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/entrant.o.d -o
        build/Debug/GNU-Linux-x86/entrant.o entrant.c
23  mkdir -p build/Debug/GNU-Linux-x86
24  rm -f build/Debug/GNU-Linux-x86/event.o.d
25  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/event.o.d -o
        build/Debug/GNU-Linux-x86/event.o event.c
26  mkdir -p build/Debug/GNU-Linux-x86
27  rm -f build/Debug/GNU-Linux-x86/fileIO.o.d
28  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/fileIO.o.d -o
        build/Debug/GNU-Linux-x86/fileIO.o fileIO.c
29  mkdir -p build/Debug/GNU-Linux-x86
30  rm -f build/Debug/GNU-Linux-x86/main.o.d
31  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/main.o.d -o build
        /Debug/GNU-Linux-x86/main.o main.c
32  mkdir -p build/Debug/GNU-Linux-x86
33  rm -f build/Debug/GNU-Linux-x86/node.o.d
34  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/node.o.d -o build
        /Debug/GNU-Linux-x86/node.o node.c
35  mkdir -p build/Debug/GNU-Linux-x86
36  rm -f build/Debug/GNU-Linux-x86/process.o.d
37  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/process.o.d -o
        build/Debug/GNU-Linux-x86/process.o process.c
38  mkdir -p build/Debug/GNU-Linux-x86
39  rm -f build/Debug/GNU-Linux-x86/track.o.d
40  gcc -ansi    -c -g -Wall -std=c89 -ansi -MMD -MP -MF build/Debug/GNU-Linux-x86/track.o.d -o
        build/Debug/GNU-Linux-x86/track.o track.c
41  mkdir -p dist/Debug/GNU-Linux-x86
42  gcc -ansi     -o dist/Debug/GNU-Linux-x86/event-manager build/Debug/GNU-Linux-x86/course.o build
        /Debug/GNU-Linux-x86/display.o build/Debug/GNU-Linux-x86/entrant.o build/Debug/GNU-Linux-
        x86/event.o build/Debug/GNU-Linux-x86/fileIO.o build/Debug/GNU-Linux-x86/main.o build/
        Debug/GNU-Linux-x86/node.o build/Debug/GNU-Linux-x86/process.o build/Debug/GNU-Linux-x86/
        track.o
43  make[2]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
44  make[1]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Manager'
45
46
47  BUILD SUCCESSFUL (total time: 527ms)
```

11 Event Manager - Example Usage

This section demonstrates the "event manager" application running using test data generated from the "event creator" and "checkpoint manager" applications to ensure that expected functionality and suitable error checking is taking place correctly.

Please note: The output below has been modified to reduce the amount of paper used, however no values/results have been changed.

Listing 13: Example output of functionality testing of the event manager application.

```
Enter event description file name: ../files/exampleevent.txt

the test horse event
08 Jul 2013
09:45

Enter node file name: ../files/nods.txt

File could not be opened.
Would you like to try again? (1 = Yes, 2 = No):
1

Enter node file name: ../files/nodes.txt

Enter track file name: ../files/tracks.txt

Enter courses file name: ../files/courses.txt

Enter entrants file name: ../files/entrants.txt

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
1

|  Competitor No  |                      Competitor Name                      |  Current Status  |
     |
|=================|===========================================================|==================|

|        1        |   Ace Abbey                                               |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        3        |   Ace Fudge                                               |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        4        |   Amber Abbey                                             |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        5        |   Amber Fudge                                             |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        6        |   April Abbey                                             |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        7        |   April Fudge                                             |   Not Started    |
         |
|-----------------|-----------------------------------------------------------|------------------|

|        8        |   Ash Abbey                                               |   Not Started    |
         |
```

```
|------------------|-----------------------------------------------------------|------------------|
|        9         |    Ash Fudge                                              |    Not Started   |
|                  |                                                           |                  |
|------------------|-----------------------------------------------------------|------------------|
|       10         |    Asti Abbey                                             |    Not Started   |
|                  |                                                           |                  |
|------------------|-----------------------------------------------------------|------------------|


... /* LIST SHORTENED TO REDUCE PAPER REQUIREMENTS */

|------------------|-----------------------------------------------------------|------------------|
|       126        |    Zizou Abbey                                            |    Not Started   |
|                  |                                                           |                  |
|------------------|-----------------------------------------------------------|------------------|
|       127        |    Zizou Fudge                                            |    Not Started   |
|                  |                                                           |                  |
|------------------|-----------------------------------------------------------|------------------|


Total competitors yet to start: 102

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
2

| Competitor No  |                         Competitor Name                  |  Current Status  |
|================|==========================================================|==================|


No competitors are currently on the course

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
3

| Competitor No  |                         Competitor Name                  |  Current Status  |
|================|==========================================================|==================|

No competitors have currently finished.

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
```

```
  8. Exit Program
*****************************************
6
Enter required competitor number:
1

Competitor 1 (Ace Abbey) -> Current Status: Not Started, Current Progress: 0, Start Time: N/A,
     End Time: N/A



*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
7

| Competitor No  |                          Competitor Name                          | Current Status
     |
|================|===================================================================|==================|

No competitors have currently been excluded.

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
4

Enter time file name: ../files/times.txt /* TIMES FILE LOADED INTO SYSTEM */

*****************************************
Welcome, please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
2

| Competitor No  |                          Competitor Name                          | Current Status
     |
|================|===================================================================|==================|

|      89        |   Prince Abbey                                                    |   Checkpoint 1
     |
|----------------|-------------------------------------------------------------------|------------------|

|     103        |   sienna Abbey                                                    |   Checkpoint 1
     |
|----------------|-------------------------------------------------------------------|------------------|

|     111        |   Snowy Abbey                                                     |   Checkpoint 1
     |
|----------------|-------------------------------------------------------------------|------------------|
```

```
Total competitors out on course: 3

******************************************
Welcome, please select an option:

******************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
******************************************
3

|  Competitor No   |                        Competitor Name                        |  Current Status
       |
|=================|===============================================================|==================|

|        1         |     Ace Abbey                                                 |     Finished
          |
|-----------------|---------------------------------------------------------------|------------------|

|        64        |     Lady Fudge                                                |     Finished
          |
|-----------------|---------------------------------------------------------------|------------------|

Total competitors finished: 2

******************************************
Welcome, please select an option:

******************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
******************************************
7

|  Competitor No   |                        Competitor Name                        |  Current Status
       |
|=================|===============================================================|==================|

|        3         |     Ace Fudge                                                 |     Excluded-IR
        |
|-----------------|---------------------------------------------------------------|------------------|

|        7         |     April Fudge                                               |     Excluded-MC
        |
|-----------------|---------------------------------------------------------------|------------------|

Total competitors excluded: 2

******************************************
Welcome, please select an option:

******************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
******************************************
6
Enter required competitor number:
89

Competitor 89 (Prince Abbey) -> Current Status: Checkpoint, Current Progress: 4, Start Time:
    12:30, End Time: N/A
```

```
*****************************************
Welcome , please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
6
Enter required competitor number:
1

Competitor 1 (Ace Abbey) -> Current Status: Finished , Current Progress: 11, Start Time: 09:22,
     End Time: 10:23


*****************************************
Welcome , please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
6
Enter required competitor number:
3

Competitor 3 (Ace Fudge) -> Current Status: Excluded - IR, Current Progress: 0, Start Time: N/A
     , End Time: DNF


*****************************************
Welcome , please select an option:

*****************************************
  1. Display competitors yet to start
  2. Display competitors out of courses
  3. Display finished competitors
  4. Load time log file into system
  5. Display event results list
  6. Display specific competitor status
  7. Display excluded competitors
  8. Exit Program
*****************************************
8

Exiting program..

RUN FINISHED; exit value 0; real time: 1m 18s; user: 0ms; system: 0ms
```

12 Event Manager - Results Output

This section contains the final results table produced by the "event manager" application using time log data generated by the "checkpoint manager" application. **Please find the attached printout of the results table which has been printed in landscape to ensure it can be read easily**.

Please note: The output has also been modified to reduce the amount of paper used, however no values/results have been changed.

13   Event Manager - System Activity Log

This section details the contents of the log file ("log.txt") produced by the "event manager" application detailing the activity described in the above usage example.

Listing 14: Contents of the log file produced by the event manager application.

```
LOG - EM: System queried for all entrants yet to start. - Mon Mar 11 14:46:13 2013
LOG - EM: System queried for all entrants that have started. - Mon Mar 11 14:46:16 2013
LOG - EM: System queried for all entrants that have finished. - Mon Mar 11 14:46:19 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 14:46:25 2013
LOG - EM: Entrant 1 status queried successfully. - Mon Mar 11 14:46:36 2013
LOG - EM: System queried for all entrants that have been excluded. - Mon Mar 11 14:46:42 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 14:46:55 2013
LOG - EM: System queried for all entrants that have started. - Mon Mar 11 14:47:30 2013
LOG - EM: System queried for all entrants that have finished. - Mon Mar 11 14:47:41 2013
LOG - EM: System queried for all entrants that have been excluded. - Mon Mar 11 14:47:50 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 14:48:38 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 14:51:06 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 14:54:55 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 14:54:59 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 14:56:10 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 14:56:16 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 15:00:08 2013
LOG - EM: Entrant 89 status queried successfully. - Mon Mar 11 15:00:23 2013
LOG - EM: Entrant 1 status queried successfully. - Mon Mar 11 15:00:32 2013
LOG - EM: Entrant 3 status queried successfully. - Mon Mar 11 15:00:36 2013
LOG - EM: System queried for all entrants that have finished. - Mon Mar 11 15:00:45 2013
LOG - EM: System queried for all entrants that have been excluded. - Mon Mar 11 15:00:49 2013
LOG - EM: System exiting. - Mon Mar 11 15:00:51 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 15:40:37 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 15:40:40 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 15:42:10 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 15:42:28 2013
LOG - EM: Entrant times file successfully loaded into system. - Mon Mar 11 15:43:22 2013
LOG - EM: Event results list compiled and displayed. - Mon Mar 11 15:43:27 2013
```

14 System Description

This section provides a general description of the structure and implementation of the three separate applications that form the complete "runners and riders" system.

14.1 Event Creator (C++)

The event creator application was built using an object-orientated design in C++. It contains a variety of class (.cpp) and header (.h) files that perform a variety of tasks to form the complete application.

The data model for the application consists of "container" classes (*Entrant, Event, Course, Node*) that represent each of the four core datatypes respectively. Instances of these (created by the application) are stored (some as collections) in a shared *Datastore* class that can accessed from the UI (*Menu*), and core processing (*Process*) classes. All external file input/output operations are performed by the *FileIO* class which can be acessed again from the *Menu* and *Process* classes as required.

Process
- io : FileIO
- data : Datastore *
+ Process(newData : Datastore *)
+ ~Process() : virtual
+ addEntrant(void) : void
+ createEvent(void) : void
+ getAllNodes(void) : void
+ showCourseEditor(void) : void
+ createNewCourse(void) : void
+ getSelectedCourse(void) : Course *
+ addCourseNode(currentCourse : Course *) : void
+ convertDate (input : string&) : string

Menu
- proc : Process *
- data : Datastore *
- io : FileIO
+ Menu(newData : Datastore *, newProc : Process *)
+ ~Menu() : virtual
+ showEventEditor(void) : void
+ showCourseEditor(void) : void
+ showEntrantEditor(void) : void
+ showMainMenu(void) : void
+ checkExistingEvent(void) : void

Main
+ main(argc : int, argv : char**) : int

FileIO
- arrayTokens : vector<vector<string>>
+ FileIO()
+ ~FileIO() : virtual
+ writeEntrants(entrantList : vector<Entrant *>) : void
+ writeCourses(courseList : vector<Course *>) : void
+ writeEvent(event : Event *) : void
+ getFile(fileName : string) : vector<vector<string>>

Datastore
- entrantList : vector<Entrant *>
- nodeList : vector<Node *>
- courseList : vector<Course *>
- event : Event *
+ Datastore()
+ ~Datastore() : virtual
+ addNewCourse(newCourse : Course *) : void
+ addNewNode(newNode : Node *) : void
+ addNewEntrant(newEntrant : Entrant *) : void
+ setNewEvent(newEvent : Event *) : void
+ getInCourse (courseID: char) : Course *
+ obtainNode (nodeNo: int) : Node *
+ getCourseList(void) : vector<Course *>
+ getNodeList(void) : vector<Node *>
+ getEntrantList(void) : vector<Entrant *>
+ getEvent (void) : const Event *

Node
- nodeType : string
- nodeNo : const int
+ Node()
+ Node(newNodeNo: const int, newNodeType : string)
+ ~Node() : virtual
+ setNodeType(nodeType : string) : void
+ getNodeNo(void) : const int
+ getNodeType(void) : string

Entrant
- entrantName : string
- entrantNo : const int
- courseID : char
+ Entrant()
+ Entrant(theName : string&, theEnNo : const int, theCourseID : char)
+ ~Entrant() : virtual
+ print(void) : void
+ getEntrantName(void) : string
+ getEntrantNo(void) : int
+ getCourseID(void) : char

Event
- eventName : string
- eventDate : string
- eventTime : string
+ Event()
+ Event(newEventname : string, newEventDate : string, newEventTime : string)
+ ~Event() : virtual
+ setEventTime(eventTime : string) : void
+ setEventDate(eventDate : string) : void
+ setEventName(eventName : string) : void
+ getEventTime(void) : const string
+ getEventDate(void) : const string
+ getEventName(void) : const string
+ addNewNode(newNode : Node *) : void

Course
- courseID : const char
- courseSize : int
- courseNodes : vector<Node *>
+ Course()
+ Course(theCourseID : const char)
+ ~Course() : virtual
+ addCourseNode(nodeNode : Node *) : void
+ setCourseSize(courseSize : int) : void
+ getCourseSize(void) : int
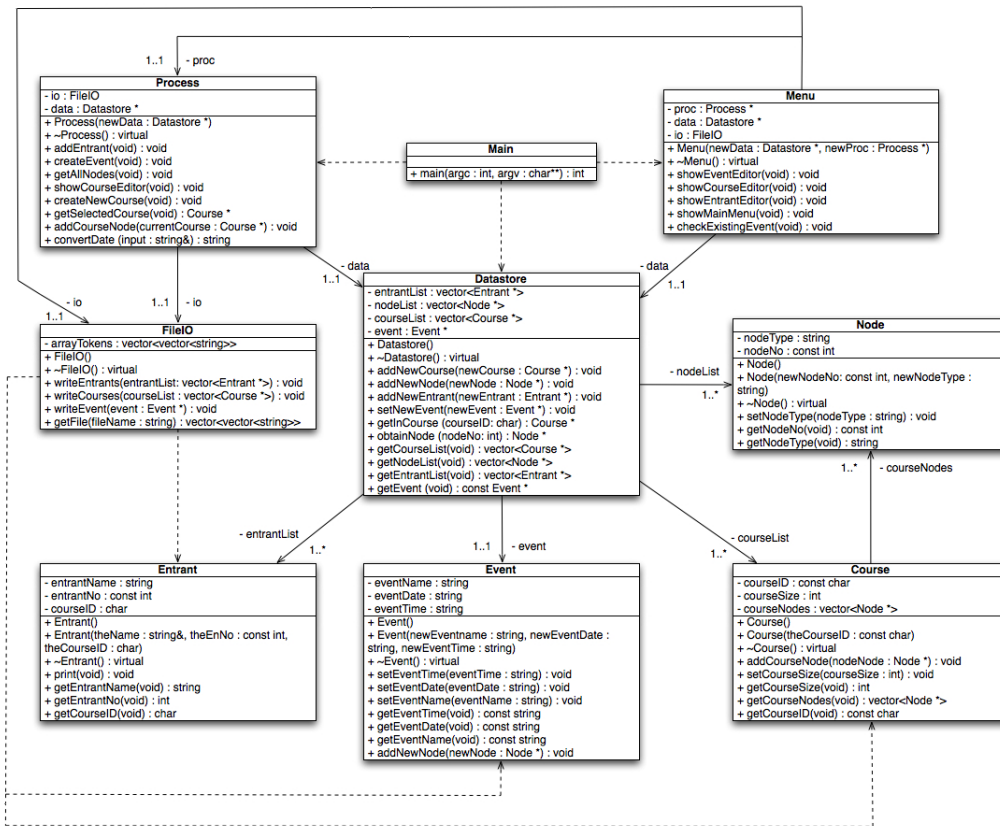+ getCourseNodes(void) : vector<Node *>
+ getCourseID(void) : const char

Figure 14: UML class diagram detailing the structural design of the event creator application.

14.2 Checkpoint Manager (Java)

This application uses a structural design not too dissimilar to that of the event creator application. Three "model" classes (*Entrant, Course, Node*) and a shared *Datastore* class form the data model for the application, and the *ProcessData* class provides the core processing/functionality for the application. A public ENUM class (*Datatype*) is used to differentiate between the data types processed by the system.

The GUI consists of a 'JFrame' *GUIFrame* class (window) and a 'JPanel' *GUIPanel* class (content) that interacts with the *Datastore*, *ProcessData* and *FileIO* classes to allow the user to interact with the system and add new checkpoint times. The majority of error checking and exception handling occurs within the *GUIPanel* class.
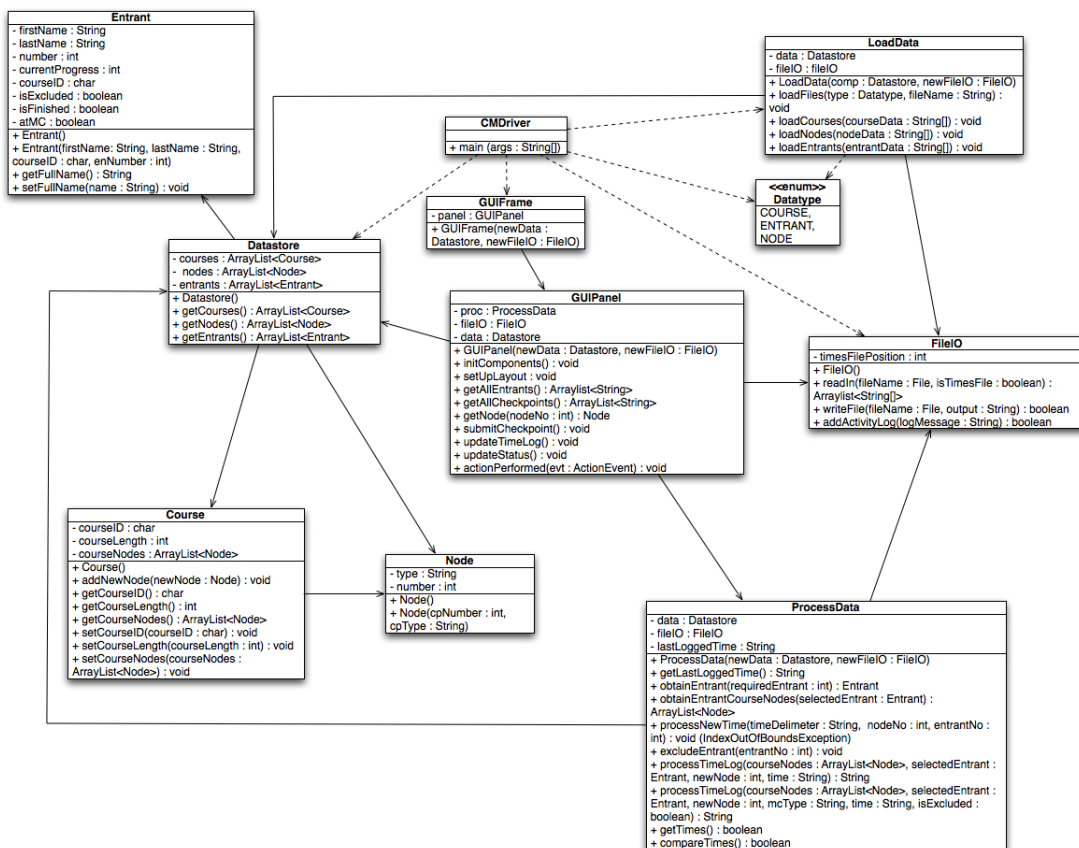

Figure 15: Checkpoint Manager GUI informing user that entrant 98 must have arrived at a MC before they can depart.

14.3 Event Manager (C)

The event manager application is almost identical to that of the 'extended mission' application produced for CS23710, with the addition of file writing and file locking operations which allow the system to log user activity to an external file.

Apart from the addition of logging functions, and the removal of the ability to manually enter in checkpoint times (now the responsibility of the checkpoint manager application) no major changes have been made to the original application.

References

[1] N. Snooke, D. Price, F. Labrosse *"CS22510 Assignment One, 2012-2013 Runners and Riders - "Out and about"* 2013: Aberystwyth University, Aberystwyth.

[2] *"How to lock files in C/C++ using fopen"* - http://stackoverflow.com/questions/7573282/ 2011: StackOverflow, StackExchange.

[3] *"Demonstrates file locking and simple file read and write operations using java.nio.channels.FileChannel"* - http://java2s.com/Code/Java/File-Input-Output/DemonstratesfilelockingandsimplefilereadandwriteusingjavaniochannelsFileChannel .htm 2011: Java2s.com, 12 Demo Source and Support.

[4] P. Prinz, T. Crawford *"C in a Nutshell"* 2008: O'Reilly Media Inc.

[5] R. Lischner *"C++ in a Nutshell"* 2009: O'Reilly Media Inc.