# CS22510 - C, C++ & Java 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 Event Creator - Source Code

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

## 2.1 Header Files

### 2.1.1 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     Menu(const Menu& orig);
26     virtual ~Menu();
27     void showEventEditor(void);
28     void showCourseEditor(void);
29     void showEntrantEditor(void);
30     void showMainMenu(void);
31     void checkExistingEvent(void);
32
33 private:
34
35     /** Pointer to shared Process class created in "main.cpp".*/
36     Process *proc;
37
38     /** Pointer to shared Datastore class created in "main.cpp".*/
39     Datastore *data;
40
41     /** Allows access to file I/O methods.*/
42     FileIO io;
43 };
44
45 #endif  /* MENU_H */
```

### 2.1.2 Process.h

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

#ifndef PROCESS_H
#define PROCESS_H

#include <vector>
#include <cstdlib>
#include "Entrant.h"
#include "Node.h"
#include "Course.h"
#include "FileIO.h"
#include "Datastore.h"
#include "Event.h"

class Process {

public:

    Process(Datastore *newData);
    Process(const Process& orig);
    virtual ~Process();
    void addEntrant(void);
    void createEvent(void);
    void getAllNodes(void);
    void showCourseEditor(void);
    void createNewCourse(void);
    Course* getSelectedCourse(void);
    void addCourseNode(Course *currentCourse);
    std::string convertDate(std::string &input);

private:

   /** Allows access to file I/O methods.*/
   FileIO io;

   /** Pointer to shared Datastore class created in "main.cpp".*/
   Datastore *data;
};

#endif  /* PROCESS_H */
```

### 2.1.3 Datastore.h

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

#ifndef DATASTORE_H
#define DATASTORE_H

#include <vector>
#include <cstdlib>
#include "Entrant.h"
#include "Node.h"
#include "Course.h"
#include "Event.h"

/**
 * Datastore class used for storing and providing access to all the of
 * shared data used throughout the application..
 */
class Datastore {
public:
    Datastore();
    virtual ~Datastore();
    std::vector<Course*> getCourseList(void);
    std::vector<Node*> getNodeList(void);
    std::vector<Entrant*> getEntrantList(void);
    Event* getEvent(void) const;
    void addNewCourse (Course* newCourse);
    void addNewNode (Node* newNode);
    void addNewEntrant (Entrant* newEntrant);
    void setNewEvent(Event* newEvent);
    Course* getInCourse (char courseID);
    Node* obtainNode (int nodeNo);

private:

    /** Vector of pointers to all Entrant objects created. */
   std::vector<Entrant*> entrantList;

   /** Vector of pointers to all Nodes objects read into the system. */
   std::vector<Node*> nodeList;

   /** Vector of pointers to all Course objects created. */
   std::vector<Course*> courseList;

   /** Pointer to Event object used to define the race event. */
   Event *event;
};

#endif  /* DATASTORE_H */
```

### 2.1.4 FileIO.h

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

#ifndef FILEIO_H
#define FILEIO_H

#include "Entrant.h"
#include "Course.h"
#include "Event.h"

class FileIO {
public:
    FileIO();
    FileIO(const FileIO& orig);
    virtual ~FileIO();
    void writeEntrants(std::vector<Entrant*> entrantList);
    void writeCourses(std::vector<Course*> courseList);
    void writeEvent(Event *event);
    std::vector<std::vector<std::string > > getFile(std::string fileName);
private:

    /**
     * Vector of vectors used to store the contents of individual line
     * that collect to form the entire file.
     */
    std::vector<std::vector<std::string > > arrayTokens;
};

#endif  /* FILEIO_H */
```

### 2.1.5 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(const Event& orig);
22     Event(std::string newEventName, std::string newEventDate, std::string newEventTime);
23     virtual ~Event();
24     void setEventTime(std::string eventTime);
25     std::string getEventTime(void) const;
26     void setEventDate(std::string eventDate);
27     std::string getEventDate(void) const;
28     void setEventName(std::string eventName);
29     std::string getEventName(void) const;
30
31 private:
32     std::string eventName; /**< The name/description of the event.*/
33     std::string eventDate; /**< The date that the event is to take place.*/
34     std::string eventTime; /**< The starting time of the event.*/
35 };
36
37 #endif  /* EVENT_H */
```

### 2.1.6 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(const std::string &theName, const int theEnNo, char theCourseID);
21     virtual ~Entrant();
22     void print(void) const;
23     std::string getEntrantName(void);
24     int getEntrantNo(void);
25     char getCourseID(void);
26 private:
27     std::string entrant_name; /**< The name of the entrant.*/
28     const int entrant_no; /**< The unique number for the entrant.*/
29     char course_id; /**< The ID that the entrant is registered for.*/
30 };
31
32 #endif  /* ENTRANT_H */
```

### 2.1.7 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 paritcular course node.
16  */
17 class Node {
18
19 public:
20     Node();
21     Node(const Node& orig);
22     Node(const int newNodeNo, std::string newNodeType);
23     virtual ~Node();
24     void setNodeType(std::string nodeType);
25     std::string getNodeType(void) const;
26     const int getNodeNo(void) const;
27
28 private:
29     const int nodeNo; /**< Unique number that represents a node.*/
30     std::string nodeType; /**< Contains the type of node.*/
31 };
32
33 #endif  /* NODE_H */
```

### 2.1.8 Course.h

```
/*
 * 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(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 */
```

## 2.2 Class Files

### 2.2.1 Main.cpp

```
1  /*
2   * File: main.cpp
3   * Description: Bootstrap loader for the application.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include <cstdlib>
10 #include "Process.h"
11 #include "Menu.h"
12
13 using namespace std;
14
15 /**
16  * Bootstrap method for the application.
17  */
18 int main(int argc, char** argv) {
19
20     /**
21      * New Datastore object created on the heap
22      * that is used throughout the program.
23      */
24     Datastore *data = new Datastore();
25
26     /**
27      * New Process object created on the heap
28      * that is used throughout the program.
29      */
30     Process *proc = new Process(data);
31
32     /**
33      * New Menu object created on the heap
34      * that will display the main menu.
35      */
36     Menu *menu = new Menu(data, proc);
37
38     //Load course nodes into system from "nodes.txt" file.
39     proc->getAllNodes();
40
41     //Display the main program menu.
42     menu->showMainMenu();
43
44    return 0;
45 }
```

### 2.2.2 Menu.cpp

```
1  /*
2   * File: Menu.cpp
3   * Description: Generates system menus to provide a means of interacting with
4   * the application.
5   * Author: Connor Luke Goddard (clg11)
6   * Date: March 2013
7   * Copyright: Aberystwyth University, Aberystwyth
8   */
9
10
11 #include <vector>
12 #include <iostream>
13 #include <limits>
14 #include <string.h>
15 #include "Menu.h"
16
17 using namespace std;
18
19 /**
20  * Constructor for Menu that allows access to Process and Datastore classes
21  * created in "main.cpp". This allows Menu to access the same data stored in
22  * Datastore as the Process class.
23  * @param newData Pointer to the shared Datastore class created in the main method.
24  * @param newProc Pointer to the shared Process class created in the main method.
25  */
26 Menu::Menu(Datastore *newData, Process *newProc) {
27
28     data = newData;
29     proc = newProc;
30
31 }
32
33 /**
34  * Destructor to be used once object is removed.
35  * Removes the objects stored on the heap.
36  */
37 Menu::~Menu() {
38
39     delete data;
40     delete proc;
41 }
42
43 /**
44  * Provides top-level interactive menu to allow user to interact with the
45  * application and utilise its functions.
46  */
47 void Menu::showMainMenu(void) {
48
49     int x;
50
51     while (x != 5) {
52
53         cout << "\n*************************************\n"
54                 << "Welcome to the Event Creator.\n"
55                 << "Please select an option:\n"
56                 << "-------------------------------------\n"
57                 << "1. Event Editor\n"
58                 << "2. Entrant Editor\n"
59                 << "3. Course Editor\n"
60                 << "4. Export ALL files.\n"
61                 << "5. Exit Program.\n"
62                 << "*************************************\n";
```

```
63
64          cin >> x;
65
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) {
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     }
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                 std::cin.ignore(std::numeric_limits<std::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                 std::cin.ignore(std::numeric_limits<std::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                     std::cin.ignore(std::numeric_limits<std::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 }
```

### 2.2.3 Process.cpp

```cpp
/*
 * File: Process.cpp
 * Description: Provides all core functionality and data processing for the
 * application.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include <vector>
#include <iostream>
#include <limits>
#include <ctime>
#include <sstream>
#include <algorithm>
#include "Process.h"

using namespace std;

/**
 * Constructor for Process that allows access to the shared Datastore class
 * created in "main.cpp".
 * @param newData Pointer to the shared Datastore class created in the main method.
 */
Process::Process(Datastore *newData) {

    data = newData;

}

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

    delete data;
}

/**
 * Prompts user for input to define an event before creating a new
 * 'Event' object and storing it's pointer in the shared Datastore class.
 */
void Process::createEvent(void) {

    string inputName, inputDate, inputTime, convertedDate;

    cout << "Please enter an event name/description: ";

    //Obtain all inputted characters including white space.
    getline(cin, inputName);

    cout << "Please enter the date of the event: (DD/MM/YYYY) ";
    getline(cin, inputDate);

    cout << "Please enter the time of the event: ";
    getline(cin, inputTime);

    //Convert inputted date string into format for writing to file.
    convertedDate = convertDate(inputDate);

    //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                 std::cin.ignore(std::numeric_limits<std::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                         std::vector<Entrant*> allEntrants = data->getEntrantList();
118
119                         //Loop through all the entrants.
120                         for (std::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     std::cin.ignore(std::numeric_limits<std::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
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 !=
                fileContents.end(); ++it) {
220
221             //Convert the first value on the line (node number) to an int.
222             int value = atoi((*it).at(0).c_str());
223
224             //Create a new Node object on the heap using the inputted information.
225             Node *tempNode = new Node(value, (*it).at(1));
226
227             //Add a pointer to the new object in the node vector stored in Datastore
228             data->addNewNode(tempNode);
229         }
230         cout << "Course nodes loaded successfully.\n" << "Loading program...\n\n";
231     }
232
233 }
234
235 /**
236  * Prompts user for input to define a course before creating a new
237  * 'Course' object and adding it's pointer to the
238  * vector of Course pointers contained in the shared Datastore class.
239  */
240 void Process::createNewCourse(void) {
241
242     char cid;
243     bool notExists = false;
244
245     if (data->getCourseList().size() > 0) {
246
247         while (!notExists) {
248
249             cid = 0;
250
251             //Check that the ID inputted by the user is a letter.
```

```
252             while (!isalpha(cid)) {
253
254                 //Prompt user for a course ID.
255                 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             std::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     std::vector<Node*> allNodes = data->getNodeList();
330 
331     //Print all the nodes to screen to provide user with a list to select from.
332     for (std::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         std::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 (std::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 }
```

### 2.2.4 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 /**
13  * Default constructor for Datastore.
14  * Sets the initial value of the 'event' pointer to NULL for
15  * error checking purposes.
16  */
17 Datastore::Datastore() {
18
19     event = NULL;
20 }
21
22 /**
23  * Destructor to be used once object is removed.
24  */
25 Datastore::~Datastore() {
26     delete event;
27 }
28
29 /**
30  * Fetches the vector of all the courses created for an event.
31  * @return A vector that contains pointers to all the Course objects created.
32  */
33 std::vector<Course*> Datastore::getCourseList(void){
34     return courseList;
35 }
36
37 /**
38  * Fetches the vector of all the nodes read in from "nodes.txt".
39  * @return A vector that contains pointers to all the Node objects.
40  */
41 std::vector<Node*> Datastore::getNodeList(void) {
42     return nodeList;
43 }
44
45 /**
46  * Fetches the vector of all the entrants created for an event.
47  * @return A vector that contains pointers to all the Entrant objects created.
48  */
49 std::vector<Entrant*> Datastore::getEntrantList(void){
50     return entrantList;
51 }
52
53 /**
54  * Fetches the Event object created to define the race event.
55  * @return A a pointer to the created Event object.
56  */
57 Event* Datastore::getEvent(void) const {
58     return event;
59 }
60
61 /**
62  * Adds a new Course object to the end of the 'courseList' vector.
```

```
63   * @param newCourse Pointer to the new Course object to be added to the vector.
64   */
65  void Datastore::addNewCourse (Course *newCourse) {
66
67      courseList.push_back(newCourse);
68
69  }
70
71  /**
72   * Adds a new Node object to the end of the 'nodeList' vector.
73   * @param newNode Pointer to the new Node object to be added to the vector.
74   */
75  void Datastore::addNewNode (Node  *newNode) {
76
77      nodeList.push_back(newNode);
78
79  }
80
81  /**
82   * Adds a new Entrant object to the end of the 'courseEntrant' vector.
83   * @param newEntrant Pointer to the new Entrant object to be added to the vector.
84   */
85  void Datastore::addNewEntrant (Entrant *newEntrant) {
86
87      entrantList.push_back(newEntrant);
88
89  }
90
91  /**
92   * Sets the 'event' pointer to a newly created Event object.
93   * @param newEvent A pointer to the new Event object created.
94   */
95  void Datastore::setNewEvent(Event *newEvent) {
96
97      this->event = newEvent;
98
99  }
100
101 /**
102  * Determines if a course with the inputted ID exists in the vector of
103  * courses ('courseList') and if so returns the pointer to that Course object.
104  * @param selectedID The course ID inputted by the user.
105  * @return Either the located course or NULL.
106  */
107 Course* Datastore::getInCourse (char selectedID) {
108
109     //Loop through the entire vector of courses.
110     for (std::vector<Course*>::iterator it = courseList.begin(); it != courseList.end();
        ++it) {
111
112         //If the ID of the current course matches the inputted ID...
113         if ((*it)->getCourseID() == selectedID) {
114
115             //... return the pointer to that Course object.
116             return (*it);
117         }
118     }
119
120     //Otherwise if no matches are found, return NULL.
121     return NULL;
122 }
123
124 /**
125  * Determines if a node with the inputted number exists in the vector of
126  * nodes ('nodeList') and if so returns the pointer to that Node object.
```

```
127  * @param nodeNo The node number inputted by the user.
128  * @return Either a pointer to the located Node object or NULL.
129  */
130 Node* Datastore::obtainNode (int nodeNo) {
131 
132     //Loop through the entire vector of courses.
133     for (std::vector<Node*>::iterator it = nodeList.begin(); it != nodeList.end(); ++it) {
134 
135         //If the number of the current node matches the inputted number...
136         if ((*it)->getNodeNo() == nodeNo) {
137 
138             //... return the pointer to that Node object.
139             return (*it);
140         }
141 
142     }
143 
144     //Otherwise if no matches are found, return NULL.
145     return NULL;
146 }
```

### 2.2.5 FileIO.cpp

```
1  /*
2   * File: FileIO.cpp
3   * Description: Provides file input/output and parsing facilities.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include <cstdlib>
10 #include <vector>
11 #include <fstream>
12 #include <iostream>
13 #include <sstream>  //for std::istringstream
14 #include <iterator> //for std::istream_iterator
15 #include "FileIO.h"
16 #include "Event.h"
17
18 using namespace std;
19
20 /**
21  * Default constructor for FileIO.
22  */
23 FileIO::FileIO() {
24 }
25
26 /**
27  * Destructor to be used once object is removed.
28  */
29 FileIO::~FileIO() {
30 }
31
32 /**
33  * Writes all the created entrants for a particular event to file using a
34  * specified format. Entrant information is obtained from the Entrant pointers vector
35  * stored within the Datastore class.
36  * @param entrantList Vector of all the Entrant pointers contained
37  * within Datastore class.
38  */
39 void FileIO::writeEntrants(vector<Entrant*> entrantList) {
40
41     //Create a new file stream.
42     ofstream myfile;
43
44     /**
45      * "Load" or create a new file with the given file name.
46      * Flags: ios::out = Output to file, ios::app = Append to existing file
47      * or create a new one.
48      */
49     myfile.open("../files/exampleentrants.txt", ios::out | ios::app);
50
51     //Loop through all the created entrants.
52     for (vector<Entrant*>::iterator it = entrantList.begin(); it != entrantList.end(); ++
           it) {
53
54         //Write the entrant details to the file using specific format.
55         myfile << (*it)->getEntrantNo() << " " << (*it)->getCourseID() << " " << (*it)->
               getEntrantName() << "\n";
56     }
57
58     //Close the file stream once completed.
59     myfile.close();
60
```

```
61  }
62
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 }
```

### 2.2.6 Event.cpp

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

#include "Event.h"

using namespace std;

/**
 * Default constructor for Event.
 */
Event::Event() {

}

/**
 * Constructor that allows the characteristics of an event to be specified.
 * @param newEventName The inputted name/description of the event.
 * @param newEventDate The inputted date of the event.
 * @param newEventTime The inputted start time of the event.
 */
Event::Event(string newEventName, string newEventDate, string newEventTime) {

    eventName = newEventName;
    eventDate = newEventDate;
    eventTime = newEventTime;

}

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

/**
 * Updates the start time of the event to an inputted value.
 * @param eventTime Recently inputted start time value.
 */
void Event::setEventTime(string eventTime) {
    this->eventTime = eventTime;
}

/**
 * Fetches the start time of the event.
 * @return The value of the 'eventTime' string variable.
 */
string Event::getEventTime(void) const {
    return eventTime;
}

/**
 * Updates the date of the event to an inputted value.
 * @param eventDate Recently inputted event date value.
 */
void Event::setEventDate(string eventDate) {
    this->eventDate = eventDate;
}
```

```
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 }
```

### 2.2.7 Entrant.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 std::string &theName, const int theEnNo, char theCourseID) :
    entrant_name(theName), entrant_no(theEnNo){

    course_id = 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 entrant_name;
}

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

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

### 2.2.8 Node.cpp

```
1  /**
2   * File: Node.cpp
3   * Description: Provides the data model for a particular course node.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include <iostream>
10 #include "Node.h"
11
12 using namespace std;
13
14 /**
15  * Constructor that allows the characteristics of a course node to be specified.
16  * Constant variable 'nodeNo' is set it's value here.
17  * @param newNodeNo The unique identifier of a particular node. (constant)
18  * @param newNodeType The course node type to be set.
19  */
20 Node::Node(const int newNodeNo, string newNodeType) : nodeNo(newNodeNo){
21
22     setNodeType(newNodeType);
23
24 }
25
26 /**
27  * Destructor to be used once object is removed.
28  */
29 Node::~Node() {
30 }
31
32
33 /**
34  * NOTE: setNodeNo() cannot be used due to 'nodeNo'
35  * being a CONSTANT value. It therefore cannot be changed
36  * once created. Making the variable MUTABLE however would allow ]
37  * it to be changed.
38  */
39
40 /**
41  * Updates the type value of the node.
42  * @param nodeType The new node type value.
43  */
44 void Node::setNodeType(string nodeType) {
45     this->nodeType = nodeType;
46 }
47
48 /**
49  * Fetches the node type of the current node.
50  * @return The type of the current node.
51  */
52 string Node::getNodeType(void) const {
53     return nodeType;
54 }
55
56 /**
57  * Fetches the unique number of the node.
58  * @return The number representing the node.
59  */
60 const int Node::getNodeNo(void) const {
61     return nodeNo;
62 }
```

### 2.2.9 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 /**
12  * Constructor that allows the constant 'courseID' variable
13  * to be specified. Also defaults the size of a course to 0.
14  * @param theCourseID The new course ID value to be set.
15  */
16 Course::Course(const char theCourseID) : courseID(theCourseID) {
17     courseSize = 0;
18 }
19
20 /**
21  * Destructor to be used once object is removed.
22  */
23 Course::~Course() {
24
25 }
26
27 /**
28  * Adds a new node to the end of the 'courseNode' vector.
29  * @param newNode Pointer to the new node to be added to the vector.
30  */
31 void Course::addCourseNode(Node *newNode) {
32
33     this->courseNodes.push_back(newNode);
34     this->setCourseSize(courseNodes.size());
35
36 }
37
38 /**
39  * Updates the total size of the course. (i.e. vector size).
40  * @param courseSize The new size value.
41  */
42 void Course::setCourseSize(int courseSize) {
43     this->courseSize = courseSize;
44 }
45
46 /**
47  * Fetches the value of 'courseSize'.
48  * @return The total number of nodes in the course.
49  */
50 int Course::getCourseSize(void) const {
51     return courseSize;
52 }
53
54 /**
55  * Fetches a vector of all the nodes in the course.
56  * @return A vector of nodes that make up the course.
57  */
58 std::vector<Node*> Course::getCourseNodes(void) const {
59     return courseNodes;
60 }
61
62 /**
```

```
63  * Fetches the ID of the course.
64  * @return The ID of the course.
65  */
66 const char Course::getCourseID(void) const {
67     return courseID;
68 }
```

# 3 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

```
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-conf
make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
rm -f -r build/Debug
rm -f dist/Debug/GNU-Linux-x86/event-creator
make[1]: Leaving directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'


CLEAN SUCCESSFUL (total time: 57ms)

"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
"/usr/bin/make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/event-creator
make[2]: Entering directory '/home/connor/Git/Endurance-Race-Tracker/Event-Creator'
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Course.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Datastore.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Entrant.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Event.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/FileIO.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Menu.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Node.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/Process.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/main.o.d
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
mkdir -p dist/Debug/GNU-Linux-x86
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)
```

# 4 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
```

# 5 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
```

# 6 Checkpoint Manager - Source Code

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

## 6.1 'Driver' Package

### 6.1.1 CMDriver.java

```
1  package aber.dcs.cs22510.clg11.driver;
2
3  import aber.dcs.cs22510.clg11.util.LoadData;
4  import aber.dcs.cs22510.clg11.gui.GUIFrame;
5  import aber.dcs.cs22510.clg11.model.Datastore;
6  import aber.dcs.cs22510.clg11.model.Datatype;
7  import aber.dcs.cs22510.clg11.util.FileIO;
8
9  /**
10  * Bootstrap class - Initialises the application.
11  *
12  * @author Connor Goddard (clg11) Copyright: Aberystwyth University,
13  * Aberystwyth.
14  */
15 public class CMDriver {
16
17     /**
18      * The main method used to initialise the main application.
19      *
20      * @param args The file names for the data files.
21      */
22     public static void main(String[] args) {
23
24         //Instantiate new Datastore object that will be shared by other classes.
25         Datastore comp = new Datastore();
26
27         //Instantiate new FileIO object to allow shared file I/O facilities.
28         FileIO fileIO = new FileIO();
29
30         //Instantiate new Datastore object that will be shared by other classes.
31         LoadData load = new LoadData(comp, fileIO);
32
33
34         //Load input files into Datastore class (nodes, tracks and courses).
35         try {
36
37             load.loadFiles(Datatype.NODE, args[0]);
38             load.loadFiles(Datatype.COURSE, args[1]);
39             load.loadFiles(Datatype.ENTRANT, args[2]);
40
41             //Once loading via textual interface is complete, display GUI.
42             new GUIFrame(comp, load, fileIO);
43
44         } catch (IndexOutOfBoundsException eX) {
45
46             System.out.println("ERROR: File parameters missing.");
47             System.out.println("Parameter format = <node path> <courses path> <entrants
                   path>");
48         }
49
50     }
51 }
```

## 6.2 'Util' Package

### 6.2.1 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;
    private String lastLoggedTime = null;

    /**
     * Allows access to the file read/write facilities.
     */
    // private FileIO fileIO = new FileIO();
    public ProcessData() {
    }

    /**
     * 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, FileIO newFileIO) {

        this.data = newData;
        this.fileIO = newFileIO;

    }

    /**
     * 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;
    }
```

```
61
62      /**
63       * Attempts to fetch a specified entrant from the internal collection of
64       * entrants.
65       *
66       * @param requiredEntrant The number of the required entrant.
67       * @return The specified Entrant object, or NULL.
68       */
69      public Entrant obtainEntrant(int requiredEntrant) {
70
71          for (Entrant e : data.getEntrants()) {
72
73              if (e.getNumber() == requiredEntrant) {
74
75                  return e;
76              }
77
78          }
79
80          return null;
81      }
82
83      /**
84       * Attempts to fetch the collection of course nodes that make up the course
85       * that a specified entrant is registered for.
86       *
87       * @param selectedEntrant The number of the required entrant.
88       * @return The collection of course nodes, or NULL.
89       */
90      public ArrayList<Node> obtainEntrantCourseNodes(Entrant selectedEntrant) {
91
92          //Loop through all the stored courses.
93          for (Course c : data.getCourses()) {
94
95              //If the current course matches the entrant's course.
96              if (c.getCourseID() == selectedEntrant.getCourseID()) {
97
98                  //Return the collection of nodes for that course.
99                  return c.getCourseNodes();
100             }
101
102         }
103
104         //Otherwise if nothing is found, return NULL.
105         return null;
106     }
107
108     /**
109      * Processes each line read in from the "times.txt" file to update the
110      * internal record of entrant's progress. This method is crucial to ensure
111      * that any time log updates created by any other running versions of the
112      * checkpoint manager are recorded in the internal entrant record within
113      * this application.
114      *
115      * @param timeDelimiter The character symbol used to represent the status of
116      * the particular time log.
117      * @param nodeNo The number of the node the time log was recorded at.
118      * @param entrantNo The number of the entrant that was recorded.
119      */
120     public void processNewTime(String timeDelimiter, int nodeNo, int entrantNo) {
121
122         //Boolean used to determine whether this particular time log has been processed.
123         boolean isUpdated = false;
124
125         //Obtain the required entrant from the internal collection of entrants.
```

```
126             Entrant currentEntrant = obtainEntrant(entrantNo);
127
128             //Check if the time log dictates that the entrant should be excluded.
129             if (timeDelimiter.equals("I") || timeDelimiter.equals("E")) {
130
131                 //If so exclude the entrant.
132                 excludeEntrant(entrantNo);
133
134                 //Log this activity in the log file ("log.txt");
135                 fileIO.addActivityLog("Entrant no: " + entrantNo + " successfully excluded.");
136
137                 /*
138                  * Otherwise if they shouldn't be excluded,
139                  * check to see if the entrant has already been excluded.
140                  */
141             } else if (!currentEntrant.getIsExcluded()) {
142
143                 ArrayList<Node> courseNodes = obtainEntrantCourseNodes(currentEntrant);
144
145                 //Loop through all the nodes that make up the course the entrant is on.
146                 for (int i = 0; i < courseNodes.size(); i++) {
147
148                     /*
149                      * Check that the current progress of the entrant < the index of
150                      * the current node in the array (to prevent nodes the entrant has
151                      * already passed being used again), and the current node equals
152                      * the node number of the current time log.
153                      */
154                     if (i > (currentEntrant.getCurrentProgress() - 1) && courseNodes.get(i).
                            getNumber() == nodeNo && !isUpdated) {
155
156                         /*
157                          * If the entrant has ARRIVED at a medical checkpoint,
158                          * their progress should not be incrememted as they are
159                          * now waiting at the MC
160                          */
161                         if (timeDelimiter.equals("A")) {
162
163                             //Just prevent this particular time log being processed any
                                    further.
164                             currentEntrant.setAtMC(true);
165                             isUpdated = true;
166
167                             /*
168                              * Otherwise, if they are DEPARTING from a MC or they
169                              * have just arrived at a normal checkpoint, then their
170                              * progress needs to be recorded and incremented.
171                              */
172                         } else {
173
174                             /*
175                              * If the read in node from time file is further along
176                              * the course than the current progress,
177                              * update the current progress.
178                              */
179                             currentEntrant.setCurrentProgress((i + 1));
180                             currentEntrant.setAtMC(false);
181
182
183                             /*
184                              * Check to see if the entrant has now completed
185                              * their course.
186                              */
187                             if (currentEntrant.getCurrentProgress() >= courseNodes.size()) {
188
```

```
189                             //Log that they have finished.
190                             currentEntrant.setIsFinished(true);
191
192                             //Log this activity in the log file ("log.txt");
193                             fileIO.addActivityLog("Entrant no: " + currentEntrant.
                                    getNumber() + " has sucessfully finished the course.");
194                         }
195
196                         isUpdated = true;
197
198                     }
199                 }
200             }
201         }
202     }
203
204     /**
205      * Updates a particular Entrant object to log the fact that they have been
206      * excluded from their race.
207      *
208      * @param entrantNo The number of the required entrant.
209      */
210     public void excludeEntrant(int entrantNo) {
211
212         for (Entrant e : data.getEntrants()) {
213
214             if (e.getNumber() == entrantNo) {
215
216                 e.setIsExcluded(true);
217             }
218
219         }
220
221     }
222
223     /**
224      * Processes a new time log submitted by the user by determining whether the
225      * entrant is on the correct path or not and updates the "times.txt" file
226      * with the resulting time log.
227      *
228      * @param courseNodes The collection of nodes that make up the course the
229      * current entrant is registered for.
230      * @param selectedEntrant The current entrant being processed.
231      * @param newNode The newly submitted node that the entrant has arrived at.
232      * @param time The inputted time of the entrant's arrival at the CP.
233      */
234     public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int
            newNode, String time) {
235
236         //Obtain the current progress of the entrant (i.e. the index of the array).
237         int nextNodeIndex = selectedEntrant.getCurrentProgress();
238         String result = null;
239         boolean timesNotLocked = true;
240         boolean logNotLocked = true;
241
242         //Check that the entrant has not already finished, or been excluded.
243         if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
244
245             result = " Entrant " + selectedEntrant.getNumber() + " successfully completed
                    their course.";
246
247         } else if (selectedEntrant.getIsExcluded()) {
248
249             result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from
                    their course.";
```

```
250
251         } else {
252
253             /*
254              * Check whether the next node in the array (i.e. the next node that the
255              * entrant SHOULD have reached) is actually the node sumbitted.
256              */
257             if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
258
259                 /*
260                  * If they do not match, the entrant has gone the wrong way.
261                  * Append this new time log with the 'I' time delimter to the
262                  * times file ("times.txt").
263                  */
264                 timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " +
                        newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
265
266                 logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                         incorrect for course. (Entrant No: " + selectedEntrant.getNumber() +
                        ")");
267
268                 result = "Entrant " + selectedEntrant.getNumber()
269                         + " has gone the INCORRECT way. (Expected node: " + courseNodes.
                                get(nextNodeIndex).getNumber() + ")";
270
271             } else {
272
273                 /*
274                  * Otherwise if they do  match, the entrant has gone the right way.
275                  * Append this new time log with the 'T' time delimter to the
276                  * times file ("times.txt").
277                  */
278                 timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "T " +
                        newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
279
280                 logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                         incorrect for course. (Entrant No: " + selectedEntrant.getNumber() +
                        ")");
281
282                 result = "Entrant " + selectedEntrant.getNumber()
283                         + " has gone the CORRECT way. (Expected node: " + courseNodes.get(
                                nextNodeIndex).getNumber() + ")";
284             }
285
286         }
287
288         if (!logNotLocked) {
289
290             result = " ERROR: System log file locked - Cannot write to file.";
291
292         }
293
294         if (!timesNotLocked) {
295
296             result = " ERROR: Times log file locked - Cannot write to file. Please try
                    again.";
297
298         }
299
300         if (!timesNotLocked && !logNotLocked) {
301
302            result = " ERROR: Cannot write to time log or log file. - Both files locked.";
303         }
304
305         return result;
```

```
306
307     }
308
309     /**
310      * Processes a new time log submitted by the user by determining whether the
311      * entrant is on the correct path or not and updates the "times.txt" file
312      * with the resulting time log (Overloaded method for processing medical
313      * checkpoints).
314      *
315      * @param courseNodes The collection of nodes that make up the course the
316      * current entrant is registered for.
317      * @param selectedEntrant The current entrant being processed.
318      * @param newNode The newly submitted node that the entrant has arrived at.
319      * @param mcType Whether the entrant was arriving or departing from the MC.
320      * @param time The inputted time of the entrant's arrival at the CP.
321      */
322     public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int
          newNode, String mcType, String time, boolean isExcluded) {
323
324         int nextNodeIndex = selectedEntrant.getCurrentProgress();
325         String result = null;
326         boolean timesNotLocked = true;
327         boolean logNotLocked = true;
328
329         //Check that the entrant has not already finished, or been excluded.
330         if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
331
332             result = " Entrant " + selectedEntrant.getNumber() + " successfully completed
                 their course.";
333
334         } else if (selectedEntrant.getIsExcluded()) {
335
336             result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from
                 their course.";
337
338         } else {
339
340             if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
341
342                 logNotLocked = fileIO.addActivityLog("Submitted checkpoint incorrect for
                     course. (Entrant No: " + selectedEntrant.getNumber() + ")");
343
344                 timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " +
                     newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
345
346                 result = "Entrant " + selectedEntrant.getNumber()
347                         + " has gone the wrong way. (Expected node: " + courseNodes.get(
                             nextNodeIndex).getNumber() + ")";
348
349             } else if (isExcluded) {
350
351                 logNotLocked = fileIO.addActivityLog("Entrant excluded for medical reasons
                     . (Entrant No: " + selectedEntrant.getNumber() + ")");
352
353                 timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "E " +
                     newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
354
355                 result = "Entrant " + selectedEntrant.getNumber()
356                         + " has been excluded for medical reasons.";
357
358             } else {
359
360                 /*
361                  * If they do match, the entrant has gone the right way.
362                  * Determine whether the entrant was arriving at, or departing
```

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

```
417
418         if (timesFile.exists()) {
419
420             ArrayList<String[]> times = fileIO.readIn(timesFile, true);
421
422             //For every time log read in from the file...
423             for (String[] newTime : times) {
424
425                 //... process this time log and update the internal record of entrants.
426                 processNewTime(newTime[0], Integer.parseInt(newTime[1]), Integer.parseInt(
                        newTime[2]));
427
428                 this.lastLoggedTime = newTime[3];
429
430             }
431
432             //Log this activity in the log file ("log.txt");
433             fileIO.addActivityLog("Time logs file loaded successfully (times.txt)");
434
435             return true;
436
437         }
438
439         return false;
440     }
441
442     /**
443      * Compares the time of the last read-in time log, with the new time being
444      * submitted to check that the user is not entering a time in the past.
445      *
446      * @param oldTimeString The last time value read-in from "times.txt".
447      * @param newTimeString The new time value being submitted by the user.
448      * @return A boolean determining if the new time value is in the past.
449      */
450     public boolean compareTimes(String oldTimeString, String newTimeString) {
451
452         SimpleDateFormat df = new SimpleDateFormat("HH:mm");
453
454         Date lastRecordedTime;
455         Date newTime;
456
457         try {
458
459             //Create new Date objects using the last logged, and new time values.
460             lastRecordedTime = df.parse(oldTimeString);
461             newTime = df.parse(newTimeString);
462
463             //Check if the new time entered is before the last read-in time.
464             if (df.format(lastRecordedTime).compareTo(df.format(newTime)) > 0) {
465
466                 //If so, then this cannot be allowed.
467
468                 //Log this activity in the log file ("log.txt");
469                 fileIO.addActivityLog("User attempted to enter new time value in the past.
                        (New time: " + df.format(newTime) + ")");
470
471                 return true;
472             }
473
474         } catch (ParseException ex) {
475             Logger.getLogger(ProcessData.class.getName()).log(Level.SEVERE, null, ex);
476         }
477
478         return false;
479     }
```

```
}
```

### 6.2.2 FileIO.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.
     * @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);


            /*
```

```
61              * Check if the current file being read in is the times file,
62              * and if so whether or not the file has been read-in previously.
63              */
64             if (isTimesFile && this.timesFilePosition > 0) {
65
66                 /*
67                  * Read down to the last logged line read-in file
68                  * without processing any of the lines (used to "skip" down
69                  * to any lines that could have been added after the last time
70                  * the file was read in by this application).
71                  */
72                 for (int i = 0; i < this.timesFilePosition; i++) {
73                     bufferedReader.readLine();
74                 }
75             }
76
77             //Initialise local variable used to store the current line being read in
78             String line;
79
80             //While there are still lines to read in from the file (i.e. read in every
                   line in the file)
81             while ((line = bufferedReader.readLine()) != null) {
82
83                 //As there is multiple data on each line, split the values up.
84                 String[] details = line.split(" ");
85
86                 //Add these broken down values to the larger collection of lines.
87                 values.add(details);
88
89                 /*
90                  * If the current file being read in is "times.txt", updated the
91                  * last line to be read in by the system. (Used for when the
92                  * file is re-"readin" by the system).
93                  */
94                 if (isTimesFile) {
95                     timesFilePosition++;
96                 }
97             }
98
99             //Once completed, safely close the file reader
100            bufferedReader.close();
101
102            return values;
103
104            //If any IO exceptions occur...
105        } catch (IOException iOE) {
106
107            return null;
108        }
109
110    }
111
112    /**
113     * Writes output data to specified files, as these files are shared, file
114     * locking has to be used to prevent corruption of data/files.
115     *
116     * @param writeFile The file that is to be written to.
117     * @param output The output data string.
118     */
119    public boolean writeFile(File writeFile, String output) {
120
121        FileOutputStream fos;
122        FileLock fl = null;
123
124        try {
```

```
125
126             //If the file does not exist, create a new file.
127             if (!writeFile.exists()) {
128                 writeFile.createNewFile();
129             }
130
131             //Create a new output stream that will append to the file.
132             fos = new FileOutputStream(writeFile.getAbsoluteFile(), true);
133
134             //Attempt to lock the file to allow the data to be written.
135             try {
136
137                 fl = fos.getChannel().tryLock();
138
139             } catch (OverlappingFileLockException flE) {
140
141                 /*
142                  * If there is already a process within the same JVM locking
143                  * the file, inform the user.
144                  */
145                 System.out.println("ERROR: File <" + writeFile.getName() + "> cannot be
                        accessed. File lock still in place.");
146             }
147
148             //Check if the lock was successfull.
149             if (fl != null) {
150
151                 try (FileWriter fw = new FileWriter(fos.getFD())) {
152
153                     fw.write(output);
154
155                     //Once the data has been successfully written, release the lock.
156                     fl.release();
157                 }
158
159                 return true;
160
161             }
162
163         } catch (IOException e) {
164
165         }
166
167         return false;
168     }
169
170     /**
171      * Adds a new log message to the "logs.txt" file. Called when a major
172      * activity occurs in the application.
173      *
174      * @param logMessage Message describing the activity.
175      */
176     public boolean addActivityLog(String logMessage) {
177
178         //Obtain the current date/time and format it for use in the log file.
179         DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
180         Calendar cal = Calendar.getInstance();
181
182         //Build the log message using predefined output template.
183         String logOutput = "LOG - CM: " + logMessage + " - " + dateFormat.format(cal.
                getTime()) + "\n";
184
185         //Write the log message to the log file.
186         return writeFile(new File("../files/log.txt"), logOutput);
187
```

```
188     }
189 }
```

### 6.2.3 LoadData.java

```
1  package aber.dcs.cs22510.clg11.util;
2
3  import aber.dcs.cs22510.clg11.model.*;
4  import java.io.File;
5  import java.util.ArrayList;
6
7  /**
8   * Responsible for loading crucial, preliminary data files into the system using
9   * a textual interface before the GUI is loaded.
10  *
11  * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
12  * Aberystwyth.
13  */
14 public class LoadData {
15
16     private Datastore data;
17     private FileIO fileIO;
18
19     /**
20      * Constructor to instantiate a new LoadData. Takes the shared data store
21      * object created in {@link aber.dcs.cs22510.clg11.driver.CMDriver} as a
22      * parameter to allow accessed to the lists of nodes/entrants/courses loaded
23      * in.
24      *
25      * @param newData Datastore object created in CMDriver.
26      */
27     public LoadData(Datastore comp, FileIO newFileIO) {
28
29         this.data = comp;
30         this.fileIO = newFileIO;
31
32     }
33
34     /**
35      * Prompts user for the file path of a specified file before attempting to
36      * load the data into it's respective data collection.
37      *
38      * @param type ENUM denoting the type of data file (Node, Course or
39      * Entrant).
40      * @param fileName The path of the file to be loaded.
41      */
42     public void loadFiles(Datatype type, String fileName) {
43
44         File f = new File(fileName);
45         ArrayList<String[]> readValues;
46
47         //Check if the file exists.
48         if (!f.exists()) {
49
50             //If it does not exist, inform the user.
51             if (type == Datatype.NODE) {
52
53                 System.out.println("ERROR: Nodes file <" + fileName + "> does not exist.")
                       ;
54
55             } else if (type == Datatype.COURSE) {
56
57                 System.out.println("ERROR: Courses file <" + fileName + "> does not exist.
                       ");
```

```
58
59              } else {
60
61                  System.out.println("ERROR: Entrants file <" + fileName + "> does not exist
                        .");
62
63              }
64
65              System.out.println("Parameter format = <node path> <courses path> <entrants
                    path>");
66              System.exit(0);
67          }
68
69
70          //If the file does exist, read in the data from the file.
71          readValues = fileIO.readIn(f, false);
72
73          //Determine the type of data being loaded.
74          if (type.equals(Datatype.NODE)) {
75
76              for (String[] newItem : readValues) {
77
78                  //Load all the nodes from the read-in data.
79                  loadNodes(newItem);
80
81              }
82
83              displayNodes();
84
85              //Log this activity in the log file ("log.txt");
86              fileIO.addActivityLog("Nodes file loaded successfully (nodes.txt)");
87
88          } else if (type.equals(Datatype.COURSE)) {
89
90              for (String[] newItem : readValues) {
91
92                  loadCourses(newItem);
93
94              }
95
96              displayCourses();
97              fileIO.addActivityLog("Courses file loaded successfully (courses.txt)");
98
99          } else {
100
101             for (String[] newItem : readValues) {
102
103                 loadEntrants(newItem);
104
105             }
106
107             displayEntrants();
108             fileIO.addActivityLog("Entrants file loaded successfully (courses.txt)");
109         }
110
111     }
112
113     /**
114      * Parses the data read-in from the "courses.txt" file and creates a new
115      * {@link aber.dcs.cs22510.clg11.model.Course} object populated with the
116      * read-in characteristics. This new Course object is then added to the
117      * internal collection of Courses.
118      *
119      * @param courseData Collection of all course characteristics data read in
120      * from "courses.txt".
```

```
121      */
122     public void loadCourses(String[] courseData) {
123 
124         try {
125 
126             //Create a new empty Course object.
127             Course newCourse = new Course();
128 
129             //Set the course ID to the first element in the course data array.
130             newCourse.setCourseID(courseData[0].charAt(0));
131 
132             //Set the course length to the second element in the course data array.
133             newCourse.setCourseLength(Integer.parseInt(courseData[1]));
134 
135             //Loop through the REST (i=2) of the "read-in" course data array..
136             for (int i = 2; i < (courseData.length); i++) {
137 
138                 //Loop through all the course nodes stored internally.
139                 for (Node n : data.getNodes()) {
140 
141                     int origNodeNo = n.getNumber();
142 
143                     //Obtain the node number currently being parsed from the read in data.
144                     int courseNodeNo = Integer.parseInt(courseData[i]);
145 
146                     /*
147                      * If the node number read-in from file matches the current node,
148                      * and the node is NOT a junction, add this node to the collection
149                      * of nodes within the new Course object.
150                      */
151                     if (origNodeNo == courseNodeNo && (n.getType().equals("CP") || n.
                            getType().equals("MC"))) {
152 
153                         newCourse.addNewNode(n);
154                     }
155                 }
156 
157             }
158 
159             /*
160              * Once the new Course object has been populated with data,
161              * add it to the collection of courses in Datastore.
162              */
163             data.getCourses().add(newCourse);
164 
165             //If an error occurs...
166         } catch (Exception e) {
167 
168             //... log the error in the "log.txt" file.
169             fileIO.addActivityLog("ERROR - Cannot create new course object (" + courseData
                    [0] + ")");
170         }
171 
172     }
173 
174     /**
175      * Parses the data read-in from the "nodes.txt" file and creates a new
176      * {@link aber.dcs.cs22510.clg11.model.Node} object populated with the
177      * read-in characteristics. This new Node object is then added to the
178      * internal collection of Nodes.
179      *
180      * @param nodeData Collection of all node characteristics data read in from
181      * "nodes.txt".
182      */
183     public void loadNodes(String[] nodeData) {
```

```
184
185             try {
186
187                 Node newNode = new Node();
188
189                 newNode.setNumber(Integer.parseInt(nodeData[0]));
190                 newNode.setType(nodeData[1]);
191
192                 data.getNodes().add(newNode);
193
194             } catch (Exception e) {
195
196                 fileIO.addActivityLog("ERROR - Cannot create new node object (" + Integer.
                        parseInt(nodeData[0]) + " / " + nodeData[1] + ")");
197             }
198
199         }
200
201         /**
202          * Parses the data read-in from the "entrants.txt" file and creates a new
203          * {@link aber.dcs.cs22510.clg11.model.Entrant} object populated with the
204          * read-in characteristics. This new Entrant object is then added to the
205          * internal collection of Entrants.
206          *
207          * @param entrantData Collection of all node characteristics data read in
208          * from "nodes.txt".
209          */
210         public void loadEntrants(String[] entrantData) {
211
212             try {
213
214                 Entrant newEntrant = new Entrant();
215
216                 newEntrant.setNumber(Integer.parseInt(entrantData[0]));
217                 newEntrant.setCourseID(entrantData[1].charAt(0));
218                 newEntrant.setFirstName(entrantData[2]);
219                 newEntrant.setLastName(entrantData[3]);
220
221                 data.getEntrants().add(newEntrant);
222
223             } catch (Exception e) {
224
225                 fileIO.addActivityLog("ERROR - Cannot create new entrant object (" + Integer.
                        parseInt(entrantData[0]) + " / " + entrantData[1] + " / " + entrantData[2]
                         + " " + entrantData[3] + ")");
226             }
227
228         }
229
230         /**
231          * Displays all {@link aber.dcs.cs22510.clg11.model.Courses} objects loaded
232          * into the internal collection of courses to screen.
233          */
234         public void displayCourses() {
235
236             for (int i = 0; i < data.getCourses().size(); i++) {
237
238                 System.out.println(data.getCourses().get(i).getCourseID());
239                 System.out.println(data.getCourses().get(i).getCourseLength());
240                 System.out.println("Course Nodes:");
241
242                 for (Node n : data.getCourses().get(i).getCourseNodes()) {
243
244                     System.out.println(n.getNumber() + " - " + n.getType());
245
```

```
246             }
247
248             System.out.println("*****************\n");
249         }
250     }
251
252     /**
253      * Displays all {@link aber.dcs.cs22510.clg11.model.Entrants} objects loaded
254      * into the internal collection of courses to screen.
255      */
256     public void displayNodes() {
257
258         for (int i = 0; i < data.getNodes().size(); i++) {
259
260             System.out.println(data.getNodes().get(i).getNumber() + " - " + data.getNodes
                    ().get(i).getType());
261             System.out.println("*****************\n");
262         }
263     }
264
265     /**
266      * Displays all {@link aber.dcs.cs22510.clg11.model.Entrants} objects loaded
267      * into the internal collection of courses to screen.
268      */
269     public void displayEntrants() {
270
271         for (int i = 0; i < data.getEntrants().size(); i++) {
272
273             System.out.println(data.getEntrants().get(i).getNumber() + " - " + data.
                    getEntrants().get(i).getCourseID() + " - " + data.getEntrants().get(i).
                    getFullName());
274             System.out.println("*****************\n");
275         }
276     }
277 }
```

## 6.3 'Model' Package

### 6.3.1 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;
    }

}
```

### 6.3.2 Entrant.java

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

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


    private String firstName;
    private String lastName;

    /** Entrant number used for tracking of entrant. */
    private int number;

    /** The current progress of the entrant along their registered course. */
    private int currentProgress;

    /** The ID character of the course the entrant is registered for. */
    private char courseID;

    /** Defines if the entrant is excluded or not. */
    private boolean isExcluded = false;

    /** Defines if the entrant has finished or not. */
    private boolean isFinished = false;

    /** Defines if the entrant is currently at a medical checkpoint. */
    private boolean atMC = false;

    /**
     * Default constructor for an Entrant.
     * Sets the current progress to 0 as a new entrant will
     * not have started the race.
     */
    public Entrant() {

        this.currentProgress = 0;

    }

    /**
     * Constructor for an Entrant that allows their characteristics to be set upon
     * instantiation.
     * @param firstName The first name of the new entrant.
     * @param lastName The last name of the new entrant.
     * @param courseID The ID of the course the new entrant is registered for.
     * @param enNumber The race number of the new entrant.
     */
    public Entrant(String firstName, String lastName, char courseID, int enNumber) {

        this.firstName = firstName;
        this.lastName = lastName;
        this.courseID = courseID;
        this.number = enNumber;
        this.currentProgress = 0;

    }

```

```
63      /**
64       * Fetches the full name (both first and last) name of the entrant.
65       * @return The full name of the entrant.
66       */
67      public String getFullName() {
68          return getFirstName() + " " + getLastName();
69      }
70
71      /**
72       * Sets the full name of the entrant by splitting the full
73       * name on a space and setting the separate first, and last names.
74       * @param name The inputted full name to be set.
75       */
76      public void setFullName(String name) {
77
78          //Split the inputted name by a space.
79          String[] tempName = name.split(" ");
80          this.setFirstName(tempName[0]);
81          this.setLastName(tempName[1]);
82      }
83
84      /**
85       * Returns the race number of the entrant.
86       * @return The race number of the entrant.
87       */
88      public int getNumber() {
89          return number;
90      }
91
92      /**
93       * Sets the race number of the entrant.
94       * @param number The race number to be set.
95       */
96      public void setNumber(int number) {
97          this.number = number;
98      }
99
100     /**
101      * Fetches the current progress of the entrant along their course.
102      * @return The current progress of the entrant on their course.
103      */
104     public int getCurrentProgress() {
105         return currentProgress;
106     }
107
108     /**
109      * Updates the current progress of the entrant along their course.
110      * @param currentProgress The incremented progress of the entrant.
111      */
112     public void setCurrentProgress(int currentProgress) {
113         this.currentProgress = currentProgress;
114     }
115
116     /**
117      * Returns the first name of the entrant.
118      * @return The first name of the entrant.
119      */
120     public String getFirstName() {
121         return firstName;
122     }
123
124     /**
125      * Sets the first name only of the entrant.
126      * @param firstName The first name of the entrant to be set.
127      */
```

```
128     public void setFirstName(String firstName) {
129         this.firstName = firstName;
130     }
131 
132     /**
133      * Returns the last name of the entrant.
134      * @return The last name of the entrant.
135      */
136     public String getLastName() {
137         return lastName;
138     }
139 
140     /**
141      * Sets the last name only of the entrant.
142      * @param lastName The last name of the entrant to be set.
143      */
144     public void setLastName(String lastName) {
145         this.lastName = lastName;
146     }
147 
148     /**
149      * Fetches the course ID that the entrant is registered for.
150      * @return The ID of the registered course.
151      */
152     public char getCourseID() {
153         return courseID;
154     }
155 
156     /**
157      * Sets the course ID of the entrant.
158      * @param courseID The ID of the course that the entrant is registered for.
159      */
160     public void setCourseID(char courseID) {
161         this.courseID = courseID;
162     }
163 
164     /**
165      * Returns whether the entrant is excluded or not.
166      * @return Boolean determining if the entrant is excluded.
167      */
168     public boolean getIsExcluded() {
169         return isExcluded;
170     }
171 
172     /**
173      * Sets whether or not the entrant is excluded.
174      * @param isExcluded Whether the entrant is excluded or not.
175      */
176     public void setIsExcluded(boolean isExcluded) {
177         this.isExcluded = isExcluded;
178     }
179 
180     /**
181      * Returns whether the entrant has finished their race or not.
182      * @return Boolean determining if the entrant has finished.
183      */
184     public boolean getIsFinished() {
185         return isFinished;
186     }
187 
188     /**
189      * Sets whether or not the entrant has finished their race or not.
190      * @param isFinshed Whether the entrant is has finished
191      */
192     public void setIsFinished(boolean isFinished) {
```

```
193             this.isFinished = isFinished;
194         }
195 
196         /**
197          * Returns if the entrant is currently at a medical checkpoint.
198          * @return Boolean determining if the entrant is at an MC.
199          */
200         public boolean getAtMC() {
201             return atMC;
202         }
203 
204         /**
205          * Sets if an entrant is at a medical checkpoint.
206          * @param atMC Whether the entrant is currently at an MC or not.
207          */
208         public void setAtMC(boolean atMC) {
209             this.atMC = atMC;
210         }
211 
212 }
```

### 6.3.3 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) {
```

```
63          this.courseLength = courseLength;
64      }
65
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  }
```

### 6.3.4 Node.java

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

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

    /** Type of the node. (CP, MX, JN) */
    private String type;

    /** The unique node number. */
    private int number;

    /**
     * Default constructor for an Entrant.
     */
    public Node() {

    }

    /**
     * Constructor for a Node that allows their characteristics to be set upon
     * instantiation.
     *
     * @param cpNumber The new node number to be set.
     * @param cpType The node type of the new node
     */
    public Node(int cpNumber, String cpType) {

        this.number = cpNumber;
        this.type = cpType;

    }

    /**
     * Returns the node type of the current node.
     * @return The type of the current node.
     */
    public String getType() {
        return type;
    }

    /**
     * Set the current node type.
     * @param type The new node type to be set.
     */
    public void setType(String type) {
        this.type = type;
    }

    /**
     * Returns the ID number of the node.
     * @return The number of the current node.
     */
    public int getNumber() {
        return number;
    }

```

```
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  }
```

### 6.3.5 Datatype.java

```
1   package aber.dcs.cs22510.clg11.model;
2
3   /**
4    * Public enumeration used to define the type of data that is to be read into
5    * the system to allow the correct file read/parse methods to be used for the
6    * type of data being read in.
7    *
8    * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
9    * Aberystwyth.
10   */
11  public enum Datatype {
12
13      COURSE,
14      ENTRANT,
15      NODE
16  };
```

## 6.4 'GUI' Package

### 6.4.1 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.
     * @param newLoad LoadData class created in main method.
     */
    public GUIFrame(Datastore newData, LoadData newLoad, FileIO newFileIO) {

        //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, newLoad, newFileIO);

        //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.
```

```
59          this.setVisible(true);
60      }
61  }
```

### 6.4.2 GUIPanel.java

```
1   package aber.dcs.cs22510.clg11.gui;
2
3   import aber.dcs.cs22510.clg11.model.Datastore;
4   import aber.dcs.cs22510.clg11.model.Entrant;
5   import aber.dcs.cs22510.clg11.model.Node;
6   import aber.dcs.cs22510.clg11.util.FileIO;
7   import aber.dcs.cs22510.clg11.util.LoadData;
8   import aber.dcs.cs22510.clg11.util.ProcessData;
9   import java.awt.Dimension;
10  import java.awt.event.ActionEvent;
11  import java.awt.event.ActionListener;
12  import java.text.SimpleDateFormat;
13  import java.util.ArrayList;
14  import java.util.Arrays;
15  import java.util.Calendar;
16  import javax.swing.*;
17  import javax.swing.border.BevelBorder;
18
19  /**
20   * Contains the GUI elements accessed by the user to interact with the
21   * application. Allows the user to select entrants and nodes. Enter a new time
22   * (or use system time) and then submit this new time to the log file. Any
23   * problems that occur will be displayed to the user.
24   *
25   * @author Connor Goddard (clg11) Copyright: Aberystwyth University,
26   * Aberystwyth.
27   */
28  public class GUIPanel extends JPanel implements ActionListener {
29
30      /**
31       * Buttons that represent system-wide operations.
32       */
33      private JButton submitTime, setCurrentTime;
34      private JLabel nodeTitle, entrantTitle, mcTypeTitle, timeTitle, statusBar;
35      /**
36       * The layout manager used by the panel.
37       */
38      private SpringLayout layout = new SpringLayout();
39      /**
40       * Drop-down selected boxes used to select entrants and nodes.
41       */
42      private JComboBox<String> entrantList;
43      private JComboBox<String> nodeList;
44      /**
45       * Determines whether an entrant is arriving or leaving medical checkpoint.
46       */
47      private JComboBox<String> mcTypeList;
48      private String[] mcArriveDepart = {"Arriving", "Departing"};
49      /**
50       * Spinner used to allow the user to select a time value.
51       */
52      private JSpinner timeSpinner;
53      private SpinnerDateModel sm;
54      private Datastore data;
55      private LoadData load;
56
57
58      private JCheckBox mcExclude;
```

```
59      /**
60       * Enables the GUI to access the methods used for processing times.
61       */
62      private ProcessData proc;
63      private FileIO fileIO;
64
65      /**
66       * Constructor to instantiate a new GUIPanel. Takes the two classes passed
67       * from GUIFrame as parameters to allow panel to acccess shared data store
68       * and loading facilities.
69       *
70       * @param newData Datastore object passed down from GUIFrame.
71       * @param newLoad LoadData object passed down from GUIFrame.
72       */
73      public GUIPanel(Datastore newData, LoadData newLoad, FileIO newFileIO) {
74
75          this.data = newData;
76          this.load = newLoad;
77          this.fileIO = newFileIO;
78
79          //Set the size of the panel
80          this.setPreferredSize(new Dimension(500, 250));
81
82          //Set the bespoke layout manager.
83          this.setLayout(layout);
84
85          //Initialise and add all of the panel GUI components.
86          initComponents();
87
88          setUpLayout();
89
90      }
91
92      /**
93       * Initialises the panel components (including linking components to
94       * listeners where required) before adding the components to the panel.
95       */
96      public void initComponents() {
97
98          String[] comboValues;
99
100         /*
101          * Instantiate new ProcessData class to allow access to data processing
102          * facilties.
103          */
104         proc = new ProcessData(data, fileIO);
105
106         //Create new instance of JLabel with specified display text
107         entrantTitle = new JLabel("Entrant List:");
108         nodeTitle = new JLabel("Checkpoint List:");
109         mcTypeTitle = new JLabel("Medical CP Type:");
110         timeTitle = new JLabel("Log Time:");
111
112         statusBar = new JLabel("Welcome to the Checkpoint Manager.");
113         statusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
114         statusBar.setPreferredSize(new Dimension(500, 20));
115
116         //Load all entrant names into entrant drop-down GUI box component.
117         comboValues = Arrays.copyOf(getAllEntrants().toArray(), getAllEntrants().toArray()
                .length, String[].class);
118
119         entrantList = new JComboBox<>(comboValues);
120         entrantList.setSelectedIndex(0);
121
122         //Load all node numbers into node drop-down GUI box component.
```

```
123         comboValues = Arrays.copyOf(getAllCheckpoints().toArray(), getAllCheckpoints().
                toArray().length, String[].class);
124
125         nodeList = new JComboBox<>(comboValues);
126         nodeList.setSelectedIndex(0);
127
128         //Add local action listener (Required for determining a MC).
129         nodeList.addActionListener(this);
130
131         //Load the MC "arrive/depart" options into drop-down GUI box.
132         mcTypeList = new JComboBox<>(mcArriveDepart);
133         mcTypeList.setSelectedIndex(0);
134         mcTypeList.setEnabled(false);
135
136
137         //Create new instance of JButton with specified button text
138         submitTime = new JButton("Submit Checkpoint Time");
139         submitTime.addActionListener(this);
140
141         setCurrentTime = new JButton("Set to Current Time");
142         setCurrentTime.addActionListener(this);
143
144
145         //Create new JSpinner model that will access the current system time.
146         sm = new SpinnerDateModel();
147         sm.setCalendarField(Calendar.MINUTE);
148
149         //Create a new Spinner object and set the above model to it.
150         timeSpinner = new JSpinner();
151         timeSpinner.setModel(sm);
152
153         //Set the time format to be diplayed in the JSpinner.
154         JSpinner.DateEditor de = new JSpinner.DateEditor(timeSpinner, "HH:mm");
155         timeSpinner.setEditor(de);
156
157          mcExclude = new JCheckBox("Exclude Entrant");
158          mcExclude.setSelected(false);
159          mcExclude.setEnabled(false);
160
161         //Add all the components to the GUI panel.
162         this.add(nodeTitle);
163         this.add(entrantTitle);
164         this.add(mcTypeTitle);
165         this.add(timeTitle);
166         this.add(statusBar);
167         this.add(timeSpinner);
168         this.add(mcExclude);
169         this.add(entrantList);
170         this.add(nodeList);
171         this.add(mcTypeList);
172         this.add(submitTime);
173         this.add(setCurrentTime);
174
175     }
176
177     /**
178      * Sets up the 'SpringLayout' layout manager to organise all components on
179      * the panel.
180      */
181     public void setUpLayout() {
182
183         //Set the NORTH edge of the button to be 5 pixels down from the NORTH edge of the
                panel
184         layout.putConstraint(SpringLayout.NORTH, nodeTitle, 10, SpringLayout.NORTH, this);
185         layout.putConstraint(SpringLayout.WEST, nodeTitle, 10, SpringLayout.WEST, this);
```

```
186
187         layout.putConstraint(SpringLayout.NORTH, nodeList, 10, SpringLayout.NORTH, this);
188         layout.putConstraint(SpringLayout.WEST, nodeList, 10, SpringLayout.EAST, nodeTitle
                );
189
190         layout.putConstraint(SpringLayout.NORTH, entrantTitle, 10, SpringLayout.SOUTH,
                nodeTitle);
191         layout.putConstraint(SpringLayout.WEST, entrantTitle, 10, SpringLayout.WEST, this)
                ;
192
193         layout.putConstraint(SpringLayout.NORTH, entrantList, 10, SpringLayout.SOUTH,
                nodeTitle);
194         layout.putConstraint(SpringLayout.WEST, entrantList, 10, SpringLayout.EAST,
                entrantTitle);
195
196         layout.putConstraint(SpringLayout.NORTH, mcTypeTitle, 10, SpringLayout.SOUTH,
                entrantTitle);
197         layout.putConstraint(SpringLayout.WEST, mcTypeTitle, 10, SpringLayout.WEST, this);
198
199         layout.putConstraint(SpringLayout.NORTH, mcTypeList, 10, SpringLayout.SOUTH,
                entrantTitle);
200         layout.putConstraint(SpringLayout.WEST, mcTypeList, 10, SpringLayout.EAST,
                mcTypeTitle);
201
202         layout.putConstraint(SpringLayout.NORTH, mcExclude, 10, SpringLayout.SOUTH,
                mcTypeTitle);
203         layout.putConstraint(SpringLayout.WEST, mcExclude, 10, SpringLayout.WEST, this);
204
205         layout.putConstraint(SpringLayout.NORTH, timeTitle, 10, SpringLayout.SOUTH,
                mcExclude);
206         layout.putConstraint(SpringLayout.WEST, timeTitle, 10, SpringLayout.WEST, this);
207
208         layout.putConstraint(SpringLayout.NORTH, timeSpinner, 10, SpringLayout.SOUTH,
                mcExclude);
209         layout.putConstraint(SpringLayout.WEST, timeSpinner, 10, SpringLayout.EAST,
                timeTitle);
210
211         layout.putConstraint(SpringLayout.NORTH, setCurrentTime, 10, SpringLayout.SOUTH,
                timeTitle);
212         layout.putConstraint(SpringLayout.WEST, setCurrentTime, 10, SpringLayout.WEST,
                this);
213
214         layout.putConstraint(SpringLayout.NORTH, submitTime, 10, SpringLayout.SOUTH,
                setCurrentTime);
215         layout.putConstraint(SpringLayout.WEST, submitTime, 10, SpringLayout.WEST, this);
216
217         layout.putConstraint(SpringLayout.SOUTH, statusBar, 0, SpringLayout.SOUTH, this);
218     }
219
220     /**
221      * Obtains a list of all the entrant names to populate selection box.
222      * Accesses them from the array list of entrants contained within
223      * {@link aber.dcs.cs22510.clg11.model.Datastore}.
224      *
225      * @return Arraylist of all the entrant's names.
226      */
227     public ArrayList<String> getAllEntrants() {
228
229         ArrayList<String> entrantList = new ArrayList<>();
230
231         for (Entrant e : data.getEntrants()) {
232
233             entrantList.add(e.getFullName());
234
235         }
```

```
236
237         return entrantList;
238
239     }
240
241     /**
242      * Obtains a list of all the checkpoints ONLY to populate the CP selection
243      * box. Accesses them from the array list of nodes contained within
244      * {@link aber.dcs.cs22510.clg11.model.Datastore}.
245      *
246      * @return Arraylist of all the nodes that are CHECKPOINTS.
247      */
248     public ArrayList<String> getAllCheckpoints() {
249
250         ArrayList<String> checkpointList = new ArrayList<>();
251
252         //Loop through all the nodes.
253         for (Node cp : data.getNodes()) {
254
255             //If the current node is a checkpoint, and not a junction, add it.
256             if (cp.getType().equals("CP") || cp.getType().equals("MC")) {
257
258                 checkpointList.add(Integer.toString(cp.getNumber()));
259
260             }
261
262         }
263
264         return checkpointList;
265
266     }
267
268     /**
269      * Attempts to fetch a specific node denoted by the node number selected
270      * from the drop-down GUI box. If such a node cannot be found, NULL us
271      * returned.
272      *
273      * @param nodeNo The number of the selected node.
274      * @return The located node or NULL.
275      */
276     public Node getNode(int nodeNo) {
277
278         for (Node n : data.getNodes()) {
279
280             if (n.getNumber() == nodeNo) {
281
282                 return n;
283             }
284         }
285
286         return null;
287
288     }
289
290     /**
291      * Submits a new time log based on user input within the GUI and determines
292      * if an time file currently exists, or if a new one has to be created.
293      */
294     public void submitCheckpoint() {
295
296
297         //Display question dialog
298         int shouldProcess = JOptionPane.YES_OPTION;
299
```

```
300             shouldProcess = JOptionPane.showConfirmDialog(null, "Are you sure you wish to
                    submit this time log?",
301                     "CM Manager | Submit Time Log", JOptionPane.YES_NO_OPTION);
302
303             //If user selects "yes"
304             if (shouldProcess == JOptionPane.YES_OPTION) {
305
306                 //Create a formatter for the time value entered by the user.
307                 SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
308
309                 String newTimeValue = sdf.format(timeSpinner.getValue());
310
311                 //Check if a times file currently exists.
312                 if (proc.getTimes()) {
313
314                     updateStatus(" Times file loaded successfully.");
315
316                     //Check to see if the time entered by the user is in the past.
317                     if (proc.compareTimes(proc.getLastLoggedTime(), newTimeValue)) {
318
319                         updateStatus(" ERROR: Time entered is before last recorded time.
                                Please try again.");
320
321                     } else {
322
323                         /*
324                          * Re-read in the "times" file to allow any new times logged by other
325                          * running versions of the checkpoint manager to update the
                                information
326                          * contained within this version of the CM.
327                          */
328                         updateTimeLog();
329                     }
330
331                 //If a time file does not currently exist, create a new one.
332                 } else {
333
334                     updateStatus(" ALERT: Times file (times.txt) not found. Creating new time
                            log file.");
335                     updateTimeLog();
336                 }
337
338             }
339         }
340
341         /**
342          * Takes the user input and processes the new time log entry before adding
343          * it to the times log file.
344          */
345         public void updateTimeLog() {
346
347             String result = null;
348
349             //Obtain the selected entrant from the arraylist of entrants.
350             Entrant currentEntrant = data.getEntrants().get(entrantList.getSelectedIndex());
351
352             //Obtain the arraylist of course nodes that make up the course that entrant is
                    registered for.
353             ArrayList<Node> entrantNodes = proc.obtainEntrantCourseNodes(currentEntrant);
354
355             //Create a formatter for the time value entered by the user.
356             SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
357
358             //Obtain the number of the node selected by the user.
359             int nodeNumber = Integer.parseInt((String) nodeList.getSelectedItem());
```

```
360
361         //Format the time value entered by the user.
362         String newTimeValue = sdf.format(timeSpinner.getValue());
363
364         //Check to see if the node selected was a MC.
365         if (mcTypeList.isEnabled()) {
366
367             //If so determine whether they were arriving or departing.
368             String mcSelection = (String) mcTypeList.getSelectedItem();
369
370             /*
371              * Check if the user is attempting to log an entrant arriving
372              * at a MC while the entrant is currently at an MC.
373              */
374             if (currentEntrant.getAtMC() && mcSelection.equals("Arriving")) {
375
376                 updateStatus(" ERROR: Entrant " + currentEntrant.getNumber() + " already
                        at medical checkpoint.");
377
378             } else if (!(currentEntrant.getAtMC()) && mcSelection.equals("Departing")) {
379
380                 updateStatus(" ERROR: Entrant must be at MC before they can depart.");
381
382             } else {
383
384                 result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber,
                        mcSelection, newTimeValue, mcExclude.isSelected());
385
386                 updateStatus(result);
387             }
388
389         } else {
390
391             //The checkpoint is not a MC, and so just process the new logged time.
392             result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber,
                    newTimeValue);
393             updateStatus(result);
394         }
395     }
396
397     public void updateStatus(String updateMessage) {
398
399         statusBar.setText(updateMessage);
400
401     }
402
403     /**
404      * Listener for actions from sub-panel components, to allow operations to be
405      * run when components are interacted with.
406      *
407      * @see
408      * java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
409      * @param evt - ActionEvent called from components in the panels that
410      * require an action to be performed.
411      */
412     @Override
413     public void actionPerformed(ActionEvent evt) {
414
415         String actionCommand = evt.getActionCommand();
416
417
418         //Switch statement used to capture action commands from buttons.
419         switch (actionCommand) {
420
421             case "Submit Checkpoint Time":
```

```
422 ||
423 ||                  //Submit the entered time values.
424 ||                  submitCheckpoint();
425 ||                  break;
426 ||
427 ||              case "Set to Current Time":
428 ||
429 ||                  //Obtain the current system time from the Calendar class.
430 ||                  Calendar currentTime = Calendar.getInstance();
431 ||
432 ||                  //Update the value of the time spinner to the current time.
433 ||                  timeSpinner.setValue(currentTime.getTime());
434 ||
435 ||                  updateStatus(" Current time updated successfully.");
436 ||                  break;
437 ||
438 ||          }
439 ||
440 ||
441 ||          //Listen for events on the nodes drop-down box component.
442 ||          if (evt.getSource() == nodeList) {
443 ||
444 ||              //Obtain the selected Node object from the collection of nodes.
445 ||              Node n = getNode(Integer.parseInt((String) nodeList.getSelectedItem()));
446 ||
447 ||              //Determine whether the selected node is a medical checkpoint.
448 ||              if (n.getType().equals("MC")) {
449 ||
450 ||                  //If it is, allow the "arrive/depart" selection box to be used.
451 ||                  mcTypeList.setEnabled(true);
452 ||                  mcExclude.setEnabled(true);
453 ||
454 ||              } else {
455 ||
456 ||                  mcTypeList.setEnabled(false);
457 ||                  mcExclude.setEnabled(false);
458 ||              }
459 ||          }
460 ||      }
461 || }
```

# 7 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)
```

# 8 Checkpoint Manager - 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.

## 8.1 Loading External Data Files

Listing 7: 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
```