

CS22510 - C, C++ & Java Paradigms

Assignment 1 - Runners & Riders

2012-2013

Connor Luke Goddard

clg11@aber.ac.uk

March 2013

Contents

1	Introduction	3
1.1	Purpose of this Document	3
1.2	Scope	3
1.3	Objectives	3
2	Event Creator (C++) - Source Code	4
2.1	Header Files	4
2.1.1	Menu.h	4
2.1.2	Process.h	5
2.1.3	Datastore.h	6
2.1.4	FileIO.h	7
2.1.5	Event.h	8
2.1.6	Entrant.h	9
2.1.7	Node.h	10
2.1.8	Course.h	11
2.2	Class Files	12
2.2.1	Main.cpp	12
2.2.2	Menu.cpp	13
2.2.3	Process.cpp	18
2.2.4	Datastore.cpp	24
2.2.5	FileIO.cpp	26
2.2.6	Event.cpp	28
2.2.7	Entrant.cpp	30
2.2.8	Node.cpp	31
2.2.9	Course.cpp	32
3	Event Creator - Build/Compilation Log	33
4	Event Creator - Example Usage	34
5	Event Creator - File Output	40
6	Checkpoint Manager (Java) - Source Code	41
6.1	'Driver' Package	41
6.1.1	CMDriver.java	41
6.2	'Util' Package	42
6.2.1	ProcessData.java	42
6.2.2	FileIO.java	48
6.2.3	LoadData.java	51
6.3	'Model' Package	55
6.3.1	Datastore.java	55
6.3.2	Entrant.java	56
6.3.3	Course.java	59
6.3.4	Node.java	61
6.3.5	Datatype.java	61
6.4	'GUI' Package	63
6.4.1	GUIFrame.java	63
6.4.2	GUIPanel.java	63
7	Checkpoint Manager - Build/Compilation Log	71

8	Checkpoint Manager - Example Usage	72
8.1	Loading External Data Files	72
8.1.1	Correct File Parameters	72
8.1.2	Incorrect File Parameters	73
8.2	Submit Correct Time Entry	73
8.3	Submit Incorrect Time Entry	74
8.4	Entrant Exclusion (Incorrect Node)	74
8.5	Entrant Course Completion	75
8.6	Medical Checkpoint - Successful Arrival	75
8.7	Medical Checkpoint - Successful Departure	76
8.8	Medical Checkpoint - Incorrect Arrival	76
8.9	Medical Checkpoint - Incorrect Departure	77
8.10	Entrant Exclusion (Medical Reasons)	77
8.11	Submission of Invalid Time Value	78
8.12	System Activity Logging	78
8.13	File Lock Access Prevention	79
8.14	Entrant Times File Generation	80
9	Event Manager (C) - Build/Compilation Log	81
10	Event Manager - Example Usage	82
11	Event Manager - Results Output	87
12	Event Manager - System Activity Log	88
13	System Description	89
13.1	Event Creator (C++)	89
13.2	Checkpoint Manager (Java)	89
13.3	Event Manager (C)	89

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 (C++) - 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

```

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

```

2.1.3 Datastore.h

```

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

```

2.1.4 FileIO.h

```

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

```

1  /*
2   * File: Course.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  #include <vector>
10 #include "Node.h"
11
12 #ifndef COURSE_H
13 #define COURSE_H
14
15 /**
16  * Course class used to define the data model for an event course.
17  */
18 class Course {
19
20 public:
21     Course(const char theCourseID);
22     virtual ~Course();
23     void addCourseNode(Node *newNode);
24     void setCourseSize(int courseSize);
25     int getCourseSize(void) const;
26     std::vector<Node*> getCourseNodes(void) const;
27     const char getCourseID(void) const;
28 private:
29
30     const char courseID; /**< Unique ID for a Course.*/
31     std::vector<Node*> courseNodes; /**< Vector of all nodes that make up a course. */
32     int courseSize; /**< The total number of nodes in the course. */
33 };
34
35 #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

```

1  /*
2   * File: Process.cpp
3   * Description: Provides all core functionality and data processing for the
4   * application.
5   * Author: Connor Luke Goddard (clg11)
6   * Date: March 2013
7   * Copyright: Aberystwyth University, Aberystwyth
8   */
9
10 #include <vector>
11 #include <iostream>
12 #include <limits>
13 #include <ctime>
14 #include <sstream>
15 #include <algorithm>
16 #include "Process.h"
17
18 using namespace std;
19
20 /**
21  * Constructor for Process that allows access to the shared Datastore class
22  * created in "main.cpp".
23  * @param newData Pointer to the shared Datastore class created in the main method.
24  */
25 Process::Process(Datastore *newData) {
26
27     data = newData;
28
29 }
30
31 /**
32  * Destructor to be used once object is removed.
33  * Removes the objects stored on the heap.
34  */
35 Process::~Process() {
36
37     delete data;
38
39 }
40
41 /**
42  * Prompts user for input to define an event before creating a new
43  * 'Event' object and storing it's pointer in the shared Datastore class.
44  */
45 void Process::createEvent(void) {
46
47     string inputName, inputDate, inputTime, convertedDate;
48
49     cout << "Please enter an event name/description: ";
50
51     //Obtain all inputted characters including white space.
52     getline(cin, inputName);
53
54     cout << "Please enter the date of the event: (DD/MM/YYYY) ";
55     getline(cin, inputDate);
56
57     cout << "Please enter the time of the event: ";
58     getline(cin, inputTime);
59
60     //Convert inputted date string into format for writing to file.
61     convertedDate = convertDate(inputDate);
62
63     //Create a new Event object on the heap using the inputted information.
64     Event *newEvent = new Event(inputName, convertedDate, inputTime);
65
66     //Check if an Event object already exists on the heap.
67     if (data->getEvent() != NULL) {
68
69         //If it does remove it to prevent a memory leak.
70         delete data->getEvent();
71
72     }
73
74     //Set a pointer to the new object in the shared Datastore class.
75     data->setNewEvent(newEvent);
76
77     cout << "\nEvent (" << inputName << ") created successfully.\n";
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
126                                  //matches.
127                                  notExists = false;
128                                  cout << "\nERROR: This entrant already exists. Please enter
129                                  another value.\n";
130                                  break;
131
132                              } else {
133
134                                  //Otherwise if there is no match, then we can continue.
135                                  notExists = true;
136
137                              }
138
139                          }
140
141                      } else {
142
143                          cout << "Please enter an entrant no: ";
144                          cin >> entrantNo;
145
146                      }
147
148                      break;
149
150                  case 'N':
151                  case 'n':
152
153                      cout << "Setting automatic entrant number\n";
154
155                      //Set entrant number to total number 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 std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
179
180 //Create a new Entrant object on the heap using the inputted information.
181 Entrant *emp = new Entrant(entrantName, entrantNo, courseID);
182
183 //Add a pointer to the new object in the entrant vector stored in Datastore.
184 data->addNewEntrant(emp);
185
186 cout << "\nEntrant(" << entrantNo << ") created successfully.\n";
187
188 }
189
190 /**
191  * Prompts user for the file path to the file that contains all the course
192  * node information, before processing and storing these nodes in a vector
193  * contained in the shared Datastore class.
194  */
195 void Process::getAllNodes(void) {
196
197     string fileName;
198
199     //Obtain the file path from the user.
200     cout << "Welcome. Please enter the file path for course nodes:\n";
201     getline(cin, fileName);
202
203     //Read-in all the node data from the file and store it all in a vector.
204     vector<vector<string> > fileContents = io.getFile(fileName);
205
206     //Check if the data has been successfully parsed and read-in.
207     if (fileContents.size() <= 0) {
208
209         cout << "ERROR: Nodes file (" << fileName << ") could not be located.\n\n"
210             << "Please check the file path and try again. Exiting...\n";
211
212         //If the node data could not be loaded, terminate the program.
213         exit(EXIT_FAILURE);
214     } else {
215
216         //Loop through every line read-in from the file.
217         for (vector<vector<string> > ::iterator it = fileContents.begin(); it != fileContents.
218             end(); ++it) {
219
220             //Convert the first value on the line (node number) to an int.
221             int value = atoi((*it).at(0).c_str());
222
223             //Create a new Node object on the heap using the inputted information.
224             Node *tempNode = new Node(value, (*it).at(1));
225
226             //Add a pointer to the new object in the node vector stored in Datastore
227             data->addNewNode(tempNode);
228         }
229         cout << "Course nodes loaded successfully.\n" << "Loading program...\n\n";
230     }
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     } else {
290
291         //Check that the ID inputted by the user is a letter.
292         while (!isalpha(cid)) {
293
294             //Prompt user for a course ID.
295             cout << "Please enter a new course ID: ";
296             cin >> cid;
297
298             if (!isalpha(cid)) {
299
300                 cout << "ERROR: Course ID's can contain letters only. Please try again\n";
301
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     cout << "\nCourse (" << cid << ") created successfully.\n";
314 }
315
316 /**
317  * Allows a user to specify a new node (loaded in from "nodes.txt") that
318  * that will form part of a particular course.
319  * @param currentCourse The course that a user wishes to add a new node to.
320  */
321 void Process::addCourseNode(Course *currentCourse) {
322
323     int nodeNo;
324
325     cout << "Please select a node to add: \n";
326
327     //Obtain a vector of ALL the course nodes stored in Datastore node vector.
328     std::vector<Node*> allNodes = data->getNodeList();
329
330     //Print all the nodes to screen to provide user with a list to select from.
331     for (std::vector<Node*>::iterator it = allNodes.begin(); it != allNodes.end(); ++it) {
332         cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << ")", ";
333     }
334
335     cout << endl;
336
337     //Prompt user for the number of the node they wish to add.
338     cin >> nodeNo;
339
340     //Attempt to fetch the specified node from the vector of nodes in Datastore.
341     Node *tempNode = data->obtainNode(nodeNo);
342
343     //Check to see if a matching node was located.
344     if (tempNode != NULL) {
345
346         /**
347          * Add a pointer to the located node object in the course's vector
348          * of nodes.
349          */
350         currentCourse->addCourseNode(tempNode);
351         cout << "\nNode (" << tempNode->getNodeNo() << ") added successfully.\n";
352
353         //Obtain the vector of all nodes contained within the course.
354         std::vector<Node*> currentCourseNodes = currentCourse->getCourseNodes();
355
356         //Display a list of all the nodes that make up the course on screen.
357         cout << "\nCurrent nodes contained in Course (" << currentCourse->getCourseID() << "):\n";
358
359         for (std::vector<Node*>::iterator it = currentCourseNodes.begin(); it !=
360             currentCourseNodes.end(); ++it) {
361             cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << ")\n";
362         }
363     } else {
364
365         //Otherwise if no matching node can be found, inform the user.
366         cout << "\nERROR: Node " << nodeNo << " not found.\n";
367     }
368 }
369
370 /**
371  * Attempts to locate a course from Datastore that matches
372  * the ID entered by the user.
373  * @returns The pointer to a matching course or NULL.
374  */
375 Course* Process::getSelectedCourse(void) {
376
377     char selectedID;
378
379     //Check that the ID inputted by the user is a letter.
380     while (!isalpha(selectedID)) {
381
382         //Prompt user for a course ID.
383         cout << "Please enter an existing course ID: ";
384         cin >> selectedID;
385
386         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     event = NULL;
19 }
20
21 /**
22  * Destructor to be used once object is removed.
23  */
24 Datastore::~Datastore() {
25     delete event;
26 }
27
28 /**
29  * Fetches the vector of all the courses created for an event.
30  * @return A vector that contains pointers to all the Course objects created.
31  */
32 std::vector<Course*> Datastore::getCourseList(void){
33     return courseList;
34 }
35
36 /**
37  * Fetches the vector of all the nodes read in from "nodes.txt".
38  * @return A vector that contains pointers to all the Node objects.
39  */
40 std::vector<Node*> Datastore::getNodeList(void) {
41     return nodeList;
42 }
43
44 /**
45  * Fetches the vector of all the entrants created for an event.
46  * @return A vector that contains pointers to all the Entrant objects created.
47  */
48 std::vector<Entrant*> Datastore::getEntrantList(void){
49     return entrantList;
50 }
51
52 /**
53  * Fetches the Event object created to define the race event.
54  * @return A pointer to the created Event object.
55  */
56 Event* Datastore::getEvent(void) const {
57     return event;
58 }
59
60 /**
61  * Adds a new Course object to the end of the 'courseList' vector.
62  * @param newCourse Pointer to the new Course object to be added to the vector.
63  */
64 void Datastore::addNewCourse (Course *newCourse) {
65     courseList.push_back(newCourse);
66 }
67
68 /**
69  * Adds a new Node object to the end of the 'nodeList' vector.
70  * @param newNode Pointer to the new Node object to be added to the vector.
71  */
72 void Datastore::addNewNode (Node *newNode) {
73     nodeList.push_back(newNode);
74 }
75
76
77
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
113         //If the ID of the current course matches the inputted ID...
114         if ((*it)->getCourseID() == selectedID) {
115
116             //... return the pointer to that Course object.
117             return (*it);
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  * Writes the details of a particular event to file using a
99  * specified format. Event information is obtained from the 'event' pointer
100  * stored within the Datastore class.
101  * @param event Pointer to the stored Event class.
102  */
103 void FileIO::writeEvent(Event *event) {
104
105     ofstream myfile;
106     myfile.open("../files/exampleevent.txt", ios::out | ios::trunc);
107
108     myfile << (*event).getEventName() << "\n" << (*event).getEventDate() << "\n" << (*event).
        getEventTime() << "\n";
109
110     myfile.close();
111 }
112
113 /**
114  * Accesses a specified file and returns the contents as a vector.
115  * @param fileName The file path of the specified file.
116  * @return A vector of vectors that each contain the contents of each line
117  * of the file that was read in.
118  */
119 vector<vector<string>>> FileIO::getFile(string fileName) {
120
121     string line;
122
123     //Create a new file stream.
124     ifstream myfile(fileName.c_str());
125
126     //Check the file has been successfully opened.
127     if (myfile.is_open()) {
128
129         //Read the entire contents of the file.
130         while (std::getline(myfile, line)) {
131
132             //Split the current line by white space into separate tokens.
133             istringstream ss(line);
134             istream_iterator<string> begin(ss), end;
135
136             //Place all the tokens into a new vector (For the line).
137             vector<string> allStrings(begin, end);
138
139             //Add this vector to the parent vector for the whole file.
140             arrayTokens.push_back(allStrings);
141
142         }
143
144         myfile.close();
145     }
146
147     //Return the vector. If the loading was un-successful, it will be empty.
148     return arrayTokens;
149 }
150
151 }

```

2.2.6 Event.cpp

```

1  /*
2   * File: Event.cpp
3   * Description: Provides a data model for a particular event.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include "Event.h"
10
11  using namespace std;
12
13  /**
14   * Default constructor for Event.
15   */
16  Event::Event() {
17
18  }
19
20  /**
21   * Constructor that allows the characteristics of an event to be specified.
22   * @param newEventName The inputted name/description of the event.
23   * @param newEventDate The inputted date of the event.
24   * @param newEventTime The inputted start time of the event.
25   */
26  Event::Event(string newEventName, string newEventDate, string newEventTime) {
27
28      eventName = newEventName;
29      eventDate = newEventDate;
30      eventTime = newEventTime;
31
32  }
33
34  /**
35   * Destructor to be used once object is removed.
36   */
37  Event::~Event() {
38
39  }
40
41  /**
42   * Updates the start time of the event to an inputted value.
43   * @param eventTime Recently inputted start time value.
44   */
45  void Event::setEventTime(string eventTime) {
46      this->eventTime = eventTime;
47  }
48
49  /**
50   * Fetches the start time of the event.
51   * @return The value of the 'eventTime' string variable.
52   */
53  string Event::getEventTime(void) const {
54      return eventTime;
55  }
56
57  /**
58   * Updates the date of the event to an inputted value.
59   * @param eventDate Recently inputted event date value.
60   */
61  void Event::setEventDate(string eventDate) {
62      this->eventDate = eventDate;
63  }
64
65  /**
66   * Fetches the date of the event.
67   * @return The value of the 'eventDate' string variable.
68   */
69  string Event::getEventDate(void) const {
70      return eventDate;
71  }
72
73  /**
74   * Updates the name/description of the event to an inputted value.
75   * @param eventTime Recently inputted event name/description..
76   */
77  void Event::setEventName(string eventName) {
78      this->eventName = eventName;

```

```
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

```

1  /*
2   * File: Entrant.cpp
3   * Description: Provides a data model for an entrant in an event.
4   * Author: Connor Luke Goddard (clg11)
5   * Date: March 2013
6   * Copyright: Aberystwyth University, Aberystwyth
7   */
8
9  #include "Entrant.h"
10 #include <iostream>
11
12 using namespace std;
13
14
15 /**
16  * Constructor that allows the constant 'entrant_name' and 'entrant_no' variables
17  * to be specified. Also specifies the ID of the course the entrant is registered for.
18  * @param theName The inputted name of the entrant.
19  * @param theEnNo The inputted unique entrant number.
20  * @param theCourseID The new course ID value to be set.
21  */
22 Entrant::Entrant(const std::string &theName, const int theEnNo, char theCourseID) :
23     entrant_name(theName), entrant_no(theEnNo){
24
25     course_id = theCourseID;
26 }
27
28 /**
29  * Destructor to be used once object is removed.
30  */
31 Entrant::~Entrant() {
32 }
33
34 /**
35  * Fetches the name of the entrant.
36  * @return A string containing the name of the entrant.
37  */
38 string Entrant::getEntrantName(void) {
39     return entrant_name;
40 }
41
42 /**
43  * Fetches the ID number of the entrant.
44  * @return An integer containing the entrant number.
45  */
46 int Entrant::getEntrantNo(void) {
47     return entrant_no;
48 }
49
50 /**
51  * Fetches the ID of the course the entrant is registered for.
52  * @return An char containing the course ID.
53  */
54 char Entrant::getCourseID(void) {
55     return course_id;
56 }

```

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 /**
48  * Fetches the value of 'courseSize'.
49  * @return The total number of nodes in the course.
50  */
51 int Course::getCourseSize(void) const {
52     return courseSize;
53 }
54
55 /**
56  * Fetches a vector of all the nodes in the course.
57  * @return A vector of nodes that make up the course.
58  */
59 std::vector<Node*> Course::getCourseNodes(void) const {
60     return courseNodes;
61 }
62
63 /**
64  * Fetches the ID of the course.
65  * @return The ID of the course.
66  */
67 const char Course::getCourseID(void) const {
68     return courseID;
69 }

```

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

```

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

```

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/identknow.txt
ERROR: Nodes file (../files/identknow.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 (Java) - 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 LoadData 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

```

1 package aber.dcs.cs22510.clg11.util;
2
3 import aber.dcs.cs22510.clg11.model.Course;
4 import aber.dcs.cs22510.clg11.model.Datastore;
5 import aber.dcs.cs22510.clg11.model.Entrant;
6 import aber.dcs.cs22510.clg11.model.Node;
7 import java.io.File;
8 import java.io.IOException;
9 import java.text.ParseException;
10 import java.text.SimpleDateFormat;
11 import java.util.ArrayList;
12 import java.util.Date;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15
16 /**
17  * Responsible for updating the internal record of entrant progress (based on
18  * data read in from "times.txt") and for processing new time logs submitted by
19  * a user. Has access to the shared
20  * {@link aber.dcs.cs22510.clg11.model.Datastore} class to allow processing and
21  * manipulation of the data collections.
22  *
23  * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
24  * Aberystwyth.
25  */
26 public class ProcessData {
27
28     private Datastore data;
29     private FileIO fileIO;
30     private String lastLoggedTime = null;
31
32     /**
33      * Allows access to the file read/write facilities.
34      */
35     // private FileIO fileIO = new FileIO();
36     public ProcessData() {
37     }
38
39     /**
40      * Constructor to instantiate a new ProcessData. Takes the shared data store
41      * object created in {@link aber.dcs.cs22510.clg11.driver.CMDriver} as a
42      * parameter to allow accessed to the lists of nodes/entrants/courses loaded
43      * in.
44      *
45      * @param newData Datastore object created in CMDriver.
46      * @param newFileIO
47      */
48     public ProcessData(Datastore newData, FileIO newFileIO) {
49
50         this.data = newData;
51         this.fileIO = newFileIO;
52
53     }
54
55     /**
56      * Returns the time value of the last read-in time log.
57      *
58      * @return The last time value of the "times.txt" file.
59      */
60     public String getLastLoggedTime() {
61         return lastLoggedTime;
62     }
63
64     /**
65      * Attempts to fetch a specified entrant from the internal collection of
66      * entrants.
67      *
68      * @param requiredEntrant The number of the required entrant.
69      * @return The specified Entrant object, or NULL.
70      */
71     public Entrant obtainEntrant(int requiredEntrant) {
72
73         for (Entrant e : data.getEntrants()) {
74
75             if (e.getNumber() == requiredEntrant) {

```

```

76         return e;
77     }
78
79 }
80
81 }
82     return null;
83 }
84
85 /**
86  * Attempts to fetch the collection of course nodes that make up the course
87  * that a specified entrant is registered for.
88  *
89  * @param selectedEntrant The number of the required entrant.
90  * @return The collection of course nodes, or NULL.
91  */
92 public ArrayList<Node> obtainEntrantCourseNodes(Entrant selectedEntrant) {
93
94     //Loop through all the stored courses.
95     for (Course c : data.getCourses()) {
96
97         //If the current course matches the entrant's course.
98         if (c.getCourseID() == selectedEntrant.getCourseID()) {
99
100             //Return the collection of nodes for that course.
101             return c.getCourseNodes();
102         }
103     }
104
105     //Otherwise if nothing is found, return NULL.
106     return null;
107 }
108
109 /**
110  * Processes each line read in from the "times.txt" file to update the
111  * internal record of entrant's progress. This method is crucial to ensure
112  * that any time log updates created by any other running versions of the
113  * checkpoint manager are recorded in the internal entrant record within
114  * this application.
115  *
116  * @param timeDelimiter The character symbol used to represent the status of
117  * the particular time log.
118  * @param nodeNo The number of the node the time log was recorded at.
119  * @param entrantNo The number of the entrant that was recorded.
120  * @throws IndexOutOfBoundsException
121  */
122 public void processNewTime(String timeDelimiter, int nodeNo, int entrantNo) throws
123     IndexOutOfBoundsException {
124
125     //Boolean used to determine whether this particular time log has been processed.
126     boolean isUpdated = false;
127
128     //Obtain the required entrant from the internal collection of entrants.
129     Entrant currentEntrant = obtainEntrant(entrantNo);
130
131     //Check if the time log dictates that the entrant should be excluded.
132     if (timeDelimiter.equals("I") || timeDelimiter.equals("E")) {
133
134         //If so exclude the entrant.
135         excludeEntrant(entrantNo);
136
137         //Log this activity in the log file ("log.txt");
138         fileIO.addActivityLog("Entrant no: " + entrantNo + " successfully excluded.");
139
140         /*
141          * Otherwise if they shouldn't be excluded,
142          * check to see if the entrant has already been excluded.
143          */
144     } else if (!currentEntrant.getIsExcluded()) {
145
146         ArrayList<Node> courseNodes = obtainEntrantCourseNodes(currentEntrant);
147
148         //Loop through all the nodes that make up the course the entrant is on.
149         for (int i = 0; i < courseNodes.size(); i++) {
150
151             /*
152              * Check that the current progress of the entrant < the index of
153              * the current node in the array (to prevent nodes the entrant has
154              * already passed being used again), and the current node equals

```

```

155         * the node number of the current time log.
156         */
157         if (i > (currentEntrant.getCurrentProgress() - 1) && courseNodes.get(i).
            getNumber() == nodeNo && !isUpdated) {
158
159             /*
160              * If the entrant has ARRIVED at a medical checkpoint,
161              * their progress should not be incremented as they are
162              * now waiting at the MC
163              */
164             if (timeDelimiter.equals("A")) {
165
166                 //Just prevent this particular time log being processed any further.
167                 currentEntrant.setAtMC(true);
168                 isUpdated = true;
169
170             /*
171              * Otherwise, if they are DEPARTING from a MC or they
172              * have just arrived at a normal checkpoint, then their
173              * progress needs to be recorded and incremented.
174              */
175             } else {
176
177                 /*
178                  * If the read in node from time file is further along
179                  * the course than the current progress,
180                  * update the current progress.
181                  */
182                 currentEntrant.setCurrentProgress((i + 1));
183                 currentEntrant.setAtMC(false);
184
185             /*
186              * Check to see if the entrant has now completed
187              * their course.
188              */
189             if (currentEntrant.getCurrentProgress() >= courseNodes.size()) {
190
191                 //Log that they have finished.
192                 currentEntrant.setIsFinished(true);
193
194                 //Log this activity in the log file ("log.txt");
195                 fileIO.addActivityLog("Entrant no: " + currentEntrant.getNumber() +
196                                     " has successfully finished the course.");
197             }
198
199             isUpdated = true;
200
201         }
202     }
203 }
204
205 /**
206  * Updates a particular Entrant object to log the fact that they have been
207  * excluded from their race.
208  *
209  * @param entrantNo The number of the required entrant.
210  */
211 public void excludeEntrant(int entrantNo) {
212     for (Entrant e : data.getEntrants()) {
213         if (e.getNumber() == entrantNo) {
214             e.setIsExcluded(true);
215         }
216     }
217 }
218
219 /**
220  * Processes a new time log submitted by the user by determining whether the
221  * entrant is on the correct path or not and updates the "times.txt" file
222  * with the resulting time log.
223  *
224  * @param courseNodes The collection of nodes that make up the course the
225  * current entrant is registered for.

```

```

233 * @param selectedEntrant The current entrant being processed.
234 * @param newNode The newly submitted node that the entrant has arrived at.
235 * @param time The inputted time of the entrant's arrival at the CP.
236 * @return
237 */
238 public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int
    newNode, String time) {
239
240     //Obtain the current progress of the entrant (i.e. the index of the array).
241     int nextNodeIndex = selectedEntrant.getCurrentProgress();
242     String result = null;
243     boolean timesNotLocked = true;
244     boolean logNotLocked = true;
245
246     //Check that the entrant has not already finished, or been excluded.
247     if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
248
249         result = " Entrant " + selectedEntrant.getNumber() + " successfully completed their
            course.";
250
251     } else if (selectedEntrant.getIsExcluded()) {
252
253         result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from their
            course.";
254
255     } else {
256
257         /*
258          * Check whether the next node in the array (i.e. the next node that the
259          * entrant SHOULD have reached) is actually the node submitted.
260          */
261         if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
262
263             /*
264              * If they do not match, the entrant has gone the wrong way.
265              * Append this new time log with the 'I' time delimiter to the
266              * times file ("times.txt").
267              */
268             timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " +
                newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
269
270             logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                incorrect for course. (Entrant No: " + selectedEntrant.getNumber() + ")");
271
272             result = "Entrant " + selectedEntrant.getNumber()
273                 + " has gone the INCORRECT way. (Expected node: " + courseNodes.get(
                    nextNodeIndex).getNumber() + ")";
274
275         } else {
276
277             /*
278              * Otherwise if they do match, the entrant has gone the right way.
279              * Append this new time log with the 'T' time delimiter to the
280              * times file ("times.txt").
281              */
282             timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "T " +
                newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
283
284             logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                incorrect for course. (Entrant No: " + selectedEntrant.getNumber() + ")");
285
286             result = "Entrant " + selectedEntrant.getNumber()
287                 + " has gone the CORRECT way. (Expected node: " + courseNodes.get(
                    nextNodeIndex).getNumber() + ")";
288
289         }
290     }
291
292     /*
293     * If any of the output files are locked by another process/application,
294     * inform the user.
295     */
296     if (!logNotLocked) {
297
298         result = " ERROR: System log file locked - Cannot write to file.";
299
300     }
301
302     if (!timesNotLocked) {
303

```

```

304         result = " ERROR: Times log file locked - Cannot write to file. Please try again.";
305     }
306 }
307
308 if (!timesNotLocked && !logNotLocked) {
309     result = " ERROR: Cannot write to time log or log file. - Both files locked.";
310 }
311
312 return result;
313
314 }
315
316 /**
317  * Processes a new time log submitted by the user by determining whether the
318  * entrant is on the correct path or not and updates the "times.txt" file
319  * with the resulting time log (Overloaded method for processing medical
320  * checkpoints).
321  *
322  * @param courseNodes The collection of nodes that make up the course the
323  * current entrant is registered for.
324  * @param selectedEntrant The current entrant being processed.
325  * @param newNode The newly submitted node that the entrant has arrived at.
326  * @param mcType Whether the entrant was arriving or departing from the MC.
327  * @param time The inputted time of the entrant's arrival at the CP.
328  * @param isExcluded
329  * @return String containing the result of processing the time log.
330  */
331 public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int
332     newNode, String mcType, String time, boolean isExcluded) {
333
334     int nextNodeIndex = selectedEntrant.getCurrentProgress();
335     String result = null;
336     boolean timesNotLocked = true;
337     boolean logNotLocked = true;
338
339     //Check that the entrant has not already finished, or been excluded.
340     if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {
341
342         result = " Entrant " + selectedEntrant.getNumber() + " successfully completed their
343             course.";
344     } else if (selectedEntrant.getIsExcluded()) {
345
346         result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from their
347             course.";
348     } else {
349
350         if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {
351
352             logNotLocked = fileIO.addActivityLog("Submitted checkpoint incorrect for course
353                 . (Entrant No: " + selectedEntrant.getNumber() + ")");
354
355             timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " +
356                 newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
357
358             result = "Entrant " + selectedEntrant.getNumber()
359                 + " has gone the wrong way. (Expected node: " + courseNodes.get(
360                     nextNodeIndex).getNumber() + ")";
361
362         } else if (isExcluded) {
363
364             logNotLocked = fileIO.addActivityLog("Entrant excluded for medical reasons. (
365                 Entrant No: " + selectedEntrant.getNumber() + ")");
366
367             timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "E " +
368                 newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
369
370             result = "Entrant " + selectedEntrant.getNumber()
371                 + " has been excluded for medical reasons.";
372
373         } else {
374
375             /*
376              * If they do match, the entrant has gone the right way.
377              * Determine whether the entrant was arriving at, or departing
378              * from the MC and update the time log file ("times.txt")
379              * accordingly.
380              */

```

```

376         if (mcType.equals("Arriving")) {
377
378             logNotLocked = fileIO.addActivityLog("New MC arrival time submitted. (
379                 Entrant No: " + selectedEntrant.getNumber() + ")");
380
381             timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "A " +
382                 newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
383
384             result = "Entrant " + selectedEntrant.getNumber()
385                 + " has successfully arrived at MC " + courseNodes.get(
386                     nextNodeIndex).getNumber() + ".";
387
388             } else {
389
390                 logNotLocked = fileIO.addActivityLog("New MC departure time submitted. (
391                     Entrant No: " + selectedEntrant.getNumber() + ")");
392                 timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "D " +
393                     newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");
394
395                 result = "Entrant " + selectedEntrant.getNumber()
396                     + " has successfully departed from MC " + courseNodes.get(
397                         nextNodeIndex).getNumber() + ".";
398             }
399         }
400     }
401
402     /*
403     * If any of the output files are locked by another process/application,
404     * inform the user.
405     */
406     if (!timesNotLocked) {
407         result = " ERROR: Times log file locked - Cannot write to file. Please try again.";
408     }
409
410     if (!logNotLocked) {
411         result = " ERROR: System log file locked - Cannot write to file. Please try again."
412             ;
413     }
414
415     if (!timesNotLocked && !logNotLocked) {
416         result = " ERROR: Cannot write to time log file or system log file. - Both files
417             locked.";
418     }
419
420     return result;
421 }
422
423 /**
424  * Obtains all the times from the time log file ("times.txt") before
425  * processing each time log.
426  * @return A boolean determining if the file was successfully loaded or not.
427  */
428 public boolean getTimes() {
429
430     //Obtain a collection of ALL the time logs read in from the "times.txt" file.
431     File timesFile = new File("../files/times.txt");
432
433     if (timesFile.exists()) {
434
435         ArrayList<String[]> times = fileIO.readIn(timesFile, true);
436
437         //For every time log read in from the file...
438         for (String[] newTime : times) {
439
440             //... process this time log and update the internal record of entrants.
441             processNewTime(newTime[0], Integer.parseInt(newTime[1]), Integer.parseInt(
442                 newTime[2]));
443
444             this.lastLoggedTime = newTime[3];
445         }
446     }

```



```

447         //Log this activity in the log file ("log.txt");
448         fileIO.addActivityLog("Time logs file loaded successfully (times.txt)");
449
450         return true;
451     }
452 }
453
454     return false;
455 }
456
457 /**
458  * Compares the time of the last read-in time log, with the new time being
459  * submitted to check that the user is not entering a time in the past.
460  *
461  * @param oldTimeString The last time value read-in from "times.txt".
462  * @param newTimeString The new time value being submitted by the user.
463  * @return A boolean determining if the new time value is in the past.
464  */
465 public boolean compareTimes(String oldTimeString, String newTimeString) {
466
467     SimpleDateFormat df = new SimpleDateFormat("HH:mm");
468
469     Date lastRecordedTime;
470     Date newTime;
471
472     try {
473
474         //Create new Date objects using the last logged, and new time values.
475         lastRecordedTime = df.parse(oldTimeString);
476         newTime = df.parse(newTimeString);
477
478         //Check if the new time entered is before the last read-in time.
479         if (df.format(lastRecordedTime).compareTo(df.format(newTime)) > 0) {
480
481             //If so, then this cannot be allowed.
482
483             //Log this activity in the log file ("log.txt");
484             fileIO.addActivityLog("User attempted to enter new time value in the past. (New
485                 time: " + df.format(newTime) + ")");
486
487             return true;
488         }
489     } catch (ParseException ex) {
490         Logger.getLogger(ProcessData.class.getName()).log(Level.SEVERE, null, ex);
491     }
492
493     return false;
494 }
495 }

```

6.2.2 FileIO.java

```

1 package aber.dcs.cs22510.clg11.util;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileOutputStream;
6 import java.io.FileReader;
7 import java.io.FileWriter;
8 import java.io.IOException;
9 import java.nio.channels.FileLock;
10 import java.nio.channels.OverlappingFileLockException;
11 import java.text.DateFormat;
12 import java.text.SimpleDateFormat;
13 import java.util.ArrayList;
14 import java.util.Calendar;
15
16 /**
17  * Provides file I/O facilities to allow data files to be read into the system,
18  * and the time file to be updated/appended to as required.
19  *
20  * @author Connor Luke Goddard (clg11) Copyright: Aberystwyth University,
21  * Aberystwyth.
22  */
23 public class FileIO {
24
25     /**
26      * The last read-in line number from "times.txt".
27      */

```

```

28     private int timesFilePosition = 0;
29
30     /**
31      * Default constructor for FileIO.
32      */
33     public FileIO() {
34     }
35
36     /**
37      * Reads in the contents of specified data files and places the contents
38      * into an ArrayList which is then returned, and used to update the internal
39      * data collections used by the application.
40      *
41      * @param fileName The directory of the file to be parsed.
42      * @param isTimesFile
43      * @return ArrayList of String arrays containing the contents of the parsed
44      *         file.
45      */
46     public ArrayList<String[]> readIn(File fileName, boolean isTimesFile) {
47
48         ArrayList<String[]> values = new ArrayList<>();
49
50         try {
51
52             //Create File IO objects
53             FileReader fileReader;
54             BufferedReader bufferedReader;
55
56             //Initialise the File IO objects, passing in the selected file path
57             fileReader = new FileReader(fileName);
58             bufferedReader = new BufferedReader(fileReader);
59
60
61             /*
62              * Check if the current file being read in is the times file,
63              * and if so whether or not the file has been read-in previously.
64              */
65             if (isTimesFile && this.timesFilePosition > 0) {
66
67                 /*
68                  * Read down to the last logged line read-in file
69                  * without processing any of the lines (used to "skip" down
70                  * to any lines that could have been added after the last time
71                  * the file was read in by this application).
72                  */
73                 for (int i = 0; i < this.timesFilePosition; i++) {
74                     bufferedReader.readLine();
75                 }
76             }
77
78             //Initialise local variable used to store the current line being read in
79             String line;
80
81             //While there are still lines to read in from the file (i.e. read in every line in
82             //the file)
83             while ((line = bufferedReader.readLine()) != null) {
84
85                 //As there is multiple data on each line, split the values up.
86                 String[] details = line.split(" ");
87
88                 //Add these broken down values to the larger collection of lines.
89                 values.add(details);
90
91                 /*
92                  * If the current file being read in is "times.txt", updated the
93                  * last line to be read in by the system. (Used for when the
94                  * file is re-"readin" by the system).
95                  */
96                 if (isTimesFile) {
97                     timesFilePosition++;
98                 }
99             }
100
101             //Once completed, safely close the file reader
102             bufferedReader.close();
103
104             return values;
105
106             //If any IO exceptions occur...
107         } catch (IOException iOE) {

```

```

107         return null;
108     }
109 }
110
111 }
112
113 /**
114  * Writes output data to specified files, as these files are shared, file
115  * locking has to be used to prevent corruption of data/files.
116  *
117  * @param writeFile The file that is to be written to.
118  * @param output The output data string.
119  * @return A boolean determining if the file was successfully written to.
120  */
121 public boolean writeFile(File writeFile, String output) {
122
123     FileOutputStream fos;
124     FileLock fl = null;
125
126     try {
127
128         //If the file does not exist, create a new file.
129         if (!writeFile.exists()) {
130             writeFile.createNewFile();
131         }
132
133         //Create a new output stream that will append to the file.
134         fos = new FileOutputStream(writeFile.getAbsolutePath(), true);
135
136         //Attempt to lock the file to allow the data to be written.
137         try {
138
139             fl = fos.getChannel().tryLock();
140
141         } catch (OverlappingFileLockException flE) {
142
143             /*
144              * If there is already a process within the same JVM locking
145              * the file, inform the user.
146              */
147             System.out.println("ERROR: File <" + writeFile.getName() + "> cannot be
148                               accessed. File lock still in place.");
149         }
150
151         //Check if the lock was successful.
152         if (fl != null) {
153
154             try (FileWriter fw = new FileWriter(fos.getFD())) {
155
156                 fw.write(output);
157
158                 //Once the data has been successfully written, release the lock.
159                 fl.release();
160             }
161
162             return true;
163         }
164
165     } catch (IOException e) {
166
167     }
168
169     return false;
170 }
171
172 /**
173  * Adds a new log message to the "logs.txt" file. Called when a major
174  * activity occurs in the application.
175  *
176  * @param logMessage Message describing the activity.
177  * @return Boolean determining if the log was successfully written to file.
178  */
179 public boolean addActivityLog(String logMessage) {
180
181     //Obtain the current date/time and format it for use in the log file.
182     DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
183     Calendar cal = Calendar.getInstance();
184
185     //Build the log message using predefined output template.

```

```

186         String logOutput = "LOG - CM: " + logMessage + " - " + dateFormat.format(cal.getTime())
187             + "\n";
188         //Write the log message to the log file.
189         return writeFile(new File("../files/log.txt"), logOutput);
190     }
191 }
192 }

```

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 comp Shared Datastore object created within CMDriver.
26       * @param newFileIO Shared FileIO object created within CMDriver.
27       */
28      public LoadData(Datastore comp, FileIO newFileIO) {
29
30          this.data = comp;
31          this.fileIO = newFileIO;
32
33      }
34
35      /**
36       * Prompts user for the file path of a specified file before attempting to
37       * load the data into it's respective data collection.
38       *
39       * @param type ENUM denoting the type of data file (Node, Course or
40       * Entrant).
41       * @param fileName The path of the file to be loaded.
42       */
43      public void loadFiles(Datatype type, String fileName) {
44
45          File f = new File(fileName);
46          ArrayList<String[]> readValues;
47
48          //Check if the file exists.
49          if (!f.exists()) {
50
51              //If it does not exist, inform the user.
52              if (type == Datatype.NODE) {
53
54                  System.out.println("ERROR: Nodes file <" + fileName + "> does not exist.");
55
56              } else if (type == Datatype.COURSE) {
57
58                  System.out.println("ERROR: Courses file <" + fileName + "> does not exist.");
59
60              } else {
61
62                  System.out.println("ERROR: Entrants file <" + fileName + "> does not exist.");
63
64              }
65
66              System.out.println("Parameter format = <node path> <courses path> <entrants path>");
67              ;
68              System.exit(0);
69          }
70      }
71  }

```

```

69
70
71 //If the file does exist, read in the data from the file.
72 readValues = fileIO.readIn(f, false);
73
74 //Determine the type of data being loaded.
75 if (type.equals(Datatype.NODE)) {
76
77     for (String[] newItem : readValues) {
78
79         //Load all the nodes from the read-in data.
80         loadNodes(newItem);
81
82     }
83
84     System.out.println("Nodes file loaded successfully (nodes.txt)");
85
86     //Log this activity in the log file ("log.txt");
87     fileIO.addActivityLog("Nodes file loaded successfully (nodes.txt)");
88
89 } else if (type.equals(Datatype.COURSE)) {
90
91     for (String[] newItem : readValues) {
92
93         loadCourses(newItem);
94
95     }
96
97     System.out.println("Courses file loaded successfully (courses.txt)");
98     fileIO.addActivityLog("Courses file loaded successfully (courses.txt)");
99
100 } else {
101
102     for (String[] newItem : readValues) {
103
104         loadEntrants(newItem);
105
106     }
107
108     System.out.println("Entrants file loaded successfully (entrants.txt)");
109     fileIO.addActivityLog("Entrants file loaded successfully (entrants.txt)");
110 }
111
112 }
113
114 /**
115  * Parses the data read-in from the "courses.txt" file and creates a new
116  * {@link aber.dcs.cs22510.clg11.model.Course} object populated with the
117  * read-in characteristics. This new Course object is then added to the
118  * internal collection of Courses.
119  *
120  * @param courseData Collection of all course characteristics data read in
121  * from "courses.txt".
122  */
123 public void loadCourses(String[] courseData) {
124
125     try {
126
127         //Create a new empty Course object.
128         Course newCourse = new Course();
129
130         //Set the course ID to the first element in the course data array.
131         newCourse.setCourseID(courseData[0].charAt(0));
132
133         //Set the course length to the second element in the course data array.
134         newCourse.setCourseLength(Integer.parseInt(courseData[1]));
135
136         //Loop through the REST (i=2) of the "read-in" course data array..
137         for (int i = 2; i < (courseData.length); i++) {
138
139             //Loop through all the course nodes stored internally.
140             for (Node n : data.getNodes()) {
141
142                 int origNodeNo = n.getNumber();
143
144                 //Obtain the node number currently being parsed from the read in data.
145                 int courseNodeNo = Integer.parseInt(courseData[i]);
146
147                 /*
148                  * If the node number read-in from file matches the current node,

```

```

149         * and the node is NOT a junction, add this node to the collection
150         * of nodes within the new Course object.
151         */
152         if (origNodeNo == courseNodeNo && (n.getType().equals("CP") || n.getType().
153             equals("MC"))) {
154             newCourse.addNode(n);
155         }
156     }
157 }
158
159
160 /*
161  * Once the new Course object has been populated with data,
162  * add it to the collection of courses in Datastore.
163  */
164 data.getCourses().add(newCourse);
165
166 //If an error occurs...
167 } catch (Exception e) {
168
169     //... log the error in the "log.txt" file.
170     fileIO.addActivityLog("ERROR - Cannot create new course object (" + courseData[0] +
171         ")");
172 }
173
174 }
175
176 /**
177  * Parses the data read-in from the "nodes.txt" file and creates a new
178  * {@link aber.dcs.cs22510.clg11.model.Node} object populated with the
179  * read-in characteristics. This new Node object is then added to the
180  * internal collection of Nodes.
181  *
182  * @param nodeData Collection of all node characteristics data read in from
183  * "nodes.txt".
184  */
185 public void loadNodes(String[] nodeData) {
186
187     try {
188         Node newNode = new Node();
189
190         newNode.setNumber(Integer.parseInt(nodeData[0]));
191         newNode.setType(nodeData[1]);
192
193         data.getNodes().add(newNode);
194
195     } catch (Exception e) {
196
197         fileIO.addActivityLog("ERROR - Cannot create new node object (" + Integer.parseInt(
198             nodeData[0]) + " / " + nodeData[1] + ")");
199     }
200 }
201
202 /**
203  * Parses the data read-in from the "entrants.txt" file and creates a new
204  * {@link aber.dcs.cs22510.clg11.model.Entrant} object populated with the
205  * read-in characteristics. This new Entrant object is then added to the
206  * internal collection of Entrants.
207  *
208  * @param entrantData Collection of all node characteristics data read in
209  * from "nodes.txt".
210  */
211 public void loadEntrants(String[] entrantData) {
212
213     try {
214
215         Entrant newEntrant = new Entrant();
216
217         newEntrant.setNumber(Integer.parseInt(entrantData[0]));
218         newEntrant.setCourseID(entrantData[1].charAt(0));
219         newEntrant.setFirstName(entrantData[2]);
220         newEntrant.setLastName(entrantData[3]);
221
222         data.getEntrants().add(newEntrant);
223
224     } catch (Exception e) {
225

```

```
226 |         fileIO.addActivityLog("ERROR - Cannot create new entrant object (" + Integer.  
    |         parseInt(entrantData[0]) + " / " + entrantData[1] + " / " + entrantData[2] + "  
227 |         " + entrantData[3] + ")");  
228 |     }  
229 | }  
230 | }
```

6.3 ‘Model’ Package

6.3.1 Datastore.java

```

1 package aber.dcs.cs22510.clg11.model;
2
3 import java.util.ArrayList;
4
5 /**
6  * Stores all internal data used by the system to process existing and new
7  * race time logs (Nodes, Courses and Entrants).
8  *
9  * @author Connor Luke Goddard (clg11)
10  * Copyright: Aberystwyth University, Aberystwyth.
11  */
12 public class Datastore {
13
14     /** ArrayList of all courses in an event. */
15     private ArrayList<Course> courses = new ArrayList<>();
16
17     /** ArrayList of all nodes in an event. */
18     private ArrayList<Node> nodes = new ArrayList<>();
19
20     /** ArrayList of all entrants registered to an event. */
21     private ArrayList<Entrant> entrants = new ArrayList<>();
22
23     /**
24      * Default constructor for a Course.
25      */
26     public Datastore() {
27
28     }
29
30
31     /**
32      * Fetches all courses that are stored for a particular event.
33      * @return The collection of courses.
34      */
35     public ArrayList<Course> getCourses() {
36         return courses;
37     }
38
39
40     /**
41      * Fetches all the nodes that are stored for a particular event.
42      * @return The collection of nodes.
43      */
44     public ArrayList<Node> getNodes() {
45         return nodes;
46     }
47
48
49     /**
50      * Fetches all the entrants that are stored for a particular event.
51      * @return The collection of entrants.
52      */
53     public ArrayList<Entrant> getEntrants() {
54         return entrants;
55     }
56
57 }

```


6.3.2 Entrant.java

```

1 package aber.dcs.cs22510.clg11.model;
2
3 /**
4  * Defines the data model for an entrant registered for an event.
5  * Allows the setting and retrieval of data about a particular entrant.
6  *
7  * @author Connor Luke Goddard (clg11)
8  * Copyright: Aberystwyth University, Aberystwyth.
9  */
10 public class Entrant {
11
12     private String firstName;
13     private String lastName;
14
15     /** Entrant number used for tracking of entrant. */
16     private int number;
17
18     /** The current progress of the entrant along their registered course. */
19     private int currentProgress;
20
21     /** The ID character of the course the entrant is registered for. */
22     private char courseID;
23
24     /** Defines if the entrant is excluded or not. */
25     private boolean isExcluded = false;
26
27     /** Defines if the entrant has finished or not. */
28     private boolean isFinished = false;
29
30     /** Defines if the entrant is currently at a medical checkpoint. */
31     private boolean atMC = false;
32
33     /**
34      * Default constructor for an Entrant.
35      * Sets the current progress to 0 as a new entrant will
36      * not have started the race.
37      */
38     public Entrant() {
39
40         this.currentProgress = 0;
41
42     }
43
44     /**
45      * Constructor for an Entrant that allows their characteristics to be set upon
46      * instantiation.
47      * @param firstName The first name of the new entrant.
48      * @param lastName The last name of the new entrant.
49      * @param courseID The ID of the course the new entrant is registered for.
50      * @param enNumber The race number of the new entrant.
51      */
52     public Entrant(String firstName, String lastName, char courseID, int enNumber) {
53
54         this.firstName = firstName;
55         this.lastName = lastName;
56         this.courseID = courseID;
57         this.number = enNumber;
58         this.currentProgress = 0;
59
60     }
61
62     /**
63      * Fetches the full name (both first and last) name of the entrant.
64      * @return The full name of the entrant.
65      */
66     public String getFullName() {
67         return getFirstName() + " " + getLastName();
68     }
69
70     /**
71      * Sets the full name of the entrant by splitting the full
72      * name on a space and setting the separate first, and last names.
73      * @param name The inputted full name to be set.
74      */
75     public void setFullName(String name) {
76
77         //Split the inputted name by a space.
78         String[] tempName = name.split(" ");

```

```

79         this.setFirstName(tempName[0]);
80         this.setLastName(tempName[1]);
81     }
82
83     /**
84      * Returns the race number of the entrant.
85      * @return The race number of the entrant.
86      */
87     public int getNumber() {
88         return number;
89     }
90
91     /**
92      * Sets the race number of the entrant.
93      * @param number The race number to be set.
94      */
95     public void setNumber(int number) {
96         this.number = number;
97     }
98
99     /**
100      * Fetches the current progress of the entrant along their course.
101      * @return The current progress of the entrant on their course.
102      */
103     public int getCurrentProgress() {
104         return currentProgress;
105     }
106
107     /**
108      * Updates the current progress of the entrant along their course.
109      * @param currentProgress The incremented progress of the entrant.
110      */
111     public void setCurrentProgress(int currentProgress) {
112         this.currentProgress = currentProgress;
113     }
114
115     /**
116      * Returns the first name of the entrant.
117      * @return The first name of the entrant.
118      */
119     public String getFirstName() {
120         return firstName;
121     }
122
123     /**
124      * Sets the first name only of the entrant.
125      * @param firstName The first name of the entrant to be set.
126      */
127     public void setFirstName(String firstName) {
128         this.firstName = firstName;
129     }
130
131     /**
132      * Returns the last name of the entrant.
133      * @return The last name of the entrant.
134      */
135     public String getLastName() {
136         return lastName;
137     }
138
139     /**
140      * Sets the last name only of the entrant.
141      * @param lastName The last name of the entrant to be set.
142      */
143     public void setLastName(String lastName) {
144         this.lastName = lastName;
145     }
146
147     /**
148      * Fetches the course ID that the entrant is registered for.
149      * @return The ID of the registered course.
150      */
151     public char getCourseID() {
152         return courseID;
153     }
154
155     /**
156      * Sets the course ID of the entrant.
157      * @param courseID The ID of the course that the entrant is registered for.
158      */

```

```

159     public void setCourseID(char courseID) {
160         this.courseID = courseID;
161     }
162
163     /**
164      * Returns whether the entrant is excluded or not.
165      * @return Boolean determining if the entrant is excluded.
166      */
167     public boolean getIsExcluded() {
168         return isExcluded;
169     }
170
171     /**
172      * Sets whether or not the entrant is excluded.
173      * @param isExcluded Whether the entrant is excluded or not.
174      */
175     public void setIsExcluded(boolean isExcluded) {
176         this.isExcluded = isExcluded;
177     }
178
179     /**
180      * Returns whether the entrant has finished their race or not.
181      * @return Boolean determining if the entrant has finished.
182      */
183     public boolean getIsFinished() {
184         return isFinished;
185     }
186
187     /**
188      * Sets whether or not the entrant has finished their race or not.
189      * @param isFinished
190      */
191     public void setIsFinished(boolean isFinished) {
192         this.isFinished = isFinished;
193     }
194
195     /**
196      * Returns if the entrant is currently at a medical checkpoint.
197      * @return Boolean determining if the entrant is at an MC.
198      */
199     public boolean getAtMC() {
200         return atMC;
201     }
202
203     /**
204      * Sets if an entrant is at a medical checkpoint.
205      * @param atMC Whether the entrant is currently at an MC or not.
206      */
207     public void setAtMC(boolean atMC) {
208         this.atMC = atMC;
209     }
210
211 }

```

6.3.3 Course.java

```

1 package aber.dcs.cs22510.clg11.model;
2
3 import java.util.ArrayList;
4
5 /**
6  * Defines the data model for an event course.
7  * Allows the setting and retrieval of data about a particular course.
8  *
9  * @author Connor Luke Goddard (clg11)
10  * Copyright: Aberystwyth University, Aberystwyth.
11  */
12 public class Course {
13
14     /** ArrayList of Nodes that make up the Course. */
15     private ArrayList<Node> courseNodes = new ArrayList<>();
16
17     /** The total length of the course (i.e. the size of the course array).*/
18     private int courseLength;
19
20     /** The unique ID of a particular course. */
21     private char courseID;
22
23     /**
24      * Default constructor for a Course.
25      */
26     public Course() {
27
28     }
29
30     /**
31      * Adds a new {@link aber.dcs.cs22510.clg11.model.Node} to the collection
32      * of nodes that make up the course.
33      * @param newNode The new node to be added to the course.
34      */
35     public void addNewNode(Node newNode) {
36
37         getCourseNodes().add(newNode);
38
39     }
40
41     /**
42      * Fetches the collection of {@link aber.dcs.cs22510.clg11.model.Node}s that
43      * make up the course.
44      * @return The collection of nodes in the course.
45      */
46     public ArrayList<Node> getCourseNodes() {
47         return courseNodes;
48     }
49
50     /**
51      * Fetches the total size of the course.
52      * @return The total size of the ArrayList of Nodes.
53      */
54     public int getCourseLength() {
55         return courseLength;
56     }
57
58     /**
59      * Sets the 'courseLength' value of the course.
60      * @param courseLength The new courselength value.
61      */
62     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

```

1 | package aber.dcs.cs22510.clg11.model;
2 |
3 | /**
4 |  * Defines the data model for a course node within an event.
5 |  * Allows the setting and retrieval of data about a particular course node.
6 |  *
7 |  * @author Connor Luke Goddard (clg11)
8 |  * Copyright: Aberystwyth University, Aberystwyth.
9 |  */
10 | public class Node {
11 |
12 |     /** Type of the node. (CP, MC, JN) */
13 |     private String type;
14 |
15 |     /** The unique node number. */
16 |     private int number;
17 |
18 |     /**
19 |      * Default constructor for an Entrant.
20 |      */
21 |     public Node() {
22 |
23 |     }
24 |
25 |     /**
26 |      * Constructor for a Node that allows their characteristics to be set upon
27 |      * instantiation.
28 |      *
29 |      * @param cpNumber The new node number to be set.
30 |      * @param cpType The node type of the new node
31 |      */
32 |     public Node(int cpNumber, String cpType) {
33 |
34 |         this.number = cpNumber;
35 |         this.type = cpType;
36 |
37 |     }
38 |
39 |     /**
40 |      * Returns the node type of the current node.
41 |      * @return The type of the current node.
42 |      */
43 |     public String getType() {
44 |         return type;
45 |     }
46 |
47 |     /**
48 |      * Set the current node type.
49 |      * @param type The new node type to be set.
50 |      */
51 |     public void setType(String type) {
52 |         this.type = type;
53 |     }
54 |
55 |     /**
56 |      * Returns the ID number of the node.
57 |      * @return The number of the current node.
58 |      */
59 |     public int getNumber() {
60 |         return number;
61 |     }
62 |
63 |     /**
64 |      * Sets the ID number of the current node.
65 |      * @param number the number to set
66 |      */
67 |     public void setNumber(int number) {
68 |         this.number = number;
69 |     }
70 |
71 | }

```

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 |     /**
14 |      *
15 |      */
16 |     COURSE,
17 |     /**
18 |      *
19 |      */
20 |     ENTRANT,
21 |     /**
22 |      *
23 |      */
24 |     NODE
25 | };
```

6.4 ‘GUI’ Package

6.4.1 GUIFrame.java

```

1 | package aber.dcs.cs22510.clg11.gui;
2 |
3 | import aber.dcs.cs22510.clg11.model.Datastore;
4 | import aber.dcs.cs22510.clg11.util.FileIO;
5 | import aber.dcs.cs22510.clg11.util.LoadData;
6 | import java.awt.Dimension;
7 | import java.awt.Toolkit;
8 | import javax.swing.JFrame;
9 |
10 | /**
11 |  * Main JFrame for displaying program GUI. Responsible displaying main GUI
12 |  * window and for instantiating the GUI sub-panel
13 |  * {@link aber.dcs.cs22510.clg11.gui.GUIPanel}. Passes the new {@link aber.dcs.cs22510.clg11.
14 |  *   model.Datastore} &
15 |  *   {@link aber.dcs.cs22510.clg11.util.LoadData} classes received from
16 |  *   {@link aber.dcs.cs22510.clg11.driver.CMDriver}, to the base panel as a
17 |  *   parameter to allow access to the data model from the sub-panel.
18 |  *
19 |  * @author Connor Goddard (clg11) Copyright: Aberystwyth University,
20 |  * Aberystwyth.
21 |  */
22 | public class GUIFrame extends JFrame {
23 |
24 |     /**
25 |      * The new GUIPanel component.
26 |      */
27 |     private GUIPanel panel;
28 |
29 |     /**
30 |      * Constructor to instantiate a new GUIFrame. Takes the two classes created
31 |      * in CMDriver as parameters to pass onto GUI sub-panel.
32 |      *
33 |      * @param newData Datastore class created in main method.
34 |      * @param newLoad LoadData class created in main method.
35 |      * @param newFileIO FileIO classes created in main method.
36 |      */
37 |     public GUIFrame(Datastore newData, LoadData newLoad, FileIO newFileIO) {
38 |
39 |         //Initialise and set up GUI frame (window).
40 |         this.setTitle("Checkpoint Manager | Connor Goddard (clg11)");
41 |         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
42 |
43 |         //Prevent user resizing frame.
44 |         this.setResizable(false);
45 |
46 |         //Initialise the sub-panel, passing the two shared components.
47 |         panel = new GUIPanel(newData, newLoad, newFileIO);
48 |
49 |         //Add this panel to the whole of the frame (No layout set).
50 |         this.add(panel);
51 |
52 |         //Fit frame to ensure all panels/components are visible.
53 |         this.pack();
54 |
55 |         //Determine centre of user's screen and position frame accordingly.
56 |         Toolkit k = Toolkit.getDefaultToolkit();
57 |         Dimension d = k.getScreenSize();
58 |         this.setLocation(d.width / 2 - this.getWidth() / 2, d.height / 2 - this.getHeight() /
59 |             2);
60 |
61 |         //Display frame on screen.
62 |         this.setVisible(true);
63 |     }
64 | }

```

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.Color;
10 import java.awt.Dimension;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import java.text.SimpleDateFormat;
14 import java.util.ArrayList;
15 import java.util.Arrays;
16 import java.util.Calendar;
17 import javax.swing.*;
18 import javax.swing.border.BevelBorder;
19
20 /**
21  * Contains the GUI elements accessed by the user to interact with the
22  * application. Allows the user to select entrants and nodes. Enter a new time
23  * (or use system time) and then submit this new time to the log file. Any
24  * problems that occur will be displayed to the user.
25  *
26  * @author Connor Goddard (clg11)
27  * Copyright: Aberystwyth University,
28  * Aberystwyth.
29  */
30 public class GUIPanel extends JPanel implements ActionListener {
31
32     /**
33      * Buttons that represent system-wide operations.
34      */
35     private JButton submitTime, setCurrentTime;
36     private JLabel nodeTitle, entrantTitle, mcTypeTitle, timeTitle, statusBar;
37     /**
38      * The layout manager used by the panel.
39      */
40     private SpringLayout layout = new SpringLayout();
41     /**
42      * Drop-down selected boxes used to select entrants and nodes.
43      */
44     private JComboBox<String> entrantList;
45     private JComboBox<String> nodeList;
46     /**
47      * Determines whether an entrant is arriving or leaving medical checkpoint.
48      */
49     private JComboBox<String> mcTypeList;
50     private String[] mcArriveDepart = {"Arriving", "Departing"};
51     /**
52      * Spinner used to allow the user to select a time value.
53      */
54     private JSpinner timeSpinner;
55     private SpinnerDateModel sm;
56     private Datastore data;
57     private LoadData load;
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 access 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      * @param newFileIO FileIO object passed down from GUIFrame.
73      */
74     public GUIPanel(Datastore newData, LoadData newLoad, FileIO newFileIO) {
75
76         this.data = newData;
77         this.load = newLoad;
78         this.fileIO = newFileIO;
79
80         //Set the size of the panel
81         this.setPreferredSize(new Dimension(500, 250));
82
83         //Set the bespoke layout manager.
84         this.setLayout(layout);
85
86         //Initialise and add all of the panel GUI components.
87         initComponents();
88

```

```

89         setUpLayout();
90     }
91 }
92
93 /**
94  * Initialises the panel components (including linking components to
95  * listeners where required) before adding the components to the panel.
96  */
97 public void initComponents() {
98
99     String[] comboValues;
100
101     /*
102      * Instantiate new ProcessData class to allow access to data processing
103      * facilities.
104      */
105     proc = new ProcessData(data, fileIO);
106
107     //Create new instance of JLabel with specified display text
108     entrantTitle = new JLabel("Entrant List:");
109     nodeTitle = new JLabel("Checkpoint List:");
110     mcTypeTitle = new JLabel("Medical CP Type:");
111     timeTitle = new JLabel("Log Time:");
112
113     statusBar = new JLabel("Welcome to the Checkpoint Manager.");
114     statusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
115     statusBar.setPreferredSize(new Dimension(500, 20));
116
117     //Load all entrant names into entrant drop-down GUI box component.
118     comboValues = Arrays.copyOf(getAllEntrants().toArray(), getAllEntrants().toArray().length, String[].class);
119
120     entrantList = new JComboBox<>(comboValues);
121     entrantList.setSelectedIndex(0);
122
123     //Load all node numbers into node drop-down GUI box component.
124     comboValues = Arrays.copyOf(getAllCheckpoints().toArray(), getAllCheckpoints().toArray().length, String[].class);
125
126     nodeList = new JComboBox<>(comboValues);
127     nodeList.setSelectedIndex(0);
128
129     //Add local action listener (Required for determining a MC).
130     nodeList.addActionListener(this);
131
132     //Load the MC "arrive/depart" options into drop-down GUI box.
133     mcTypeList = new JComboBox<>(mcArriveDepart);
134     mcTypeList.setSelectedIndex(0);
135     mcTypeList.setEnabled(false);
136
137
138     //Create new instance of JButton with specified button text
139     submitTime = new JButton("Submit Checkpoint Time");
140     submitTime.addActionListener(this);
141
142     setCurrentTime = new JButton("Set to Current Time");
143     setCurrentTime.addActionListener(this);
144
145
146     //Create new JSpinner model that will access the current system time.
147     sm = new SpinnerDateModel();
148     sm.setCalendarField(Calendar.MINUTE);
149
150     //Create a new Spinner object and set the above model to it.
151     timeSpinner = new JSpinner();
152     timeSpinner.setModel(sm);
153
154     //Set the time format to be displayed in the JSpinner.
155     JSpinner.DateEditor de = new JSpinner.DateEditor(timeSpinner, "HH:mm");
156     timeSpinner.setEditor(de);
157
158     mcExclude = new JCheckBox("Exclude Entrant");
159     mcExclude.setSelected(false);
160     mcExclude.setEnabled(false);
161
162     //Add all the components to the GUI panel.
163     this.add(nodeTitle);
164     this.add(entrantTitle);
165     this.add(mcTypeTitle);
166     this.add(timeTitle);

```

```

167         this.add(statusBar);
168         this.add(timeSpinner);
169         this.add(mcExclude);
170         this.add(entrantList);
171         this.add(nodeList);
172         this.add(mcTypeList);
173         this.add(submitTime);
174         this.add(setCurrentTime);
175     }
176 }
177
178 /**
179  * Sets up the 'SpringLayout' layout manager to organise all components on
180  * the panel.
181  */
182 public void setUpLayout() {
183
184     //Set the NORTH edge of the label to be 10 pixels down from the NORTH edge of the panel
185     layout.putConstraint(SpringLayout.NORTH, nodeTitle, 10, SpringLayout.NORTH, this);
186
187     //Set the WEST edge of the label to be 10 pixels left of the WEST edge of the panel.
188     layout.putConstraint(SpringLayout.WEST, nodeTitle, 10, SpringLayout.WEST, this);
189
190     layout.putConstraint(SpringLayout.NORTH, nodeList, 10, SpringLayout.NORTH, this);
191     layout.putConstraint(SpringLayout.WEST, nodeList, 10, SpringLayout.EAST, nodeTitle);
192
193     layout.putConstraint(SpringLayout.NORTH, entrantTitle, 10, SpringLayout.SOUTH,
194         nodeTitle);
195     layout.putConstraint(SpringLayout.WEST, entrantTitle, 10, SpringLayout.WEST, this);
196     layout.putConstraint(SpringLayout.NORTH, entrantList, 10, SpringLayout.SOUTH, nodeTitle
197         );
198     layout.putConstraint(SpringLayout.WEST, entrantList, 10, SpringLayout.EAST,
199         entrantTitle);
200
201     layout.putConstraint(SpringLayout.NORTH, mcTypeTitle, 10, SpringLayout.SOUTH,
202         entrantTitle);
203     layout.putConstraint(SpringLayout.WEST, mcTypeTitle, 10, SpringLayout.WEST, this);
204
205     layout.putConstraint(SpringLayout.NORTH, mcTypeList, 10, SpringLayout.SOUTH,
206         entrantTitle);
207     layout.putConstraint(SpringLayout.WEST, mcTypeList, 10, SpringLayout.EAST, mcTypeTitle)
208         ;
209
210     layout.putConstraint(SpringLayout.NORTH, mcExclude, 10, SpringLayout.SOUTH, mcTypeTitle
211         );
212     layout.putConstraint(SpringLayout.WEST, mcExclude, 10, SpringLayout.WEST, this);
213
214     layout.putConstraint(SpringLayout.NORTH, timeTitle, 10, SpringLayout.SOUTH, mcExclude);
215     layout.putConstraint(SpringLayout.WEST, timeTitle, 10, SpringLayout.WEST, this);
216
217     layout.putConstraint(SpringLayout.NORTH, timeSpinner, 10, SpringLayout.SOUTH, mcExclude
218         );
219     layout.putConstraint(SpringLayout.WEST, timeSpinner, 10, SpringLayout.EAST, timeTitle);
220
221     layout.putConstraint(SpringLayout.NORTH, setCurrentTime, 10, SpringLayout.SOUTH,
222         timeTitle);
223     layout.putConstraint(SpringLayout.WEST, setCurrentTime, 10, SpringLayout.WEST, this);
224
225     layout.putConstraint(SpringLayout.NORTH, submitTime, 10, SpringLayout.SOUTH,
226         setCurrentTime);
227     layout.putConstraint(SpringLayout.WEST, submitTime, 10, SpringLayout.WEST, this);
228
229     layout.putConstraint(SpringLayout.SOUTH, statusBar, 0, SpringLayout.SOUTH, this);
230 }
231
232 /**
233  * Obtains a list of all the entrant names to populate selection box.
234  * Accesses them from the array list of entrants contained within
235  * {@link aber.dcs.cs22510.clg11.model.Datastore}.
236  *
237  * @return ArrayList of all the entrant's names.
238  */
239 public ArrayList<String> getAllEntrants() {
240
241     ArrayList<String> entrantList = new ArrayList<>();
242
243     for (Entrant e : data.getEntrants()) {

```

```

236         entrantList.add(e.getFullName());
237     }
238 }
239
240     return entrantList;
241 }
242
243
244 /**
245  * Obtains a list of all the checkpoints ONLY to populate the CP selection
246  * box. Accesses them from the array list of nodes contained within
247  * {@link aber.dcs.cs22510.clg11.model.Datastore}.
248  *
249  * @return ArrayList of all the nodes that are CHECKPOINTS.
250  */
251 public ArrayList<String> getAllCheckpoints() {
252
253     ArrayList<String> checkpointList = new ArrayList<>();
254
255     //Loop through all the nodes.
256     for (Node cp : data.getNodes()) {
257
258         //If the current node is a checkpoint, and not a junction, add it.
259         if (cp.getType().equals("CP") || cp.getType().equals("MC")) {
260
261             checkpointList.add(Integer.toString(cp.getNumber()));
262
263         }
264     }
265
266     return checkpointList;
267 }
268
269 }
270
271 /**
272  * Attempts to fetch a specific node denoted by the node number selected
273  * from the drop-down GUI box. If such a node cannot be found, NULL is
274  * returned.
275  *
276  * @param nodeNo The number of the selected node.
277  * @return The located node or NULL.
278  */
279 public Node getNode(int nodeNo) {
280
281     for (Node n : data.getNodes()) {
282
283         if (n.getNumber() == nodeNo) {
284
285             return n;
286
287         }
288     }
289
290     return null;
291 }
292
293 /**
294  * Submits a new time log based on user input within the GUI and determines
295  * if an time file currently exists, or if a new one has to be created.
296  */
297 public void submitCheckpoint() {
298
299     try {
300
301
302         //Display question dialog
303         int shouldProcess = JOptionPane.YES_OPTION;
304
305         shouldProcess = JOptionPane.showConfirmDialog(null, "Are you sure you wish to
306             submit this time log?",
307             "CM Manager | Submit Time Log", JOptionPane.YES_NO_OPTION);
308
309         //If user selects "yes"
310         if (shouldProcess == JOptionPane.YES_OPTION) {
311
312             //Create a formatter for the time value entered by the user.
313             SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
314
315             String newTimeValue = sdf.format(timeSpinner.getValue());

```

```

315
316 //Check if a times file currently exists.
317 if (proc.getTimes()) {
318
319     updateStatus(" Times file loaded successfully.", false);
320
321     //Check to see if the time entered by the user is in the past.
322     if (proc.compareTimes(proc.getLastLoggedTime(), newTimeValue)) {
323
324         updateStatus(" ERROR: Time entered is before last recorded time. Please
            try again.", true);
325
326     } else {
327
328         /*
329          * Re-read in the "times" file to allow any new times logged by other
330          * running versions of the checkpoint manager to update the information
331          * contained within this version of the CM.
332          */
333         updateTimeLog();
334     }
335
336     //If a time file does not currently exist, create a new one.
337 } else {
338
339     updateStatus(" ALERT: Times file (times.txt) not found. Creating new time
        log file.", true);
340     updateTimeLog();
341 }
342
343 }
344
345 } catch (IndexOutOfBoundsException iOB) {
346
347     updateStatus(" ERROR: Cannot parse times file. Problem reading file.", true);
348
349 }
350 }
351
352 /**
353  * Takes the user input and processes the new time log entry before adding
354  * it to the times log file.
355  */
356 public void updateTimeLog() {
357
358     String result = null;
359
360     //Obtain the selected entrant from the arraylist of entrants.
361     Entrant currentEntrant = data.getEntrants().get(entrantList.getSelectedIndex());
362
363     //Obtain the arraylist of course nodes that make up the course that entrant is
        registered for.
364     ArrayList<Node> entrantNodes = proc.obtainEntrantCourseNodes(currentEntrant);
365
366     //Create a formatter for the time value entered by the user.
367     SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");
368
369     //Obtain the number of the node selected by the user.
370     int nodeNumber = Integer.parseInt((String) nodeList.getSelectedItem());
371
372     //Format the time value entered by the user.
373     String newTimeValue = sdf.format(timeSpinner.getValue());
374
375     //Check to see if the node selected was a MC.
376     if (mcTypeList.isEnabled()) {
377
378         //If so determine whether they were arriving or departing.
379         String mcSelection = (String) mcTypeList.getSelectedItem();
380
381         /*
382          * Check if the user is attempting to log an entrant arriving
383          * at a MC while the entrant is currently at an MC.
384          */
385         if (currentEntrant.getAtMC() && mcSelection.equals("Arriving")) {
386
387             updateStatus(" ERROR: Entrant " + currentEntrant.getNumber() + " already at
                medical checkpoint.", true);
388
389         } else if (!(currentEntrant.getAtMC()) && mcSelection.equals("Departing")) {
390

```

```

391         updateStatus(" ERROR: Entrant must be at MC before they can depart.", true);
392
393     } else {
394
395         result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber,
                                     mcSelection, newTimeValue, mcExclude.isSelected());
396
397         updateStatus(result, false);
398     }
399
400     } else {
401
402         //The checkpoint is not a MC, and so just process the new logged time.
403         result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber, newTimeValue
                                     );
404         updateStatus(result, false);
405     }
406 }
407
408 /**
409  * Displays system status/error messages to the user via the GUI.
410  * @param updateMessage The message to be displayed.
411  * @param isError Determines whether the message is an error or not.
412  */
413 public void updateStatus(String updateMessage, boolean isError) {
414
415     statusBar.setText(updateMessage);
416
417     if (isError) {
418
419         statusBar.setForeground(Color.RED);
420
421     } else {
422
423         statusBar.setForeground(Color.BLACK);
424     }
425 }
426
427
428 /**
429  * Listener for actions from sub-panel components, to allow operations to be
430  * run when components are interacted with.
431  *
432  * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
433  * @param evt - ActionEvent called from components in the panels that
434  * require an action to be performed.
435  */
436 @Override
437 public void actionPerformed(ActionEvent evt) {
438
439     String actionCommand = evt.getActionCommand();
440
441
442     //Switch statement used to capture action commands from buttons.
443     switch (actionCommand) {
444
445         case "Submit Checkpoint Time":
446
447             //Submit the entered time values.
448             submitCheckpoint();
449             break;
450
451         case "Set to Current Time":
452
453             //Obtain the current system time from the Calendar class.
454             Calendar currentTime = Calendar.getInstance();
455
456             //Update the value of the time spinner to the current time.
457             timeSpinner.setValue(currentTime.getTime());
458
459             updateStatus(" Current time updated successfully.", false);
460             break;
461
462     }
463
464
465     //Listen for events on the nodes drop-down box component.
466     if (evt.getSource() == nodeList) {
467
468         //Obtain the selected Node object from the collection of nodes.

```

```
469 |         Node n = getNode(Integer.parseInt((String) nodeList.getSelectedItem()));
470 |
471 |         //Determine whether the selected node is a medical checkpoint.
472 |         if (n.getType().equals("MC")) {
473 |
474 |             //If it is, allow the "arrive/depart" selection box to be used.
475 |             mcTypeList.setEnabled(true);
476 |             mcExclude.setEnabled(true);
477 |
478 |         } else {
479 |
480 |             mcTypeList.setEnabled(false);
481 |             mcExclude.setEnabled(false);
482 |         }
483 |     }
484 | }
485 | }
```

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 “checkpoint manager” 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

8.1.1 Correct File Parameters

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

Output:

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

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

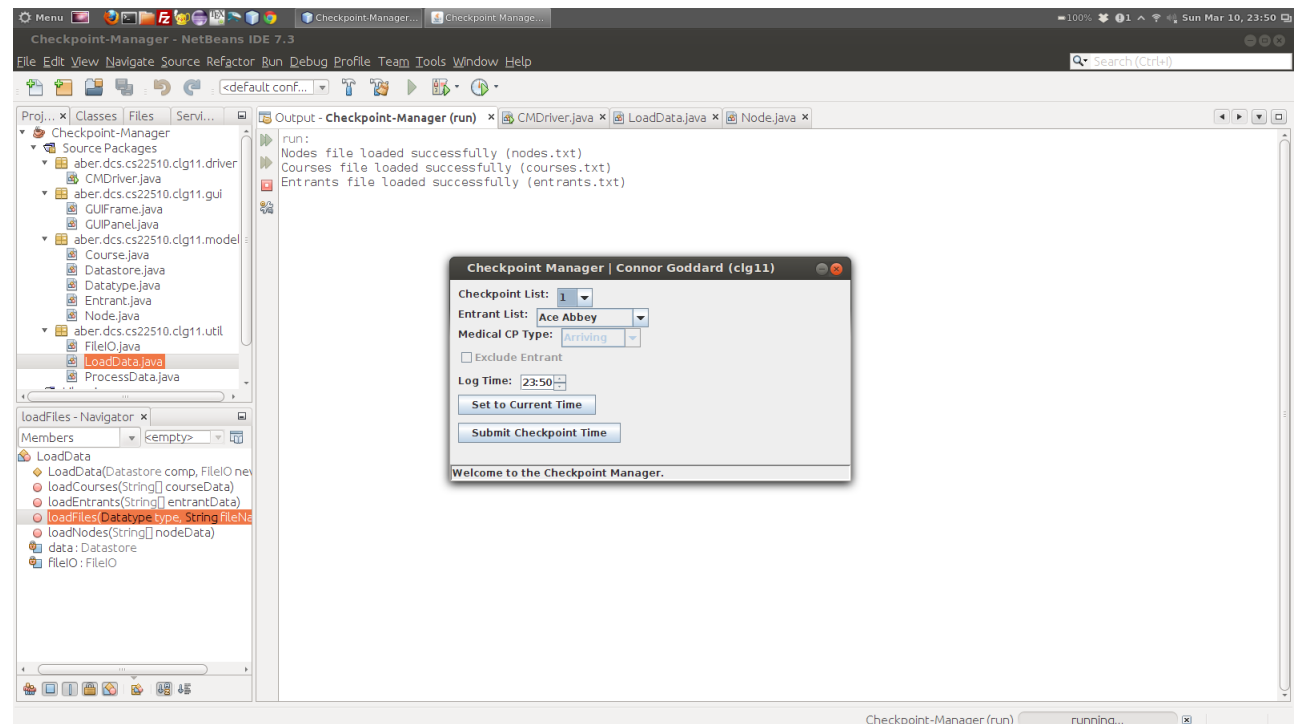


Figure 1: Screenshot displaying the GUI display after successfully loading the external data files.

8.1.2 Incorrect File Parameters

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

Output:

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

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

8.2 Submit Correct Time Entry

Input: Entrant 1 - Checkpoint 1 (Starting checkpoint for course).

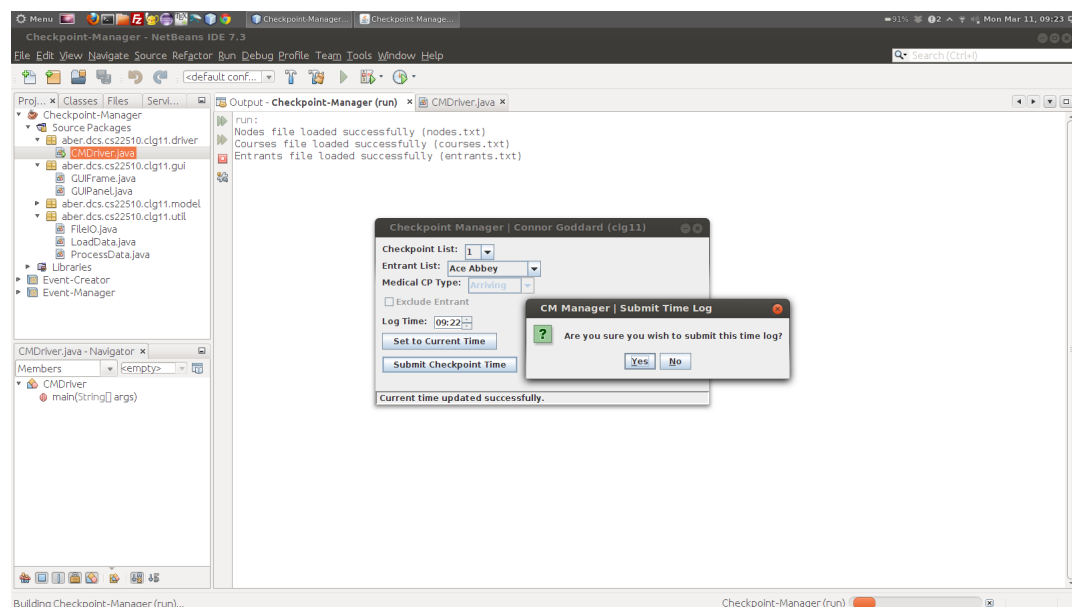


Figure 2: Screenshot displaying attempt to submit a correct time entry for an entrant.

Output:

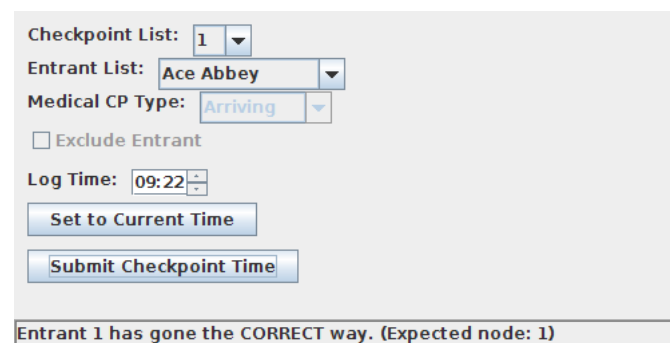


Figure 3: Checkpoint Manager GUI confirming successful submission of time entry.

8.3 Submit Incorrect Time Entry

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

Output:

The screenshot shows the Checkpoint Manager GUI. The 'Checkpoint List' dropdown is set to '4'. The 'Entrant List' dropdown is set to 'Ace Fudge'. The 'Medical CP Type' dropdown is set to 'Arriving'. The 'Exclude Entrant' checkbox is unchecked. The 'Log Time' is set to '09:25'. There are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. Below the form, a message box states: 'Entrant 3 has gone the INCORRECT way. (Expected node: 1)'.

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

8.4 Entrant Exclusion (Incorrect Node)

Input: Entrant 3 - Checkpoint 1.

Output:

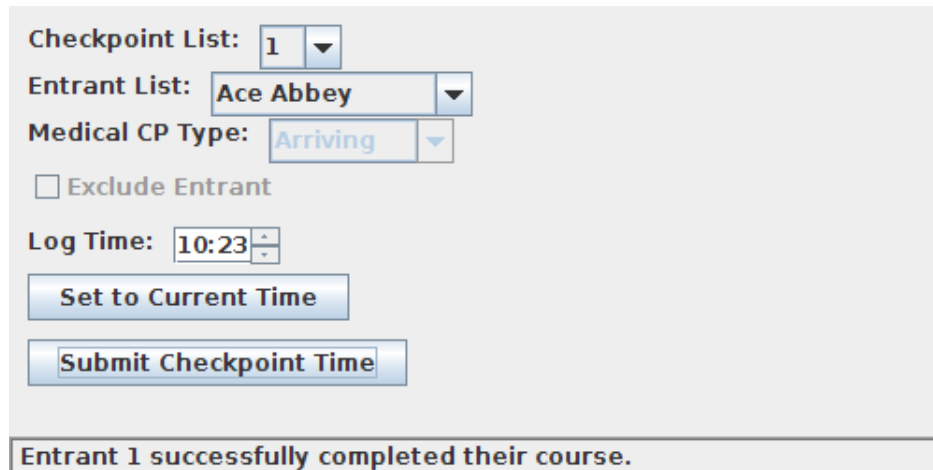
The screenshot shows the Checkpoint Manager GUI. The 'Checkpoint List' dropdown is set to '1'. The 'Entrant List' dropdown is set to 'Ace Fudge'. The 'Medical CP Type' dropdown is set to 'Arriving'. The 'Exclude Entrant' checkbox is unchecked. The 'Log Time' is set to '09:25'. There are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. Below the form, a message box states: 'Entrant 3 has been excluded from their course.'

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

8.5 Entrant Course Completion

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

Output:



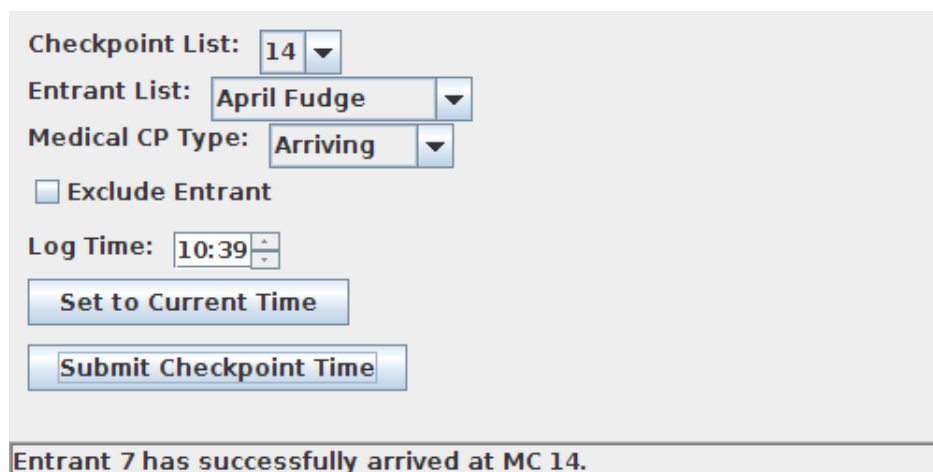
The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 1, Entrant List: Ace Abbey, Medical CP Type: Arriving. The 'Exclude Entrant' checkbox is unchecked. The Log Time is 10:23. There are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a message box states: 'Entrant 1 successfully completed their course.'

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

8.6 Medical Checkpoint - Successful Arrival

Input: Entrant 7 - MC 14 (Arriving).

Output:



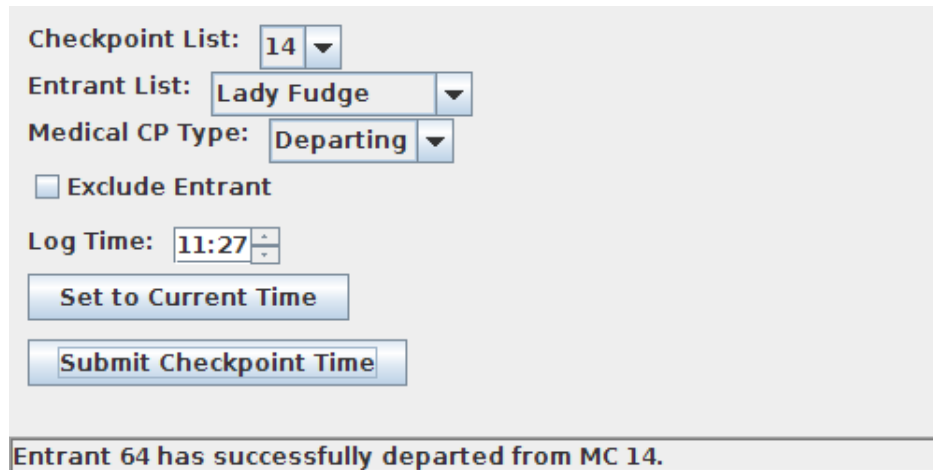
The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 14, Entrant List: April Fudge, Medical CP Type: Arriving. The 'Exclude Entrant' checkbox is unchecked. The Log Time is 10:39. There are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a message box states: 'Entrant 7 has successfully arrived at MC 14.'

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

8.7 Medical Checkpoint - Successful Departure

Input: Entrant 64 - MC 14 (Departing).

Output:



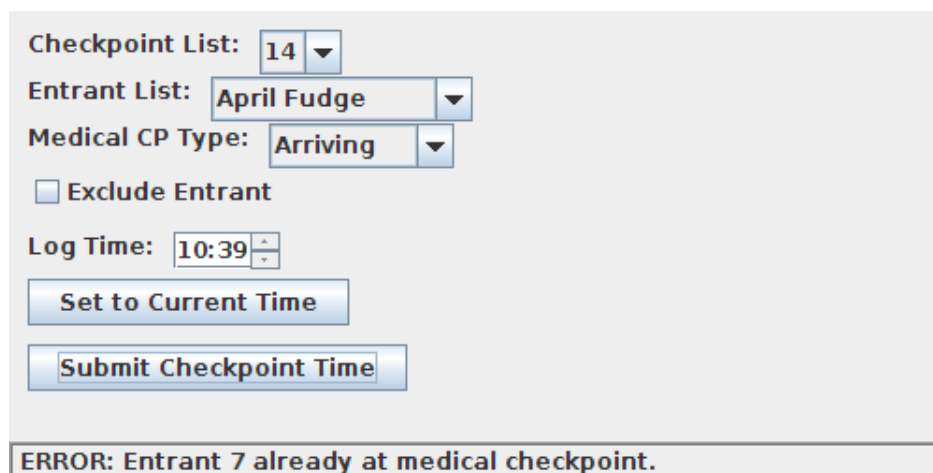
The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 14, Entrant List: Lady Fudge, Medical CP Type: Departing, Exclude Entrant: unchecked, Log Time: 11:27. Below the settings are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a status bar displays the message: 'Entrant 64 has successfully departed from MC 14.'

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

8.8 Medical Checkpoint - Incorrect Arrival

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

Output:



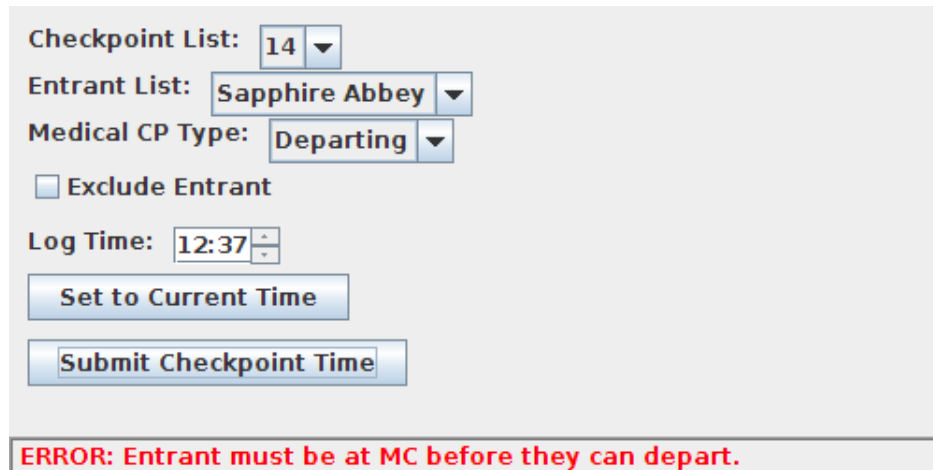
The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 14, Entrant List: April Fudge, Medical CP Type: Arriving, Exclude Entrant: unchecked, Log Time: 10:39. Below the settings are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a status bar displays the message: 'ERROR: Entrant 7 already at medical checkpoint.'

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

8.9 Medical Checkpoint - Incorrect Departure

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

Output:



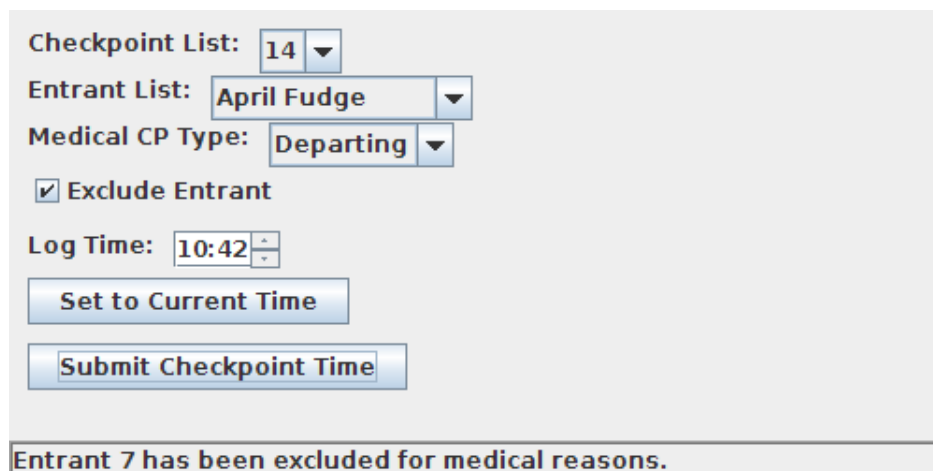
The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 14, Entrant List: Sapphire Abbey, Medical CP Type: Departing, Exclude Entrant: unchecked, Log Time: 12:37. Below the settings are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a red error message reads: 'ERROR: Entrant must be at MC before they can depart.'

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

8.10 Entrant Exclusion (Medical Reasons)

Input: Entrant 7 - MC 14.

Output:



The screenshot shows the Checkpoint Manager GUI with the following settings: Checkpoint List: 14, Entrant List: April Fudge, Medical CP Type: Departing, Exclude Entrant: checked, Log Time: 10:42. Below the settings are two buttons: 'Set to Current Time' and 'Submit Checkpoint Time'. At the bottom, a message reads: 'Entrant 7 has been excluded for medical reasons.'

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

8.11 Submission of Invalid Time Value

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

Output:

The screenshot shows the Checkpoint Manager GUI with the following fields and buttons:

- Checkpoint List: 13 (dropdown)
- Entrant List: Ace Abbey (dropdown)
- Medical CP Type: Arriving (dropdown)
- ☐ Exclude Entrant
- Log Time: 08:46 (time picker)
- Set to Current Time (button)
- Submit Checkpoint Time (button)

At the bottom, a red error message is displayed: **ERROR: Time entered is before last recorded time. Please try again.**

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

8.12 System Activity Logging

Input:

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

Output:

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

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

8.13 File Lock Access Prevention

Input:

Two instances of the Checkpoint Manager application deployed.

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

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

```

1  //Check if the lock was successfull.
2  if (fl != null) {
3
4      try (FileWriter fw = new FileWriter(fos.getFD())) {
5
6          fw.write(output);
7
8          //Once the data has been successfully written, release the lock.
9
10         //FILE LOCK RELEASE PREVENTED.
11         //fl.release();
12     }
13
14     return true;
15
16 }
```

Output:

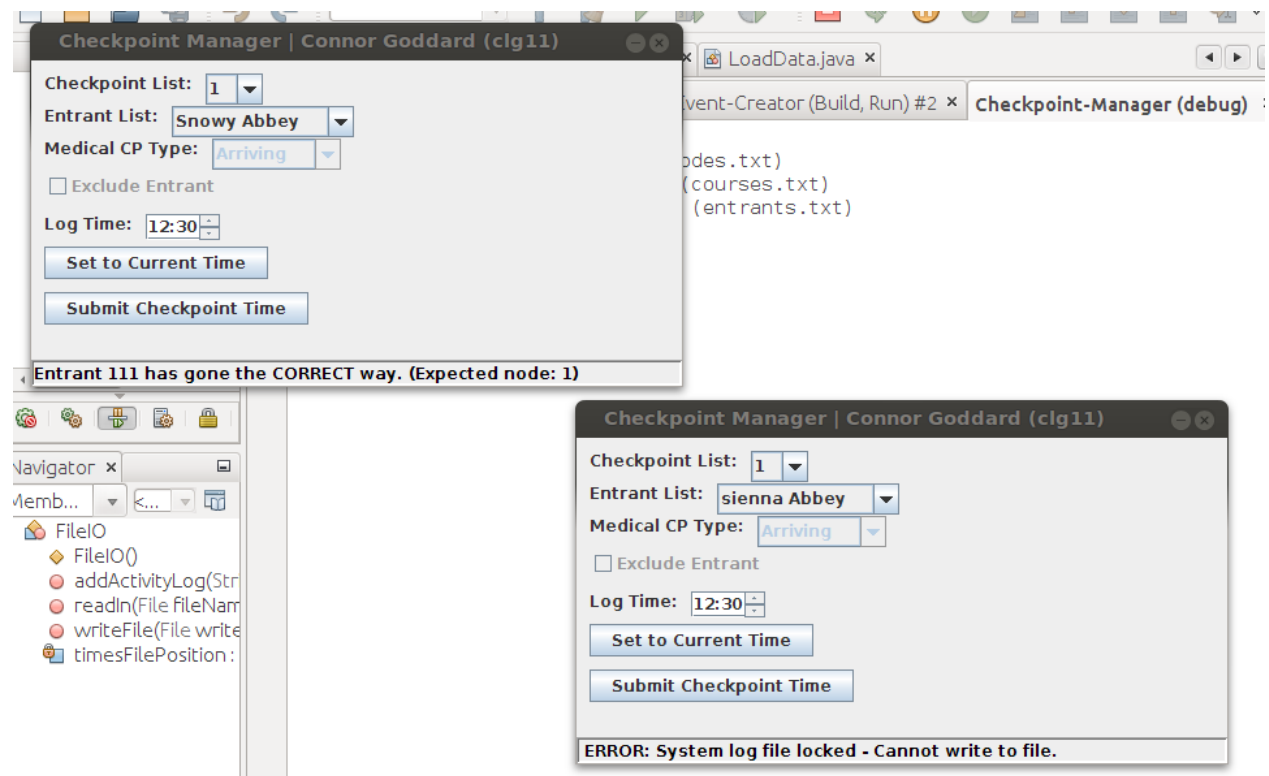


Figure 13: Screenshot demonstrating modified CM application (top-left) preventing original CM application (bottom-right) from accessing log file “log.txt”.

8.14 Entrant Times File Generation

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

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

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

9 Event Manager (C) - Build/Compilation Log

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

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

```

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

```

10 Event Manager - Example Usage

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

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

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

```

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

the test horse event
08 Jul 2013
09:45

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

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

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

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

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

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

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

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

| Competitor No | Competitor Name | Current Status |
|=====|=====|=====|
| 1 | Ace Abbey | Not Started |
|-----|-----|-----|
| 3 | Ace Fudge | Not Started |
|-----|-----|-----|
| 4 | Amber Abbey | Not Started |
|-----|-----|-----|
| 5 | Amber Fudge | Not Started |
|-----|-----|-----|
| 6 | April Abbey | Not Started |
|-----|-----|-----|
| 7 | April Fudge | Not Started |
|-----|-----|-----|
| 8 | Ash Abbey | Not Started |
|-----|-----|-----|

```

```

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

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

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

Total competitors yet to start: 102

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

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

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

No competitors are currently on the course

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

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

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

No competitors have currently finished.

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

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

```

```

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

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

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

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

| Competitor No | Competitor Name | Current Status
|=====|=====|=====|
|
No competitors have currently been excluded.

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

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

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

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

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

| Competitor No | Competitor Name | Current Status
|=====|=====|=====|
| 89 | Prince Abbey | Checkpoint 1
|-----|-----|-----|
| 103 | sienna Abbey | Checkpoint 1
|-----|-----|-----|
| 111 | Snowy Abbey | Checkpoint 1
|-----|-----|-----|

```

```

Total competitors out on course: 3

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

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

| Competitor No | Competitor Name | Current Status |
|=====|=====|=====|
| 1 | Ace Abbey | Finished |
|-----|-----|-----|
| 64 | Lady Fudge | Finished |
|-----|-----|-----|

Total competitors finished: 2

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

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

| Competitor No | Competitor Name | Current Status |
|=====|=====|=====|
| 3 | Ace Fudge | Excluded-IR |
|-----|-----|-----|
| 7 | April Fudge | Excluded-MC |
|-----|-----|-----|

Total competitors excluded: 2

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

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

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

```

```

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

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

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

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

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

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

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

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

Exiting program..

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

```

11 Event Manager - Results Output

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

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

12 Event Manager - System Activity Log

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

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

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

13 System Description

This section provides a general description of the structure and implementation of the separate applications that form the “runners and riders” system.

13.1 Event Creator (C++)

13.2 Checkpoint Manager (Java)

13.3 Event Manager (C)