CS22510 - C, C++ & Java Paradigms

Assignment 1 - Runners & Riders

2012-2013

Connor Luke Goddard
clg11@aber.ac.uk

March 2013

Contents

1 Introduction

1.1 Purpose of this Document

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

1.2 Scope

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

1.3 Objectives

The objectives of this document are:

- To provide the complete source code for the "event creator" application, and evidence of it's compilation and functionality.
- To provide the complete source code for the "checkpoint manager" application, and evidence of it's compilation and functionality.
- To provide evidence of the compilation and functionality of the "event manager" applicaiton.
- To briefly describe the structure and programming language choice of each of the three applications.

2 Event Creator (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

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

#ifndef MENU_H
#define MENU_H

#include "Process.h"
#include "Course.h"
#include "FileIO.h"

/**
 * Used to provide interactive interface with the application through
 * the use of menus.
 */
class Menu {

public:

    Menu(Datastore *newData, Process *newProc);
    Menu(const Menu& orig);
    virtual ~Menu();
    void showEventEditor(void);
    void showCourseEditor(void);
    void showEntrantEditor(void);
    void showMainMenu(void);
    void checkExistingEvent(void);

private:

    /** Pointer to shared Process class created in "main.cpp".*/
    Process *proc;

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

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

#endif  /* MENU_H */
```

2.1.2 Process.h

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

#ifndef PROCESS_H
#define PROCESS_H

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

class Process {

public:

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

private:

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

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

#endif  /* PROCESS_H */
```

2.1.3 Datastore.h

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

#ifndef DATASTORE_H
#define DATASTORE_H

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

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

private:

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

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

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

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

#endif  /* DATASTORE_H */
```

2.1.4 FileIO.h

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

#ifndef FILEIO_H
#define FILEIO_H

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

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

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

#endif  /* FILEIO_H */
```

2.1.5 Event.h

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

#ifndef EVENT_H
#define EVENT_H

#include <string>

/**
 * Event class used to define the data model for a particular event.
 */
class Event {

public:
    Event();
    Event(const Event& orig);
    Event(std::string newEventName, std::string newEventDate, std::string newEventTime);
    virtual ~Event();
    void setEventTime(std::string eventTime);
    std::string getEventTime(void) const;
    void setEventDate(std::string eventDate);
    std::string getEventDate(void) const;
    void setEventName(std::string eventName);
    std::string getEventName(void) const;

private:
    std::string eventName; /**< The name/description of the event.*/
    std::string eventDate; /**< The date that the event is to take place.*/
    std::string eventTime; /**< The starting time of the event.*/
};

#endif  /* EVENT_H */
```

2.1.6 Entrant.h

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

#ifndef ENTRANT_H
#define ENTRANT_H

#include <string>

/**
 * Entrant class used to define the data model for a particular entrant.
 */
class Entrant {

public:
    Entrant(const std::string &theName, const int theEnNo, char theCourseID);
    virtual ~Entrant();
    void print(void) const;
    std::string getEntrantName(void);
    int getEntrantNo(void);
    char getCourseID(void);
private:
    std::string entrant_name; /**< The name of the entrant.*/
    const int entrant_no; /**< The unique number for the entrant.*/
    char course_id; /**< The ID that the entrant is registered for.*/
};

#endif  /* ENTRANT_H */
```

2.1.7 Node.h

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

#ifndef NODE_H
#define NODE_H

#include <string>

/**
 *  Course class used to define the data model for a paritcular course node.
 */
class Node {

public:
    Node();
    Node(const Node& orig);
    Node(const int newNodeNo, std::string newNodeType);
    virtual ~Node();
    void setNodeType(std::string nodeType);
    std::string getNodeType(void) const;
    const int getNodeNo(void) const;

private:
    const int nodeNo; /**< Unique number that represents a node.*/
    std::string nodeType; /**< Contains the type of node.*/
};

#endif  /* NODE_H */
```

2.1.8 Course.h

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

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

#ifndef COURSE_H
#define COURSE_H

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

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

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

#endif  /* COURSE_H */
```

2.2 Class Files

2.2.1 Main.cpp

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

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

using namespace std;

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

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

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

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

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

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

   return 0;
}
```

2.2.2 Menu.cpp

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


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

using namespace std;

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

    data = newData;
    proc = newProc;

}

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

    delete data;
    delete proc;
}

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

    int x;

    while (x != 5) {

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

        cin >> x;

        switch (x) {

            case 1:

                showEventEditor();
                break;

            case 2:

                showEntrantEditor();
                break;

            case 3:

                showCourseEditor();
                break;

            case 4:

                /**
                 * Export all data to their files.
                 * As this method writes ALL the data, it has to check
                 * that at least one instance of each object (Entrant, Event
                 * and Course) exists before being able to write them all to file.
                 */

                cout << "Writing all data to files...\n";

                //Check if an event has been created.
                if (data->getEvent() == NULL) {

                    cout << "ERROR: No event created. Nothing to export.\n";

                    //Check if any entrants have been created.
                } else if (data->getEntrantList().size() <= 0) {

                    cout << "ERROR: No entrants created. Nothing to export.\n";

                    //Check if any courses have been created.
                } else if (data->getCourseList().size() <= 0) {

                    cout << "ERROR: No courses created. Nothing to export.\n";

                } else {

                    //If there are no problems, write all the data to file.
                    io.writeEvent(data->getEvent());
                    io.writeEntrants(data->getEntrantList());
                    io.writeCourses(data->getCourseList());
                }

                break;

            case 5:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Incorrect option. Please try again.\n";
        }
    }
}

/**
 * Provides sub-level interactive menu to allow user to create new
 * courses and write them to file.
 */
void Menu::showCourseEditor(void) {

    int x;

    while (x != 4) {

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

        cin >> x;

        switch (x) {

            case 1:

                proc->createNewCourse();

                break;

            case 2:
            {
                Course *newCourse = NULL;

                //Prompt user for the ID of the course they wish to edit.
                newCourse = proc->getSelectedCourse();

                //If the specified course does not exist..
                if (newCourse == NULL) {

                    //.. inform the user.
                    cout << "ERROR: Course does not exist. Please try again";

                //Otherwise if the course does exist..
                } else {

                    //Prompt the user for the node they wish to add and add it.
                    proc->addCourseNode(newCourse);

                }

                break;
            }
            case 3:

                cout << "Exporting all courses to file.\n";

                //Check if any courses have been created.
                if (data->getCourseList().size() > 0) {
                    io.writeCourses(data->getCourseList());
                } else {
                    cout << "\nERROR: No courses created. Nothing to export.\n";
                }

                break;

            case 4:
                cout << "Returning to main menu...\n";
                break;
            default:
                cout << "Incorrect option. Please try again.\n";
        }
    }
}

/**
 * Provides sub-level interactive menu to allow user to create new
 * entrants and write them to file.
 */
void Menu::showEntrantEditor(void) {

    int x;

    while (x != 3) {

        cout << "\n*************************************\n"
                << "Entrant Editor | Please make a choice:\n"
                << "-------------------------------------\n"
                << "1. Create a new entrant.\n"
                << "2. Export entrants to file.\n"
                << "3. Return to main menu.\n"
                << "*************************************\n";

        cin >> x;

        switch (x) {

            case 1:

                //Flush the input buffer to prevent skipping on "getline()".
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

                //Run method to add a new entrant.
                proc->addEntrant();
                break;

            case 2:

                cout << "Exporting all entrants to file.\n";

                //Check is any entrants have been created.
                if (data->getEntrantList().size() > 0) {
```

```cpp
                    io.writeEntrants(data->getEntrantList());
                } else {
                    cout << "\nERROR: No entrants created. Nothing to export.\n";
                }

                break;

            case 3:
                cout << "Returning to main menu...\n";
                break;
            default:
                cout << "Incorrect option. Please try again.\n";
        }
    }
}

/**
 * Provides sub-level interactive menu to allow user to create a new
 * event and write it's details to file.
 */
void Menu::showEventEditor(void) {

    int x;

    while (x != 3) {

        cout << "\n*************************************\n"
                << "Event Editor | Please make a choice:\n"
                << "-------------------------------------\n"
                << "1. Create new event.\n"
                << "2. Write event to file.\n"
                << "3. Return to main menu.\n"
                << "*************************************\n";

        cin >> x;

        switch (x) {

            case 1:

                //Flush the input buffer to prevent skipping on "getline()".
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

                //Perform check to see if an event has already been created.
                checkExistingEvent();
                break;

            case 2:

                cout << "Exporting event to file.\n";

                //Check to see if an event had been created.
                if (data->getEvent() != NULL) {
                    io.writeEvent(data->getEvent());
                } else {
                    cout << "\nERROR: No event created. Nothing to export.\n";
                }

                break;

            case 3:
                cout << "Returning to main menu...\n";
                break;
            default:
                cout << "Incorrect option. Please try again.\n";
        }
    }
}

/**
 * Checks to see if an existing event has already been created in this session,
 * and provides suitable prompting and error checking as required.
 */
void Menu::checkExistingEvent(void) {

    char input;

    //Check to see if the 'event' pointer in Datastore has been set to an Event object.
    if (data->getEvent() != NULL) {

        //If an event has already been created, prompt user for confirmation.
        while (!((input == 'y') || (input == 'n') || (input == 'Y') || (input == 'N'))) {

            cout << "WARNING: An event has already been created.\n";
            cout << "Do you wish to create a new event? (Y/N)\n";
            cin >> input;

            switch (input) {

                case 'Y':
                case 'y':

                    //Flush the input buffer to prevent skipping on "getline()".
                    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

                    //Prompt user for event information and create the new event.
                    proc->createEvent();
                    break;

                //If the answer is no, nothing needs to happen.
                //Case is left in to prevent system thinking 'n/N' keys are incorrect.
                case 'N':
                case 'n':
                    break;
                default:
                    cout << "Not a valid option. Please try again.\n";
                    break;
            }
        }
    } else {

        //Otherwise if no event has been created as of yet, create a new one.
        proc->createEvent();
    }
}
```

2.2.3 Process.cpp

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

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

using namespace std;

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

    data = newData;

}

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

    delete data;
}

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

    string inputName, inputDate, inputTime, convertedDate;

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

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

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

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

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

    //Create a new Event object on the heap using the inputted information.
    Event *newEvent = new Event(inputName, convertedDate, inputTime);

    //Check if an Event object already exists on the heap.
    if (data->getEvent() != NULL) {

        //If it does remove it to prevent a memory leak.
        delete data->getEvent();

    }

    //Set a pointer to the new object in the shared Datastore class.
    data->setNewEvent(newEvent);

    cout << "\nEvent (" << inputName << ") created successfully.\n";
}

```

```
/**
 * Prompts user for input to define an entrant before creating a new
 * 'Entrant' object and adding it's pointer to the
 * vector of Entrant pointers contained in the shared Datastore class.
 */
void Process::addEntrant(void) {

    string entrantName;
    char courseID, input;
    int entrantNo;

    cout << "Please enter a name: ";
    getline(cin, entrantName);

    //Prompt user to ask if they wish to specify their own entrant number.
    while (!((input == 'y') || (input == 'n') || (input == 'Y') || (input == 'N'))) {

        cout << "Do you wish to set a manual entrant no? (Y/N)";
        cin >> input;

        switch (input) {

            case 'Y':
            case 'y':
            {
                bool notExists = false;

                //Flush the input buffer to prevent input skipping.
                std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

                if (data->getEntrantList().size() > 0) {

                    while (!notExists) {

                        cout << "Please enter an entrant no: ";
                        cin >> entrantNo;

                        //Obtain a vector of ALL the entrants stored in Datastore node vector.
                        std::vector<Entrant*> allEntrants = data->getEntrantList();

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

                            //Check to see if another entrant already has the entered value
                            if ((*it)->getEntrantNo() == entrantNo) {

                                //If so break out of the loop as there should not be ANY
                                    matches.
                                notExists = false;
                                cout << "\nERROR: This entrant already exists. Please enter
                                    another value.\n";
                                break;

                            } else {

                                //Otherwise if there is no match, then we can continue.
                                notExists = true;

                            }
                        }

                    }

                } else {

                    cout << "Please enter an entrant no: ";
                    cin >> entrantNo;

                }

                break;

            }
            case 'N':
            case 'n':

                cout << "Setting automatic entrant number\n";

                //Set entrant number to total numbeer of entrants + 1 (increment).
                entrantNo = (data->getEntrantList().size() + 1);
                break;
            default:
                cout << "Not a valid option. Please try again.\n";
                break;
        }
    }

    //Perform error checking to confirm entered course ID is a letter.
    while (!isalpha(courseID)) {

        cout << "Please enter a course ID: ";
        cin >> courseID;

        //If the user has not entered a letter, they must enter another value.
        if (!isalpha(courseID)) {

            cout << "ERROR: Course ID's can contain letters only. Please try again\n";

        }

    }

    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');

    //Create a new Entrant object on the heap using the inputted information.
    Entrant *emp = new Entrant(entrantName, entrantNo, courseID);

    //Add a pointer to the new object in the entrant vector stored in Datastore.
    data->addNewEntrant(emp);

    cout << "\nEntrant(" << entrantNo << ") created successfully.\n";

}

/**
 * Prompts user for the file path to the file that contains all the course
 * node information, before processing and storing these nodes in a vector
 * contained in the shared Datastore class.
 */
void Process::getAllNodes(void) {

    string fileName;

    //Obtain the file path from the user.
    cout << "Welcome. Please enter the file path for course nodes:\n";
    getline(cin, fileName);

    //Read-in all the node data from the file and store it all in a vector.
    vector<vector<string > > fileContents = io.getFile(fileName);

    //Check if the data has been successfully parsed and read-in.
    if (fileContents.size() <= 0) {

        cout << "ERROR: Nodes file (" << fileName << ") could not be located.\n\n"
                << "Please check the file path and try again. Exiting...\n";

        //If the node data could not be loaded, terminate the program.
        exit(EXIT_FAILURE);

    } else {

        //Loop through every line read-in from the file.
        for (vector<vector<string > >::iterator it = fileContents.begin(); it != fileContents.
             end(); ++it) {

            //Convert the first value on the line (node number) to an int.
            int value = atoi((*it).at(0).c_str());

            //Create a new Node object on the heap using the inputted information.
            Node *tempNode = new Node(value, (*it).at(1));

            //Add a pointer to the new object in the node vector stored in Datastore
            data->addNewNode(tempNode);
        }
        cout << "Course nodes loaded successfully.\n" << "Loading program...\n\n";
    }

}

```

```cpp
/**
 * Prompts user for input to define a course before creating a new
 * 'Course' object and adding it's pointer to the
 * vector of Course pointers contained in the shared Datastore class.
 */
void Process::createNewCourse(void) {

    char cid;
    bool notExists = false;

    if (data->getCourseList().size() > 0) {

        while (!notExists) {

            cid = 0;

            //Check that the ID inputted by the user is a letter.
            while (!isalpha(cid)) {

                //Prompt user for a course ID.
                cout << "Please enter a new course ID: ";
                cin >> cid;

                if (!isalpha(cid)) {

                    cout << "ERROR: Course ID's can contain letters only. Please try again\n";

                }

            }

            //Obtain a vector of ALL the courses stored in Datastore node vector.
            std::vector<Course*> allCourses = data->getCourseList();

            //Loop through all the stored courses.
            for (vector<Course*>::iterator it = allCourses.begin(); it != allCourses.end(); ++
                 it) {

                //Check to see if another course already has the entered value.
                if ((*it)->getCourseID() == cid) {

                    notExists = false;

                    //If so break out of the loop as there should not be ANY matches.
                    cout << "\nERROR: This course already exists. Please enter another value.\n
                        ";
                    break;

                } else {

                    //Otherwise if there is no match, then we can continue.
                    notExists = true;

                }
            }
        }

    } else {

        //Check that the ID inputted by the user is a letter.
        while (!isalpha(cid)) {

            //Prompt user for a course ID.
            cout << "Please enter a new course ID: ";
            cin >> cid;

            if (!isalpha(cid)) {

                cout << "ERROR: Course ID's can contain letters only. Please try again\n";

            }

        }
    }

    //Create a new Course object on the heap using the inputted information.
    Course *newCourse = new Course(cid);

    //Add a pointer to the new object in the course vector stored in Datastore.
    data->addNewCourse(newCourse);

    cout << "\nCourse (" << cid << ") created successfully.\n";
}

/**
 * Allows a user to specify a new node (loaded in from "nodes.txt") that
 * that will form part of a particular course.
 * @param currentCourse The course that a user wishes to add a new node to.
 */
void Process::addCourseNode(Course *currentCourse) {

    int nodeNo;

    cout << "Please select a node to add: \n";

    //Obtain a vector of ALL the course nodes stored in Datastore node vector.
    std::vector<Node*> allNodes = data->getNodeList();

    //Print all the nodes to screen to provide user with a list to select from.
    for (std::vector<Node*>::iterator it = allNodes.begin(); it != allNodes.end(); ++it) {
        cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << "), ";
    }

    cout << endl;

    //Prompt user for the number of the node they wish to add.
    cin >> nodeNo;

    //Attempt to fetch the specified node from the vector of nodes in Datastore.
    Node *tempNode = data->obtainNode(nodeNo);

    //Check to see if a matching node was located.
    if (tempNode != NULL) {

        /**
         * Add a pointer to the located node object in the course's vector
         * of nodes.
         */
        currentCourse->addCourseNode(tempNode);
        cout << "\nNode (" << tempNode->getNodeNo() << ") added successfully.\n";


        //Obtain the vector of all nodes contained within the course.
        std::vector<Node*> currentCourseNodes = currentCourse->getCourseNodes();

        //Display a list of all the nodes that make up the course on screen.
        cout << "\nCurrent nodes contained in Course (" << currentCourse->getCourseID() << "):\
            n";

        for (std::vector<Node*>::iterator it = currentCourseNodes.begin(); it !=
             currentCourseNodes.end(); ++it) {
            cout << (*it)->getNodeNo() << " (" << (*it)->getNodeType() << ")\n";
        }

    } else {

        //Otherwise if no matching node can be found, inform the user.
        cout << "\nERROR: Node " << nodeNo << " not found.\n";

    }

}

/**
 * Attempts to locate a course from Datastore that matches
 * the ID entered by the user.
 * @returns The pointer to a matching course or NULL.
 */
Course* Process::getSelectedCourse(void) {

    char selectedID;

    //Check that the ID inputted by the user is a letter.
    while (!isalpha(selectedID)) {

        //Prompt user for a course ID.
        cout << "Please enter an existing course ID: ";
        cin >> selectedID;

        if (!isalpha(selectedID)) {

            cout << "ERROR: Course ID's can contain letters only. Please try again\n";

        }

    }

    //Return the matching course fetched from the course vector in Datastore.
    return data->getInCourse(selectedID);

}

/**
 * Converts an inputted date string from "DD/MM/YYYY" to correct format
 * "%d %b %Y" (e.g. 05 July 1993) required to write event details to file.
 * @param input The original inputted event date in format "DD/MM/YYYY".
 * @returns A string containing the same date in a modified format.
 */
string Process::convertDate(string &input) {

    string convertDate, result;

    //Resize the conversion string to the same sized as the original.
    convertDate.resize(input.size());

    //Remove any '/' characters from the original string and pass to new string.
    remove_copy(input.begin(), input.end(), convertDate.begin(), '/');

    //Create new string stream and input modified date string into it.
    ostringstream date1;
    date1 << convertDate;

    //Create new time structure used to process date conversion.
    struct tm tm;
    strptime(date1.str().c_str(), "%d%m%Y", &tm);

    char date2[30];

    //Re-format the date into the correct format.
    strftime(date2, sizeof (date2), "%d %b %Y", &tm);

    //Set a resulting string to the output of the re-arranged time struct.
    result = string(date2);

    //Return the re-formatted date string.
    return result;

}
```

2.2.4 Datastore.cpp

```
/*
 * File: Datastore.cpp
 * Description: Contains and stores all the persistent data used
 * by the application to allow data to be accessed by multiple classes.
 * Author: Connor Luke Goddard (clg11)
 * Date: March 2013
 * Copyright: Aberystwyth University, Aberystwyth
 */

#include "Datastore.h"

/**
 * Default constructor for Datastore.
 * Sets the initial value of the 'event' pointer to NULL for
 * error checking purposes.
 */
Datastore::Datastore() {

    event = NULL;
}

/**
 * Destructor to be used once object is removed.
 */
Datastore::~Datastore() {
    delete event;
}

/**
 * Fetches the vector of all the courses created for an event.
 * @return A vector that contains pointers to all the Course objects created.
 */
std::vector<Course*> Datastore::getCourseList(void){
    return courseList;
}

/**
 * Fetches the vector of all the nodes read in from "nodes.txt".
 * @return A vector that contains pointers to all the Node objects.
 */
std::vector<Node*> Datastore::getNodeList(void) {
    return nodeList;
}

/**
 * Fetches the vector of all the entrants created for an event.
 * @return A vector that contains pointers to all the Entrant objects created.
 */
std::vector<Entrant*> Datastore::getEntrantList(void){
    return entrantList;
}

/**
 * Fetches the Event object created to define the race event.
 * @return A a pointer to the created Event object.
 */
Event* Datastore::getEvent(void) const {
    return event;
}

/**
 * Adds a new Course object to the end of the 'courseList' vector.
 * @param newCourse Pointer to the new Course object to be added to the vector.
 */
void Datastore::addNewCourse (Course *newCourse) {

    courseList.push_back(newCourse);

}

/**
 * Adds a new Node object to the end of the 'nodeList' vector.
 * @param newNode Pointer to the new Node object to be added to the vector.
 */
void Datastore::addNewNode (Node  *newNode) {

    nodeList.push_back(newNode);

```

```cpp
}

/**
 * Adds a new Entrant object to the end of the 'courseEntrant' vector.
 * @param newEntrant Pointer to the new Entrant object to be added to the vector.
 */
void Datastore::addNewEntrant (Entrant *newEntrant) {

    entrantList.push_back(newEntrant);

}

/**
 * Sets the 'event' pointer to a newly created Event object.
 * @param newEvent A pointer to the new Event object created.
 */
void Datastore::setNewEvent(Event *newEvent) {

    this->event = newEvent;

}

/**
 * Determines if a course with the inputted ID exists in the vector of
 * courses ('courseList') and if so returns the pointer to that Course object.
 * @param selectedID The course ID inputted by the user.
 * @return Either the located course or NULL.
 */
Course* Datastore::getInCourse (char selectedID) {

    //Loop through the entire vector of courses.
    for (std::vector<Course*>::iterator it = courseList.begin(); it != courseList.end(); ++it)
         {

        //If the ID of the current course matches the inputted ID...
        if ((*it)->getCourseID() == selectedID) {

            //... return the pointer to that Course object.
            return (*it);
        }
    }

    //Otherwise if no matches are found, return NULL.
    return NULL;
}

/**
 * Determines if a node with the inputted number exists in the vector of
 * nodes ('nodeList') and if so returns the pointer to that Node object.
 * @param nodeNo The node number inputted by the user.
 * @return Either a pointer to the located Node object or NULL.
 */
Node* Datastore::obtainNode (int nodeNo) {

    //Loop through the entire vector of courses.
    for (std::vector<Node*>::iterator it = nodeList.begin(); it != nodeList.end(); ++it) {

        //If the number of the current node matches the inputted number...
        if ((*it)->getNodeNo() == nodeNo) {

            //... return the pointer to that Node object.
            return (*it);
        }

    }

    //Otherwise if no matches are found, return NULL.
    return NULL;
}
```

2.2.5 FileIO.cpp

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

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

using namespace std;

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

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

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

    //Create a new file stream.
    ofstream myfile;

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

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

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

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

}

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

    ofstream myfile;
    myfile.open("../files/examplecourses.txt", ios::out | ios::app);

    //Loop through all the the courses in the vector.
    for (vector<Course*>::iterator it = courseList.begin(); it != courseList.end(); ++it) {

        //Write the current course ID and total course size to file.
        myfile << (*it)->getCourseID() << " " << (*it)->getCourseSize() << " ";

        //Create a temporary array of current course nodes.
        vector<Node*> currentCourseNodes = (*it)->getCourseNodes();

        //Loop through all nodes that make up the current course.
        for (vector<Node*>::iterator jt = currentCourseNodes.begin(); jt != currentCourseNodes.
            end(); ++jt) {

            //Write the node number to the file.
            myfile << (*jt)->getNodeNo() << " ";
        }

        myfile << "\n";
    }

    myfile.close();

}

/**
 * Writes the details of a particular event to file using a
 * specified format. Event information is obtained from the 'event' pointer
 * stored within the Datastore class.
 * @param event Pointer to the stored Event class.
 */
void FileIO::writeEvent(Event *event) {

    ofstream myfile;
    myfile.open("../files/exampleevent.txt", ios::out | ios::trunc);

    myfile << (*event).getEventName() << "\n" << (*event).getEventDate() << "\n" << (*event).
        getEventTime() << "\n";

    myfile.close();

}

/**
 * Accesses a specified file and returns the contents as a vector.
 * @param fileName The file path of the specified file.
 * @return A vector of vectors that each contain the contents of each line
 * of the file that was read in.
 */
vector<vector<string > > FileIO::getFile(string fileName) {

    string line;

    //Create a new file stream.
    ifstream myfile(fileName.c_str());

    //Check the file has been successfully opened.
    if (myfile.is_open()) {

        //Read the entire contents of the file.
        while (std::getline(myfile, line)) {

            //Split the current line by white space into separate tokens.
            istringstream ss(line);
            istream_iterator<string> begin(ss), end;

            //Place all the tokens into a new vector (For the line).
            vector<string> allStrings(begin, end);

            //Add this vector to the parent vector for the whole file.
            arrayTokens.push_back(allStrings);

        }

        myfile.close();
    }

    //Return the vector. If the loading was un-successful, it will be empty.
    return arrayTokens;
}
```

2.2.6 Event.cpp

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

#include "Event.h"

using namespace std;

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

}

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

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

}

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

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

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

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

/**
 * Fetches the date of the event.
 * @return The value of the 'eventDate' string variable.
 */
string Event::getEventDate(void) const {
    return eventDate;
}

/**
 * Updates the name/description of the event to an inputted value.
 * @param eventTime Recently inputted event name/description..
 */
void Event::setEventName(string eventName) {
    this->eventName = eventName;
}
```

```

/**
 * Fetches the name/description of the event.
 * @return The value of the 'eventName' string variable.
 */
string Event::getEventName(void) const {
    return eventName;
}
```

2.2.7 Entrant.cpp

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

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

using namespace std;


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

    course_id = theCourseID;
}

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

}

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

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

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

2.2.8 Node.cpp

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

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

using namespace std;

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

    setNodeType(newNodeType);

}

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


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

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

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

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

2.2.9 Course.cpp

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

#include "Course.h"

/**
 * Constructor that allows the constant 'courseID' variable
 * to be specified. Also defaults the size of a course to 0.
 * @param theCourseID The new course ID value to be set.
 */
Course::Course(const char theCourseID) : courseID(theCourseID) {
    courseSize = 0;
}

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

}

/**
 * Adds a new node to the end of the 'courseNode' vector.
 * @param newNode Pointer to the new node to be added to the vector.
 */
void Course::addCourseNode(Node *newNode) {

    this->courseNodes.push_back(newNode);
    this->setCourseSize(courseNodes.size());

}

/**
 * Updates the total size of the course. (i.e. vector size).
 * @param courseSize The new size value.
 */
void Course::setCourseSize(int courseSize) {
    this->courseSize = courseSize;
}

/**
 * Fetches the value of 'courseSize'.
 * @return The total number of nodes in the course.
 */
int Course::getCourseSize(void) const {
    return courseSize;
}

/**
 * Fetches a vector of all the nodes in the course.
 * @return A vector of nodes that make up the course.
 */
std::vector<Node*> Course::getCourseNodes(void) const {
    return courseNodes;
}

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

3 Event Creator - Build/Compilation Log

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

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

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


CLEAN SUCCESSFUL (total time: 57ms)

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


BUILD SUCCESSFUL (total time: 3s)
```

4 Event Creator - Example Usage

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

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

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

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

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


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


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

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

ERROR: No event created. Nothing to export.

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

Event (the test running event) created successfully.

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

Event (the test horse event) created successfully.

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

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

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

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

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

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

Course (U) created successfully.

*************************************
Course Editor | Please make a choice:
-------------------------------------
1. Create a new course.
2. Add a new node to existing course.
3. Export courses to file.
4. Return to main menu.
*************************************
2
Please enter an existing course ID: U
Please select a node to add:
1 (CP), 2 (JN), 3 (JN), 4 (CP), 5 (CP), 6 (JN), 7 (CP), 8 (JN), 9 (CP), 10 (JN), 11 (JN), 12 (
    JN), 13 (CP), 14 (MC), 15 (JN), 16 (JN), 17 (CP), 18 (JN),
45

ERROR: Node 45 not found.

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

Node (1) added successfully.

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

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

Node (3) added successfully.

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

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

Node (11) added successfully.

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

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

Node (18) added successfully.

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

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

Course (C) created successfully.

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

Node (5) added successfully.

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

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

Node (11) added successfully.

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

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

Node (7) added successfully.

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

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

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

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

Entrant(1) created successfully.

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

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

Entrant(13) created successfully.

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

```
Please enter a course ID: U

Entrant(3) created successfully.

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

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

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

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

5 Event Creator - File Output

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

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

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

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

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

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

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

6 Checkpoint Manager (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

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

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

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

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

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

        //Instantiate new FileIO object to allow shared file I/O facilities.
        FileIO fileIO = new FileIO();

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


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

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

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

       } catch (IndexOutOfBoundsException eX) {

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

    }
}
```

6.2 ‘Util’ Package

6.2.1 ProcessData.java

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

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

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

    private Datastore data;
    private FileIO fileIO;
    private String lastLoggedTime = null;

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

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

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

    }

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

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

        for (Entrant e : data.getEntrants()) {

            if (e.getNumber() == requiredEntrant) {

                return e;
            }

        }

        return null;
    }

    /**
     * Attempts to fetch the collection of course nodes that make up the course
     * that a specified entrant is registered for.
     *
     * @param selectedEntrant The number of the required entrant.
     * @return The collection of course nodes, or NULL.
     */
    public ArrayList<Node> obtainEntrantCourseNodes(Entrant selectedEntrant) {

        //Loop through all the stored courses.
        for (Course c : data.getCourses()) {

            //If the current course matches the entrant's course.
            if (c.getCourseID() == selectedEntrant.getCourseID()) {

                //Return the collection of nodes for that course.
                return c.getCourseNodes();
            }

        }

        //Otherwise if nothing is found, return NULL.
        return null;
    }

    /**
     * Processes each line read in from the "times.txt" file to update the
     * internal record of entrant's progress. This method is crucial to ensure
     * that any time log updates created by any other running versions of the
     * checkpoint manager are recorded in the internal entrant record within
     * this application.
     *
     * @param timeDelimiter The character symbol used to represent the status of
     * the particular time log.
     * @param nodeNo The number of the node the time log was recorded at.
     * @param entrantNo The number of the entrant that was recorded.
     * @throws IndexOutOfBoundsException
     */
    public void processNewTime(String timeDelimiter, int nodeNo, int entrantNo) throws
         IndexOutOfBoundsException {

        //Boolean used to determine whether this particular time log has been processed.
        boolean isUpdated = false;

        //Obtain the required entrant from the internal collection of entrants.
        Entrant currentEntrant = obtainEntrant(entrantNo);

        //Check if the time log dictates that the entrant should be excluded.
        if (timeDelimiter.equals("I") || timeDelimiter.equals("E")) {

            //If so exclude the entrant.
            excludeEntrant(entrantNo);

            //Log this activity in the log file ("log.txt");
            fileIO.addActivityLog("Entrant no: " + entrantNo + " successfully excluded.");

            /*
             * Otherwise if they shouldn't be excluded,
             * check to see if the entrant has already been excluded.
             */
        } else if (!currentEntrant.getIsExcluded()) {

            ArrayList<Node> courseNodes = obtainEntrantCourseNodes(currentEntrant);

            //Loop through all the nodes that make up the course the entrant is on.
            for (int i = 0; i < courseNodes.size(); i++) {

                /*
                 * Check that the current progress of the entrant < the index of
                 * the current node in the array (to prevent nodes the entrant has
                 * already passed being used again), and the current node equals
                 * the node number of the current time log.
                 */
                if (i > (currentEntrant.getCurrentProgress() - 1) && courseNodes.get(i).
                    getNumber() == nodeNo && !isUpdated) {

                    /*
                     * If the entrant has ARRIVED at a medical checkpoint,
                     * their progress should not be incrememted as they are
                     * now waiting at the MC
                     */
                    if (timeDelimiter.equals("A")) {

                        //Just prevent this particular time log being processed any further.
                        currentEntrant.setAtMC(true);
                        isUpdated = true;

                        /*
                         * Otherwise, if they are DEPARTING from a MC or they
                         * have just arrived at a normal checkpoint, then their
                         * progress needs to be recorded and incremented.
                         */
                    } else {

                        /*
                         * If the read in node from time file is further along
                         * the course than the current progress,
                         * update the current progress.
                         */
                        currentEntrant.setCurrentProgress((i + 1));
                        currentEntrant.setAtMC(false);


                        /*
                         * Check to see if the entrant has now completed
                         * their course.
                         */
                        if (currentEntrant.getCurrentProgress() >= courseNodes.size()) {

                            //Log that they have finished.
                            currentEntrant.setIsFinished(true);

                            //Log this activity in the log file ("log.txt");
                            fileIO.addActivityLog("Entrant no: " + currentEntrant.getNumber() +
                                 " has sucessfully finished the course.");
                        }

                        isUpdated = true;

                    }
                }
            }
        }
    }

    /**
     * Updates a particular Entrant object to log the fact that they have been
     * excluded from their race.
     *
     * @param entrantNo The number of the required entrant.
     */
    public void excludeEntrant(int entrantNo) {

        for (Entrant e : data.getEntrants()) {

            if (e.getNumber() == entrantNo) {

                e.setIsExcluded(true);
            }

        }

    }

    /**
     * Processes a new time log submitted by the user by determining whether the
     * entrant is on the correct path or not and updates the "times.txt" file
     * with the resulting time log.
     *
     * @param courseNodes The collection of nodes that make up the course the
     * current entrant is registered for.
     * @param selectedEntrant The current entrant being processed.
     * @param newNode The newly submitted node that the entrant has arrived at.
     * @param time The inputted time of the entrant's arrival at the CP.
     * @return
     */
    public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int
         newNode, String time) {

        //Obtain the current progress of the entrant (i.e. the index of the array).
        int nextNodeIndex = selectedEntrant.getCurrentProgress();
        String result = null;
        boolean timesNotLocked = true;
        boolean logNotLocked = true;

        //Check that the entrant has not already finished, or been excluded.
        if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {

            result = " Entrant " + selectedEntrant.getNumber() + " successfully completed their
                 course.";

        } else if (selectedEntrant.getIsExcluded()) {

            result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from their
                 course.";

        } else {

            /*
             * Check whether the next node in the array (i.e. the next node that the
             * entrant SHOULD have reached) is actually the node sumbitted.
             */
            if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {

                /*
                 * If they do not match, the entrant has gone the wrong way.
                 * Append this new time log with the 'I' time delimter to the
                 * times file ("times.txt").
                 */
                timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " +
                     newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                     incorrect for course. (Entrant No: " + selectedEntrant.getNumber() + ")");

                result = "Entrant " + selectedEntrant.getNumber()
                        + " has gone the INCORRECT way. (Expected node: " + courseNodes.get(
                             nextNodeIndex).getNumber() + ")";

            } else {

                /*
                 * Otherwise if they do  match, the entrant has gone the right way.
                 * Append this new time log with the 'T' time delimter to the
                 * times file ("times.txt").
                 */
                timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "T " +
                     newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                logNotLocked = fileIO.addActivityLog("Submitted checkpoint " + newNode + "
                     incorrect for course. (Entrant No: " + selectedEntrant.getNumber() + ")");

                result = "Entrant " + selectedEntrant.getNumber()
                        + " has gone the CORRECT way. (Expected node: " + courseNodes.get(
                             nextNodeIndex).getNumber() + ")";
            }

        }

        /*
         * If any of the output files are locked by another process/application,
         * inform the user.
         */
        if (!logNotLocked) {

            result = " ERROR: System log file locked - Cannot write to file.";

        }

        if (!timesNotLocked) {

```

```java
            result = " ERROR: Times log file locked - Cannot write to file. Please try again.";

        }

        if (!timesNotLocked && !logNotLocked) {

           result = " ERROR: Cannot write to time log or log file. - Both files locked.";
        }

        return result;

    }

    /**
     * Processes a new time log submitted by the user by determining whether the
     * entrant is on the correct path or not and updates the "times.txt" file
     * with the resulting time log (Overloaded method for processing medical
     * checkpoints).
     *
     * @param courseNodes The collection of nodes that make up the course the
     * current entrant is registered for.
     * @param selectedEntrant The current entrant being processed.
     * @param newNode The newly submitted node that the entrant has arrived at.
     * @param mcType Whether the entrant was arriving or departing from the MC.
     * @param time The inputted time of the entrant's arrival at the CP.
     * @param isExcluded
     * @return String containing the result of processing the time log.
     */
    public String processTimeLog(ArrayList<Node> courseNodes, Entrant selectedEntrant, int newNode, String mcType, String time, boolean isExcluded) {

        int nextNodeIndex = selectedEntrant.getCurrentProgress();
        String result = null;
        boolean timesNotLocked = true;
        boolean logNotLocked = true;

        //Check that the entrant has not already finished, or been excluded.
        if (selectedEntrant.getCurrentProgress() >= courseNodes.size()) {

            result = " Entrant " + selectedEntrant.getNumber() + " successfully completed their course.";

        } else if (selectedEntrant.getIsExcluded()) {

            result = " Entrant " + selectedEntrant.getNumber() + " has been excluded from their course.";

        } else {

            if (courseNodes.get(nextNodeIndex).getNumber() != newNode) {

                logNotLocked = fileIO.addActivityLog("Submitted checkpoint incorrect for course. (Entrant No: " + selectedEntrant.getNumber() + ")");

                timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "I " + newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                result = "Entrant " + selectedEntrant.getNumber()
                        + " has gone the wrong way. (Expected node: " + courseNodes.get(nextNodeIndex).getNumber() + ")";

            } else if (isExcluded) {

                logNotLocked = fileIO.addActivityLog("Entrant excluded for medical reasons. (Entrant No: " + selectedEntrant.getNumber() + ")");

                timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "E " + newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                result = "Entrant " + selectedEntrant.getNumber()
                        + " has been excluded for medical reasons.";

            } else {

                /*
                 * If they do match, the entrant has gone the right way.
                 * Determine whether the entrant was arriving at, or departing
                 * from the MC and update the time log file ("times.txt")
                 * accordingly.
                 */
                if (mcType.equals("Arriving")) {

                    logNotLocked = fileIO.addActivityLog("New MC arrival time submitted. (
                        Entrant No: " + selectedEntrant.getNumber() + ")");

                    timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "A " +
                        newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                    result = "Entrant " + selectedEntrant.getNumber()
                            + " has successfully arrived at MC " + courseNodes.get(
                                nextNodeIndex).getNumber() + ".";

                } else {

                    logNotLocked = fileIO.addActivityLog("New MC departure time submitted. (
                        Entrant No: " + selectedEntrant.getNumber() + ")");
                    timesNotLocked = fileIO.writeFile(new File("../files/times.txt"), "D " +
                        newNode + " " + selectedEntrant.getNumber() + " " + time + "\n");

                    result = "Entrant " + selectedEntrant.getNumber()
                            + " has successfully departed from MC " + courseNodes.get(
                                nextNodeIndex).getNumber() + ".";
                }
            }

        }

        /*
         * If any of the output files are locked by another process/application,
         * inform the user.
         */
        if (!timesNotLocked) {

            result = " ERROR: Times log file locked - Cannot write to file. Please try again.";

        }

        if (!logNotLocked) {

            result = " ERROR: System log file locked - Cannot write to file. Please try again."
                ;

        }

         if (!timesNotLocked && !logNotLocked) {

           result = " ERROR: Cannot write to time log file or system log file. - Both files
               locked.";

        }

        return result;

    }

    /**
     * Obtains all the times from the time log file ("times.txt") before
     * processing each time log.
     * @return A boolean determining if the file was successfully loaded or not.
     */
    public boolean getTimes() {

        //Obtain a collection of ALL the time logs read in from the "times.txt" file.
        File timesFile = new File("../files/times.txt");

        if (timesFile.exists()) {

            ArrayList<String[]> times = fileIO.readIn(timesFile, true);

            //For every time log read in from the file...
            for (String[] newTime : times) {

                //... process this time log and update the internal record of entrants.
                processNewTime(newTime[0], Integer.parseInt(newTime[1]), Integer.parseInt(
                    newTime[2]));

                this.lastLoggedTime = newTime[3];

            }

            //Log this activity in the log file ("log.txt");
            fileIO.addActivityLog("Time logs file loaded successfully (times.txt)");

            return true;

        }

        return false;
    }

    /**
     * Compares the time of the last read-in time log, with the new time being
     * submitted to check that the user is not entering a time in the past.
     *
     * @param oldTimeString The last time value read-in from "times.txt".
     * @param newTimeString The new time value being submitted by the user.
     * @return A boolean determining if the new time value is in the past.
     */
    public boolean compareTimes(String oldTimeString, String newTimeString) {

        SimpleDateFormat df = new SimpleDateFormat("HH:mm");

        Date lastRecordedTime;
        Date newTime;

        try {

            //Create new Date objects using the last logged, and new time values.
            lastRecordedTime = df.parse(oldTimeString);
            newTime = df.parse(newTimeString);

            //Check if the new time entered is before the last read-in time.
            if (df.format(lastRecordedTime).compareTo(df.format(newTime)) > 0) {

                //If so, then this cannot be allowed.

                //Log this activity in the log file ("log.txt");
                fileIO.addActivityLog("User attempted to enter new time value in the past. (New
                     time: " + df.format(newTime) + ")");

                return true;
            }

        } catch (ParseException ex) {
            Logger.getLogger(ProcessData.class.getName()).log(Level.SEVERE, null, ex);
        }

        return false;
    }
}
```

6.2.2 FileIO.java

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

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

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

    /**
     * The last read-in line number from "times.txt".
     */
```

```java
    private int timesFilePosition = 0;

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

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

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

        try {

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

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


            /*
             * Check if the current file being read in is the times file,
             * and if so whether or not the file has been read-in previously.
             */
            if (isTimesFile && this.timesFilePosition > 0) {

                /*
                 * Read down to the last logged line read-in file
                 * without processing any of the lines (used to "skip" down
                 * to any lines that could have been added after the last time
                 * the file was read in by this application).
                 */
                for (int i = 0; i < this.timesFilePosition; i++) {
                    bufferedReader.readLine();
                }
            }

            //Initialise local variable used to store the current line being read in
            String line;

            //While there are still lines to read in from the file (i.e. read in every line in
                the file)
            while ((line = bufferedReader.readLine()) != null) {

                //As there is multiple data on each line, split the values up.
                String[] details = line.split(" ");

                //Add these broken down values to the larger collection of lines.
                values.add(details);

                /*
                 * If the current file being read in is "times.txt", updated the
                 * last line to be read in by the system. (Used for when the
                 * file is re-"readin" by the system).
                 */
                if (isTimesFile) {
                    timesFilePosition++;
                }
            }

            //Once completed, safely close the file reader
            bufferedReader.close();

            return values;

            //If any IO exceptions occur...
        } catch (IOException iOE) {

                return null;
            }

        }

        /**
         * Writes output data to specified files, as these files are shared, file
         * locking has to be used to prevent corruption of data/files.
         *
         * @param writeFile The file that is to be written to.
         * @param output The output data string.
         * @return A boolean determining if the file was successfully written to.
         */
        public boolean writeFile(File writeFile, String output) {

            FileOutputStream fos;
            FileLock fl = null;

            try {

                //If the file does not exist, create a new file.
                if (!writeFile.exists()) {
                    writeFile.createNewFile();
                }

                //Create a new output stream that will append to the file.
                fos = new FileOutputStream(writeFile.getAbsoluteFile(), true);

                //Attempt to lock the file to allow the data to be written.
                try {

                    fl = fos.getChannel().tryLock();

                } catch (OverlappingFileLockException flE) {

                    /*
                     * If there is already a process within the same JVM locking
                     * the file, inform the user.
                     */
                    System.out.println("ERROR: File <" + writeFile.getName() + "> cannot be
                         accessed. File lock still in place.");
                }

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

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

                        fw.write(output);

                        //Once the data has been successfully written, release the lock.
                        fl.release();
                    }

                    return true;

                }

            } catch (IOException e) {

            }

            return false;
        }

        /**
         * Adds a new log message to the "logs.txt" file. Called when a major
         * activity occurs in the application.
         *
         * @param logMessage Message describing the activity.
         * @return Boolean determining if the log was successfully written to file.
         */
        public boolean addActivityLog(String logMessage) {

            //Obtain the current date/time and format it for use in the log file.
            DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
            Calendar cal = Calendar.getInstance();

            //Build the log message using predefined output template.
            String logOutput = "LOG - CM: " + logMessage + " - " + dateFormat.format(cal.getTime())
                 + "\n";

            //Write the log message to the log file.
            return writeFile(new File("../files/log.txt"), logOutput);

        }
    }
```

6.2.3 LoadData.java

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

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

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

    private Datastore data;
    private FileIO fileIO;

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

        this.data = comp;
        this.fileIO = newFileIO;

    }

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

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

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

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

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

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

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

            } else {

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

            }

            System.out.println("Parameter format = <node path> <courses path> <entrants path>")
                ;
            System.exit(0);
        }


        //If the file does exist, read in the data from the file.
        readValues = fileIO.readIn(f, false);

        //Determine the type of data being loaded.
        if (type.equals(Datatype.NODE)) {

            for (String[] newItem : readValues) {

                //Load all the nodes from the read-in data.
                loadNodes(newItem);

            }

            System.out.println("Nodes file loaded successfully (nodes.txt)");

            //Log this activity in the log file ("log.txt");
            fileIO.addActivityLog("Nodes file loaded successfully (nodes.txt)");

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

            for (String[] newItem : readValues) {

                loadCourses(newItem);

            }

            System.out.println("Courses file loaded successfully (courses.txt)");
            fileIO.addActivityLog("Courses file loaded successfully (courses.txt)");

        } else {

            for (String[] newItem : readValues) {

                loadEntrants(newItem);

            }

            System.out.println("Entrants file loaded successfully (entrants.txt)");
            fileIO.addActivityLog("Entrants file loaded successfully (entrants.txt)");
        }

    }

    /**
     * Parses the data read-in from the "courses.txt" file and creates a new
     * {@link aber.dcs.cs22510.clg11.model.Course} object populated with the
     * read-in characteristics. This new Course object is then added to the
     * internal collection of Courses.
     *
     * @param courseData Collection of all course characteristics data read in
     * from "courses.txt".
     */
    public void loadCourses(String[] courseData) {

        try {

            //Create a new empty Course object.
            Course newCourse = new Course();

            //Set the course ID to the first element in the course data array.
            newCourse.setCourseID(courseData[0].charAt(0));

            //Set the course length to the second element in the course data array.
            newCourse.setCourseLength(Integer.parseInt(courseData[1]));

            //Loop through the REST (i=2) of the "read-in" course data array..
            for (int i = 2; i < (courseData.length); i++) {

                //Loop through all the course nodes stored internally.
                for (Node n : data.getNodes()) {

                    int origNodeNo = n.getNumber();

                    //Obtain the node number currently being parsed from the read in data.
                    int courseNodeNo = Integer.parseInt(courseData[i]);

                    /*
                     * If the node number read-in from file matches the current node,
                     * and the node is NOT a junction, add this node to the collection
                     * of nodes within the new Course object.
                     */
                    if (origNodeNo == courseNodeNo && (n.getType().equals("CP") || n.getType().
                        equals("MC"))) {

                        newCourse.addNewNode(n);
                    }
                }

            }

            /*
             * Once the new Course object has been populated with data,
             * add it to the collection of courses in Datastore.
             */
            data.getCourses().add(newCourse);

            //If an error occurs...
        } catch (Exception e) {

            //... log the error in the "log.txt" file.
            fileIO.addActivityLog("ERROR - Cannot create new course object (" + courseData[0] +
                 ")");
        }

    }

    /**
     * Parses the data read-in from the "nodes.txt" file and creates a new
     * {@link aber.dcs.cs22510.clg11.model.Node} object populated with the
     * read-in characteristics. This new Node object is then added to the
     * internal collection of Nodes.
     *
     * @param nodeData Collection of all node characteristics data read in from
     * "nodes.txt".
     */
    public void loadNodes(String[] nodeData) {

        try {

            Node newNode = new Node();

            newNode.setNumber(Integer.parseInt(nodeData[0]));
            newNode.setType(nodeData[1]);

            data.getNodes().add(newNode);

        } catch (Exception e) {

            fileIO.addActivityLog("ERROR - Cannot create new node object (" + Integer.parseInt(
                 nodeData[0]) + " / " + nodeData[1] + ")");
        }

    }

    /**
     * Parses the data read-in from the "entrants.txt" file and creates a new
     * {@link aber.dcs.cs22510.clg11.model.Entrant} object populated with the
     * read-in characteristics. This new Entrant object is then added to the
     * internal collection of Entrants.
     *
     * @param entrantData Collection of all node characteristics data read in
     * from "nodes.txt".
     */
    public void loadEntrants(String[] entrantData) {

        try {

            Entrant newEntrant = new Entrant();

            newEntrant.setNumber(Integer.parseInt(entrantData[0]));
            newEntrant.setCourseID(entrantData[1].charAt(0));
            newEntrant.setFirstName(entrantData[2]);
            newEntrant.setLastName(entrantData[3]);

            data.getEntrants().add(newEntrant);

        } catch (Exception e) {

            fileIO.addActivityLog("ERROR - Cannot create new entrant object (" + Integer.
                parseInt(entrantData[0]) + " / " + entrantData[1] + " / " + entrantData[2] + "
                 " + entrantData[3] + ")");
        }

    }
}
```

6.3 'Model' Package

6.3.1 Datastore.java

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

import java.util.ArrayList;

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

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

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

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

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


    }

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


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


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

}
```

6.3.2 Entrant.java

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

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

    private String firstName;
    private String lastName;

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

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

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

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

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

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

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

        this.currentProgress = 0;

    }

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

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

    }

    /**
     * Fetches the full name (both first and last) name of the entrant.
     * @return The full name of the entrant.
     */
    public String getFullName() {
        return getFirstName() + " " + getLastName();
    }

    /**
     * Sets the full name of the entrant by splitting the full
     * name on a space and setting the separate first, and last names.
     * @param name The inputted full name to be set.
     */
    public void setFullName(String name) {

        //Split the inputted name by a space.
        String[] tempName = name.split(" ");
```

```
            this.setFirstName(tempName[0]);
            this.setLastName(tempName[1]);
        }

        /**
         * Returns the race number of the entrant.
         * @return The race number of the entrant.
         */
        public int getNumber() {
            return number;
        }

        /**
         * Sets the race number of the entrant.
         * @param number The race number to be set.
         */
        public void setNumber(int number) {
            this.number = number;
        }

        /**
         * Fetches the current progress of the entrant along their course.
         * @return The current progress of the entrant on their course.
         */
        public int getCurrentProgress() {
            return currentProgress;
        }

        /**
         * Updates the current progress of the entrant along their course.
         * @param currentProgress The incremented progress of the entrant.
         */
        public void setCurrentProgress(int currentProgress) {
            this.currentProgress = currentProgress;
        }

        /**
         * Returns the first name of the entrant.
         * @return The first name of the entrant.
         */
        public String getFirstName() {
            return firstName;
        }

        /**
         * Sets the first name only of the entrant.
         * @param firstName The first name of the entrant to be set.
         */
        public void setFirstName(String firstName) {
            this.firstName = firstName;
        }

        /**
         * Returns the last name of the entrant.
         * @return The last name of the entrant.
         */
        public String getLastName() {
            return lastName;
        }

        /**
         * Sets the last name only of the entrant.
         * @param lastName The last name of the entrant to be set.
         */
        public void setLastName(String lastName) {
            this.lastName = lastName;
        }

        /**
         * Fetches the course ID that the entrant is registered for.
         * @return The ID of the registered course.
         */
        public char getCourseID() {
            return courseID;
        }

        /**
         * Sets the course ID of the entrant.
         * @param courseID The ID of the course that the entrant is registered for.
         */
```

```java
    public void setCourseID(char courseID) {
        this.courseID = courseID;
    }

    /**
     * Returns whether the entrant is excluded or not.
     * @return Boolean determining if the entrant is excluded.
     */
    public boolean getIsExcluded() {
        return isExcluded;
    }

    /**
     * Sets whether or not the entrant is excluded.
     * @param isExcluded Whether the entrant is excluded or not.
     */
    public void setIsExcluded(boolean isExcluded) {
        this.isExcluded = isExcluded;
    }

    /**
     * Returns whether the entrant has finished their race or not.
     * @return Boolean determining if the entrant has finished.
     */
    public boolean getIsFinished() {
        return isFinished;
    }

    /**
     * Sets whether or not the entrant has finished their race or not.
     * @param isFinished
     */
    public void setIsFinished(boolean isFinished) {
        this.isFinished = isFinished;
    }

    /**
     * Returns if the entrant is currently at a medical checkpoint.
     * @return Boolean determining if the entrant is at an MC.
     */
    public boolean getAtMC() {
        return atMC;
    }

    /**
     * Sets if an entrant is at a medical checkpoint.
     * @param atMC Whether the entrant is currently at an MC or not.
     */
    public void setAtMC(boolean atMC) {
        this.atMC = atMC;
    }

}
```

6.3.3 Course.java

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

import java.util.ArrayList;

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

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

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

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

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

    }

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

        getCourseNodes().add(newNode);

    }

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

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

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

    /**
     * Sets the Arraylist of course nodes.
     * @param courseNodes The collection of course nodes to be set.
     */
    public void setCourseNodes(ArrayList<Node> courseNodes) {
        this.courseNodes = courseNodes;
    }

    /**
     * Fetches the ID character of the course.
     * @return The ID of the current course.
     */
    public char getCourseID() {
        return courseID;
    }

    /**
     * Sets the ID of the current course.
     * @param courseID The course ID to be set.
     */
    public void setCourseID(char courseID) {
        this.courseID = courseID;
    }

}
```

6.3.4 Node.java

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

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

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

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

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

    }

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

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

    }

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

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

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

    /**
     * Sets the ID number of the current node.
     * @param number the number to set
     */
    public void setNumber(int number) {
        this.number = number;
    }

}
```

6.3.5 Datatype.java

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

/**
```

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

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

6.4 'GUI' Package

6.4.1 GUIFrame.java

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

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

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

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

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

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

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

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

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

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

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

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

6.4.2 GUIPanel.java

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

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

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

    /**
     * Buttons that represent system-wide operations.
     */
    private JButton submitTime, setCurrentTime;
    private JLabel nodeTitle, entrantTitle, mcTypeTitle, timeTitle, statusBar;
    /**
     * The layout manager used by the panel.
     */
    private SpringLayout layout = new SpringLayout();
    /**
     * Drop-down selected boxes used to select entrants and nodes.
     */
    private JComboBox<String> entrantList;
    private JComboBox<String> nodeList;
    /**
     * Determines whether an entrant is arriving or leaving medical checkpoint.
     */
    private JComboBox<String> mcTypeList;
    private String[] mcArriveDepart = {"Arriving", "Departing"};
    /**
     * Spinner used to allow the user to select a time value.
     */
    private JSpinner timeSpinner;
    private SpinnerDateModel sm;
    private Datastore data;
    private LoadData load;
    private JCheckBox mcExclude;
    /**
     * Enables the GUI to access the methods used for processing times.
     */
    private ProcessData proc;
    private FileIO fileIO;

    /**
     * Constructor to instantiate a new GUIPanel. Takes the two classes passed
     * from GUIFrame as parameters to allow panel to acccess shared data store
     * and loading facilities.
     *
     * @param newData Datastore object passed down from GUIFrame.
     * @param newLoad LoadData object passed down from GUIFrame.
     * @param newFileIO FileIO object passed down from GUIFrame.
     */
    public GUIPanel(Datastore newData, LoadData newLoad, FileIO newFileIO) {

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

        //Set the size of the panel
        this.setPreferredSize(new Dimension(500, 250));

        //Set the bespoke layout manager.
        this.setLayout(layout);

        //Initialise and add all of the panel GUI components.
        initComponents();

        setUpLayout();

    }

    /**
     * Initialises the panel components (including linking components to
     * listeners where required) before adding the components to the panel.
     */
    public void initComponents() {

        String[] comboValues;

        /*
         * Instantiate new ProcessData class to allow access to data processing
         * facilties.
         */
        proc = new ProcessData(data, fileIO);

        //Create new instance of JLabel with specified display text
        entrantTitle = new JLabel("Entrant List:");
        nodeTitle = new JLabel("Checkpoint List:");
        mcTypeTitle = new JLabel("Medical CP Type:");
        timeTitle = new JLabel("Log Time:");

        statusBar = new JLabel("Welcome to the Checkpoint Manager.");
        statusBar.setBorder(new BevelBorder(BevelBorder.LOWERED));
        statusBar.setPreferredSize(new Dimension(500, 20));

        //Load all entrant names into entrant drop-down GUI box component.
        comboValues = Arrays.copyOf(getAllEntrants().toArray(), getAllEntrants().toArray().
            length, String[].class);

        entrantList = new JComboBox<>(comboValues);
        entrantList.setSelectedIndex(0);

        //Load all node numbers into node drop-down GUI box component.
        comboValues = Arrays.copyOf(getAllCheckpoints().toArray(), getAllCheckpoints().toArray
            ().length, String[].class);

        nodeList = new JComboBox<>(comboValues);
        nodeList.setSelectedIndex(0);

        //Add local action listener (Required for determining a MC).
        nodeList.addActionListener(this);

        //Load the MC "arrive/depart" options into drop-down GUI box.
        mcTypeList = new JComboBox<>(mcArriveDepart);
        mcTypeList.setSelectedIndex(0);
        mcTypeList.setEnabled(false);


        //Create new instance of JButton with specified button text
        submitTime = new JButton("Submit Checkpoint Time");
        submitTime.addActionListener(this);

        setCurrentTime = new JButton("Set to Current Time");
        setCurrentTime.addActionListener(this);


        //Create new JSpinner model that will access the current system time.
        sm = new SpinnerDateModel();
        sm.setCalendarField(Calendar.MINUTE);

        //Create a new Spinner object and set the above model to it.
        timeSpinner = new JSpinner();
        timeSpinner.setModel(sm);

        //Set the time format to be diplayed in the JSpinner.
        JSpinner.DateEditor de = new JSpinner.DateEditor(timeSpinner, "HH:mm");
        timeSpinner.setEditor(de);

        mcExclude = new JCheckBox("Exclude Entrant");
        mcExclude.setSelected(false);
        mcExclude.setEnabled(false);

        //Add all the components to the GUI panel.
        this.add(nodeTitle);
        this.add(entrantTitle);
        this.add(mcTypeTitle);
        this.add(timeTitle);
        this.add(statusBar);
        this.add(timeSpinner);
        this.add(mcExclude);
        this.add(entrantList);
        this.add(nodeList);
        this.add(mcTypeList);
        this.add(submitTime);
        this.add(setCurrentTime);

    }

    /**
     * Sets up the 'SpringLayout' layout manager to organise all components on
     * the panel.
     */
    public void setUpLayout() {

        //Set the NORTH edge of the label to be 10 pixels down from the NORTH edge of the panel
            .
        layout.putConstraint(SpringLayout.NORTH, nodeTitle, 10, SpringLayout.NORTH, this);

        //Set the WEST edge of the label to be 10 pixels left of the WEST edge of the panel.
        layout.putConstraint(SpringLayout.WEST, nodeTitle, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, nodeList, 10, SpringLayout.NORTH, this);
        layout.putConstraint(SpringLayout.WEST, nodeList, 10, SpringLayout.EAST, nodeTitle);

        layout.putConstraint(SpringLayout.NORTH, entrantTitle, 10, SpringLayout.SOUTH,
            nodeTitle);
        layout.putConstraint(SpringLayout.WEST, entrantTitle, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, entrantList, 10, SpringLayout.SOUTH, nodeTitle
            );
        layout.putConstraint(SpringLayout.WEST, entrantList, 10, SpringLayout.EAST,
            entrantTitle);

        layout.putConstraint(SpringLayout.NORTH, mcTypeTitle, 10, SpringLayout.SOUTH,
            entrantTitle);
        layout.putConstraint(SpringLayout.WEST, mcTypeTitle, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, mcTypeList, 10, SpringLayout.SOUTH,
            entrantTitle);
        layout.putConstraint(SpringLayout.WEST, mcTypeList, 10, SpringLayout.EAST, mcTypeTitle)
            ;

        layout.putConstraint(SpringLayout.NORTH, mcExclude, 10, SpringLayout.SOUTH, mcTypeTitle
            );
        layout.putConstraint(SpringLayout.WEST, mcExclude, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, timeTitle, 10, SpringLayout.SOUTH, mcExclude);
        layout.putConstraint(SpringLayout.WEST, timeTitle, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, timeSpinner, 10, SpringLayout.SOUTH, mcExclude
            );
        layout.putConstraint(SpringLayout.WEST, timeSpinner, 10, SpringLayout.EAST, timeTitle);

        layout.putConstraint(SpringLayout.NORTH, setCurrentTime, 10, SpringLayout.SOUTH,
            timeTitle);
        layout.putConstraint(SpringLayout.WEST, setCurrentTime, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.NORTH, submitTime, 10, SpringLayout.SOUTH,
            setCurrentTime);
        layout.putConstraint(SpringLayout.WEST, submitTime, 10, SpringLayout.WEST, this);

        layout.putConstraint(SpringLayout.SOUTH, statusBar, 0, SpringLayout.SOUTH, this);
    }

    /**
     * Obtains a list of all the entrant names to populate selection box.
     * Accesses them from the array list of entrants contained within
     * {@link aber.dcs.cs22510.clg11.model.Datastore}.
     *
     * @return Arraylist of all the entrant's names.
     */
    public ArrayList<String> getAllEntrants() {

        ArrayList<String> entrantList = new ArrayList<>();

        for (Entrant e : data.getEntrants()) {

            entrantList.add(e.getFullName());

        }

        return entrantList;

    }

    /**
     * Obtains a list of all the checkpoints ONLY to populate the CP selection
     * box. Accesses them from the array list of nodes contained within
     * {@link aber.dcs.cs22510.clg11.model.Datastore}.
     *
     * @return Arraylist of all the nodes that are CHECKPOINTS.
     */
    public ArrayList<String> getAllCheckpoints() {

        ArrayList<String> checkpointList = new ArrayList<>();

        //Loop through all the nodes.
        for (Node cp : data.getNodes()) {

            //If the current node is a checkpoint, and not a junction, add it.
            if (cp.getType().equals("CP") || cp.getType().equals("MC")) {

                checkpointList.add(Integer.toString(cp.getNumber()));

            }

        }

        return checkpointList;

    }

    /**
     * Attempts to fetch a specific node denoted by the node number selected
     * from the drop-down GUI box. If such a node cannot be found, NULL us
     * returned.
     *
     * @param nodeNo The number of the selected node.
     * @return The located node or NULL.
     */
    public Node getNode(int nodeNo) {

        for (Node n : data.getNodes()) {

            if (n.getNumber() == nodeNo) {

                return n;
            }
        }

        return null;

    }

    /**
     * Submits a new time log based on user input within the GUI and determines
     * if an time file currently exists, or if a new one has to be created.
     */
    public void submitCheckpoint() {

        try {


            //Display question dialog
            int shouldProcess = JOptionPane.YES_OPTION;

            shouldProcess = JOptionPane.showConfirmDialog(null, "Are you sure you wish to
                submit this time log?",
                    "CM Manager | Submit Time Log", JOptionPane.YES_NO_OPTION);

            //If user selects "yes"
            if (shouldProcess == JOptionPane.YES_OPTION) {

                //Create a formatter for the time value entered by the user.
                SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

                String newTimeValue = sdf.format(timeSpinner.getValue());
```

```java

                //Check if a times file currently exists.
                if (proc.getTimes()) {

                    updateStatus(" Times file loaded successfully.", false);

                    //Check to see if the time entered by the user is in the past.
                    if (proc.compareTimes(proc.getLastLoggedTime(), newTimeValue)) {

                        updateStatus(" ERROR: Time entered is before last recorded time. Please
                             try again.", true);

                    } else {

                        /*
                         * Re-read in the "times" file to allow any new times logged by other
                         * running versions of the checkpoint manager to update the information
                         * contained within this version of the CM.
                         */
                        updateTimeLog();
                    }

                    //If a time file does not currently exist, create a new one.
                } else {

                    updateStatus(" ALERT: Times file (times.txt) not found. Creating new time
                         log file.", true);
                    updateTimeLog();
                }

            }

        } catch (IndexOutOfBoundsException iOB) {

            updateStatus(" ERROR: Cannot parse times file. Problem reading file.", true);

        }
    }

    /**
     * Takes the user input and processes the new time log entry before adding
     * it to the times log file.
     */
    public void updateTimeLog() {

        String result = null;

        //Obtain the selected entrant from the arraylist of entrants.
        Entrant currentEntrant = data.getEntrants().get(entrantList.getSelectedIndex());

        //Obtain the arraylist of course nodes that make up the course that entrant is
             registered for.
        ArrayList<Node> entrantNodes = proc.obtainEntrantCourseNodes(currentEntrant);

        //Create a formatter for the time value entered by the user.
        SimpleDateFormat sdf = new SimpleDateFormat("HH:mm");

        //Obtain the number of the node selected by the user.
        int nodeNumber = Integer.parseInt((String) nodeList.getSelectedItem());

        //Format the time value entered by the user.
        String newTimeValue = sdf.format(timeSpinner.getValue());

        //Check to see if the node selected was a MC.
        if (mcTypeList.isEnabled()) {

            //If so determine whether they were arriving or departing.
            String mcSelection = (String) mcTypeList.getSelectedItem();

            /*
             * Check if the user is attempting to log an entrant arriving
             * at a MC while the entrant is currently at an MC.
             */
            if (currentEntrant.getAtMC() && mcSelection.equals("Arriving")) {

                updateStatus(" ERROR: Entrant " + currentEntrant.getNumber() + " already at
                     medical checkpoint.", true);

            } else if (!(currentEntrant.getAtMC()) && mcSelection.equals("Departing")) {

                updateStatus(" ERROR: Entrant must be at MC before they can depart.", true);

            } else {

                result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber,
                    mcSelection, newTimeValue, mcExclude.isSelected());

                updateStatus(result, false);
            }

        } else {

            //The checkpoint is not a MC, and so just process the new logged time.
            result = proc.processTimeLog(entrantNodes, currentEntrant, nodeNumber, newTimeValue
                );
            updateStatus(result, false);
        }
    }

    /**
     * Displays system status/error messages to the user via the GUI.
     * @param updateMessage The message to be displayed.
     * @param isError Determines whether the message is an error or not.
     */
    public void updateStatus(String updateMessage, boolean isError) {

        statusBar.setText(updateMessage);

        if (isError) {

            statusBar.setForeground(Color.RED);

        } else {

            statusBar.setForeground(Color.BLACK);
        }

    }

    /**
     * Listener for actions from sub-panel components, to allow operations to be
     * run when components are interacted with.
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     * @param evt - ActionEvent called from components in the panels that
     * require an action to be performed.
     */
    @Override
    public void actionPerformed(ActionEvent evt) {

        String actionCommand = evt.getActionCommand();


        //Switch statement used to capture action commands from buttons.
        switch (actionCommand) {

            case "Submit Checkpoint Time":

                //Submit the entered time values.
                submitCheckpoint();
                break;

            case "Set to Current Time":

                //Obtain the current system time from the Calendar class.
                Calendar currentTime = Calendar.getInstance();

                //Update the value of the time spinner to the current time.
                timeSpinner.setValue(currentTime.getTime());

                updateStatus(" Current time updated successfully.", false);
                break;

        }


        //Listen for events on the nodes drop-down box component.
        if (evt.getSource() == nodeList) {

            //Obtain the selected Node object from the collection of nodes.
            Node n = getNode(Integer.parseInt((String) nodeList.getSelectedItem()));

            //Determine whether the selected node is a medical checkpoint.
            if (n.getType().equals("MC")) {

                //If it is, allow the "arrive/depart" selection box to be used.
                mcTypeList.setEnabled(true);
                mcExclude.setEnabled(true);

            } else {

                mcTypeList.setEnabled(false);
                mcExclude.setEnabled(false);
            }
        }
    }
}
```

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

```
ant -f /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager clean jar
init:
deps-clean:
Updating property file: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/built-
     clean.properties
Deleting directory /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
clean:
init:
deps-jar:
Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
Updating property file: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/built-
     jar.properties
Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/classes
Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/empty
Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/generated-sources
     /ap-source-output
Compiling 11 source files to /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build/
     classes
compile:
Created dir: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist
Copying 1 file to /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/build
Nothing to copy.
Building jar: /home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist/Checkpoint-
     Manager.jar
To run this application from the command line without Ant, try:
java -jar "/home/connor/Git/Endurance-Race-Tracker/Checkpoint-Manager/dist/Checkpoint-Manager.
     jar"
jar:
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

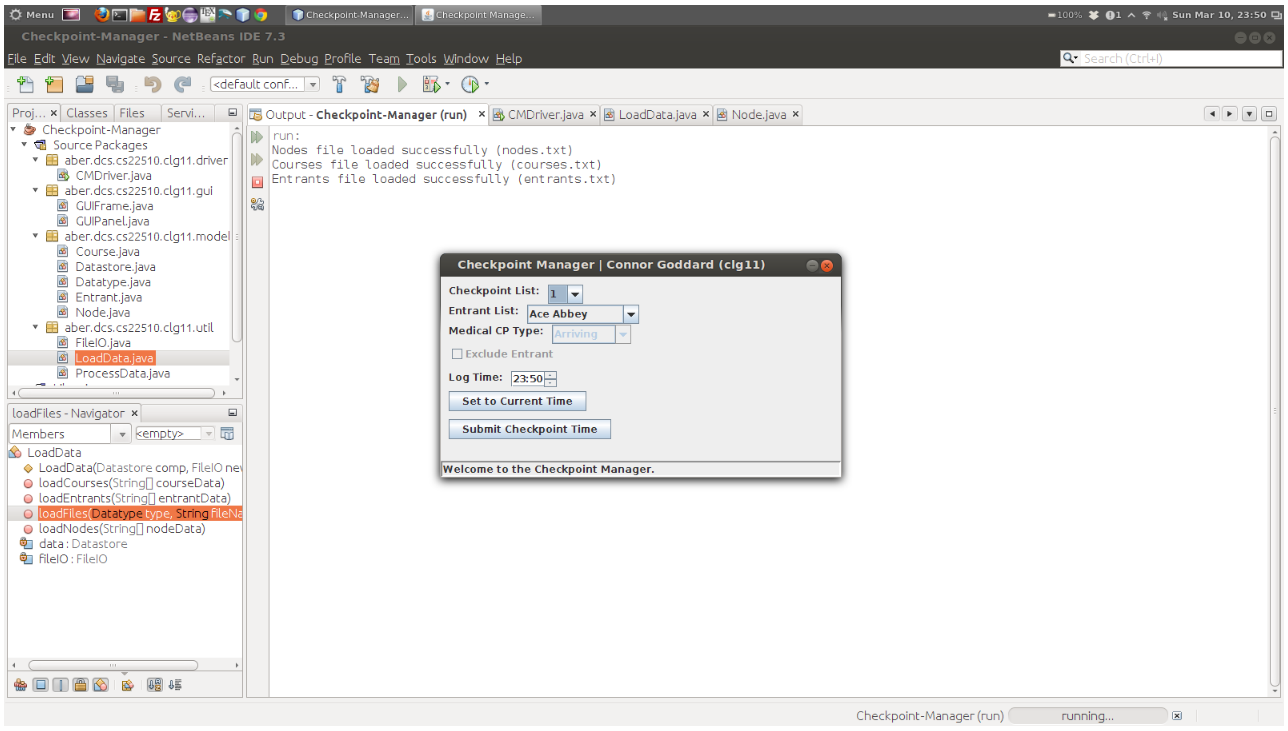
8.1.1 Correct File Parameters

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)
```



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

Checkpoint Manager | Connor Goddard (clg11)

Checkpoint List: 1
Entrant List: Ace Abbey
Medical CP Type: Arriving
Exclude Entrant
Log Time: 23:50
Set to Current Time
Submit Checkpoint Time
Welcome to the Checkpoint Manager.

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""`

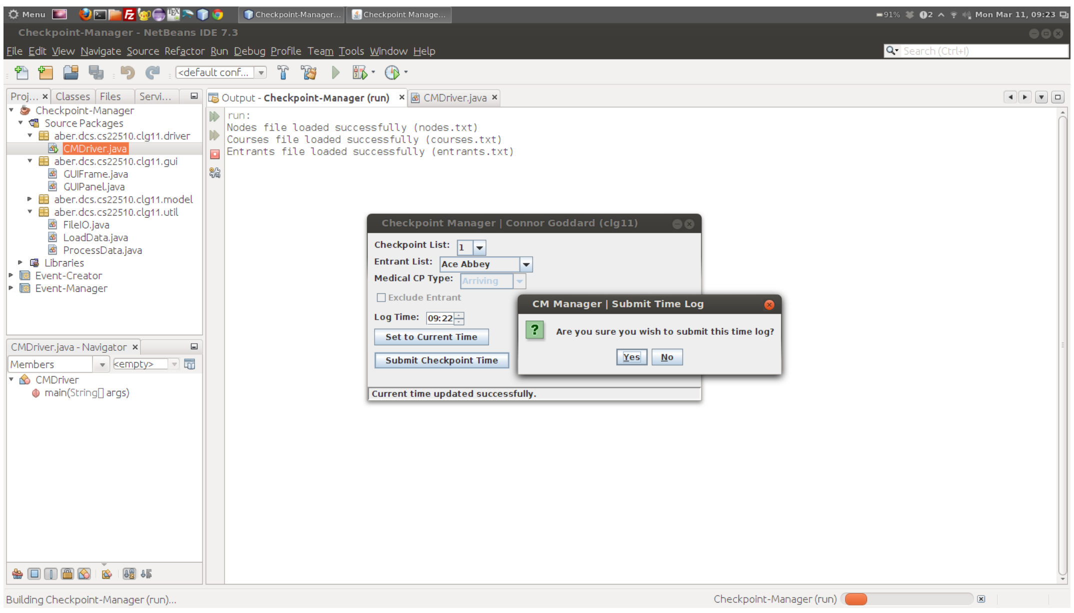
Output:

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).`



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

Checkpoint Manager | Connor Goddard (clg11)

Checkpoint List: 1
Entrant List: Ace Abbey
Medical CP Type: Arriving
Exclude Entrant
Log Time: 09:22

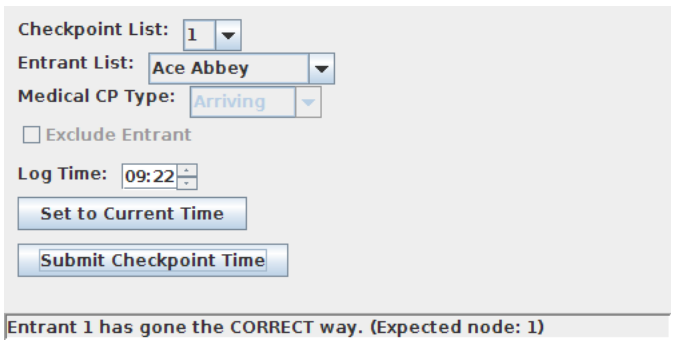
CM Manager | Submit Time Log

Are you sure you wish to submit this time log?

Current time updated successfully.

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

Output:



Checkpoint List: 1
Entrant List: Ace Abbey
Medical CP Type: Arriving
Exclude Entrant
Log Time: 09:22

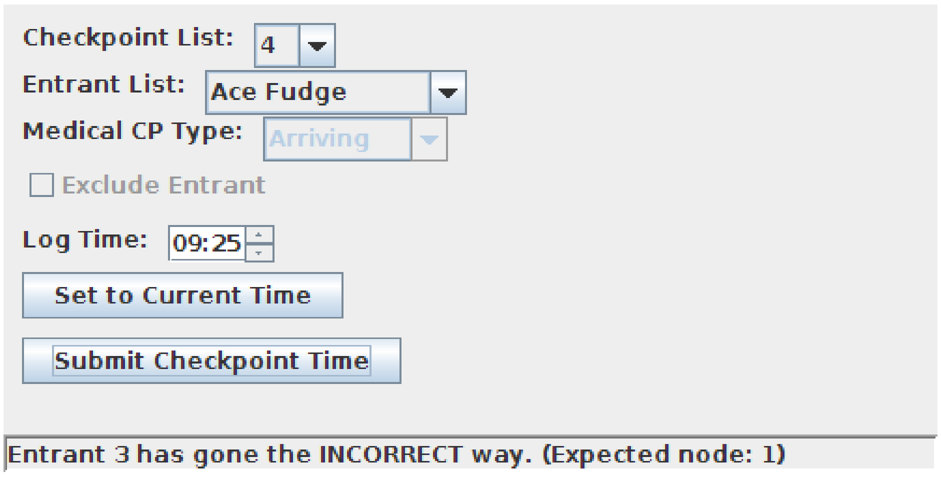
Entrant 1 has gone the CORRECT way. (Expected node: 1)

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:



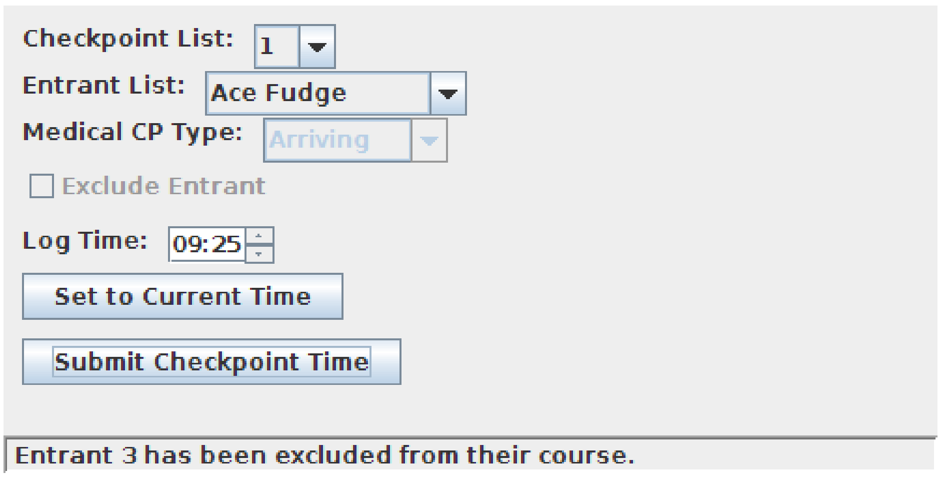
Entrant 3 has gone the INCORRECT way. (Expected node: 1)

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:



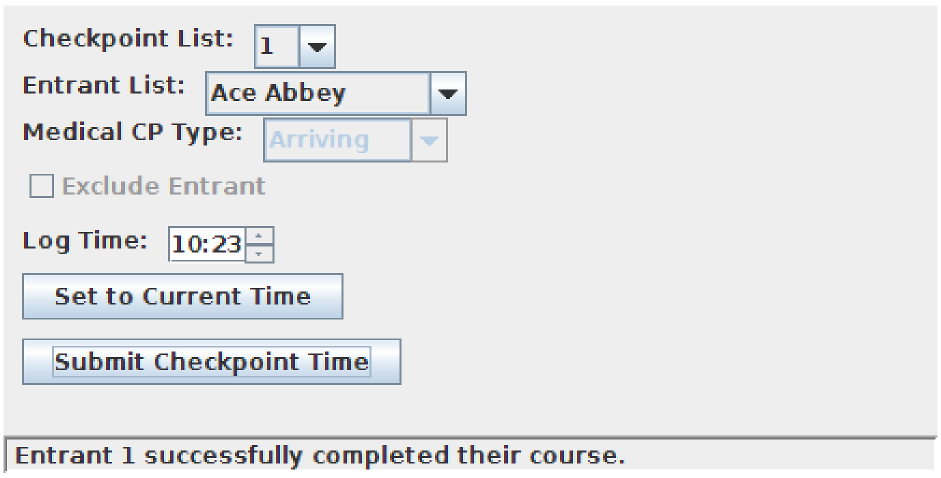
Entrant 3 has been excluded from their course.

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:



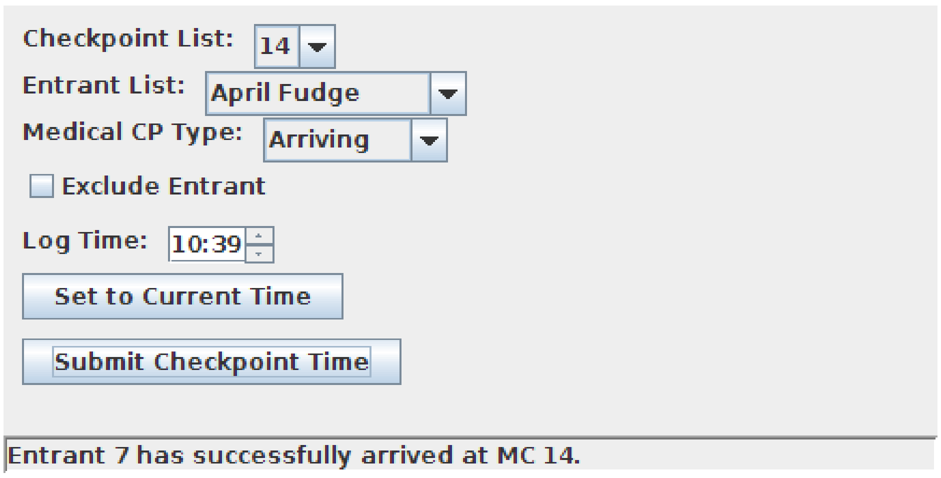
Entrant 1 successfully completed their course.

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:



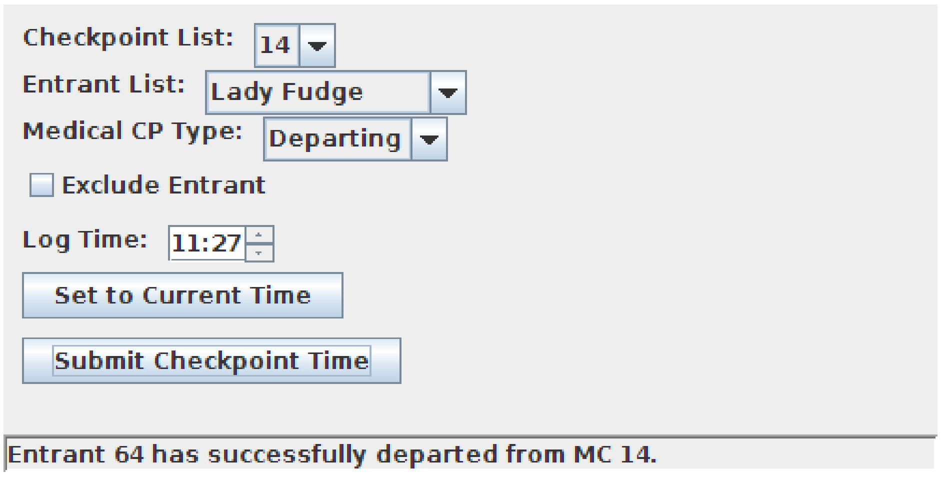
Entrant 7 has successfully arrived at MC 14.

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:


Checkpoint List: 14
Entrant List: Lady Fudge
Medical CP Type: Departing
Exclude Entrant: [ ]
Log Time: 11:27

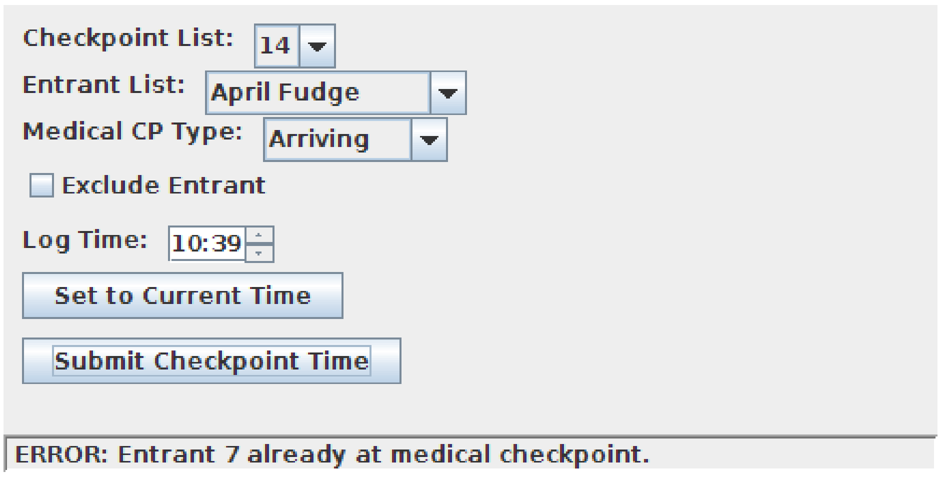
Entrant 64 has successfully departed from MC 14.

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:


Checkpoint List: 14
Entrant List: April Fudge
Medical CP Type: Arriving
Exclude Entrant: [ ]
Log Time: 10:39

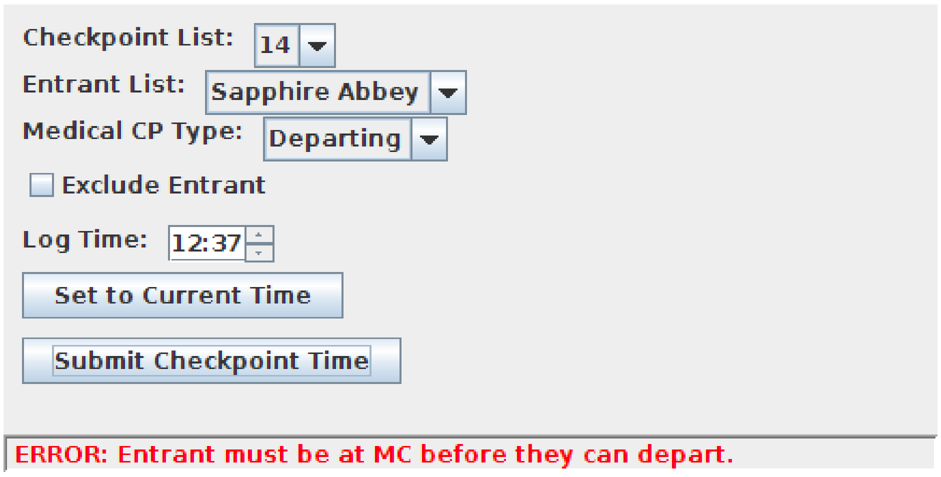
ERROR: Entrant 7 already at medical checkpoint.

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:



Checkpoint List: 14
Entrant List: Sapphire Abbey
Medical CP Type: Departing
☐ Exclude Entrant
Log Time: 12:37

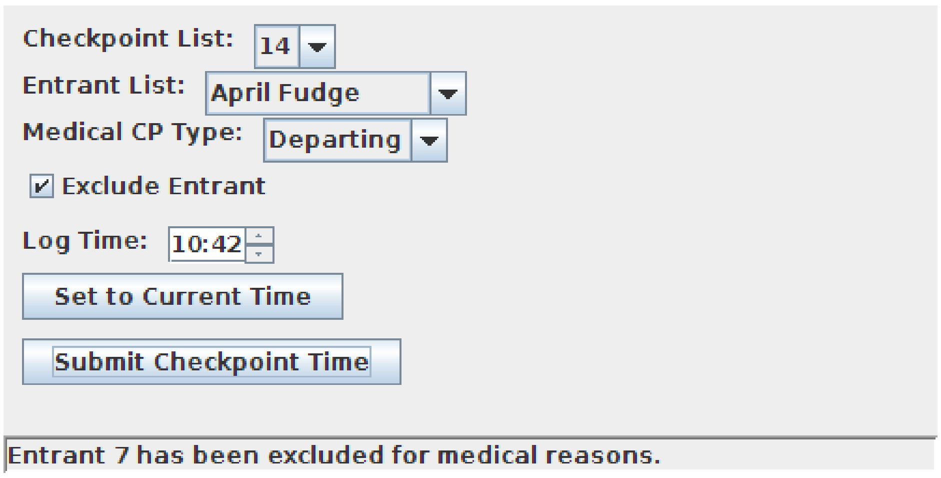
ERROR: Entrant must be at MC before they can depart.

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:



Checkpoint List: 14
Entrant List: April Fudge
Medical CP Type: Departing
☑ Exclude Entrant
Log Time: 10:42

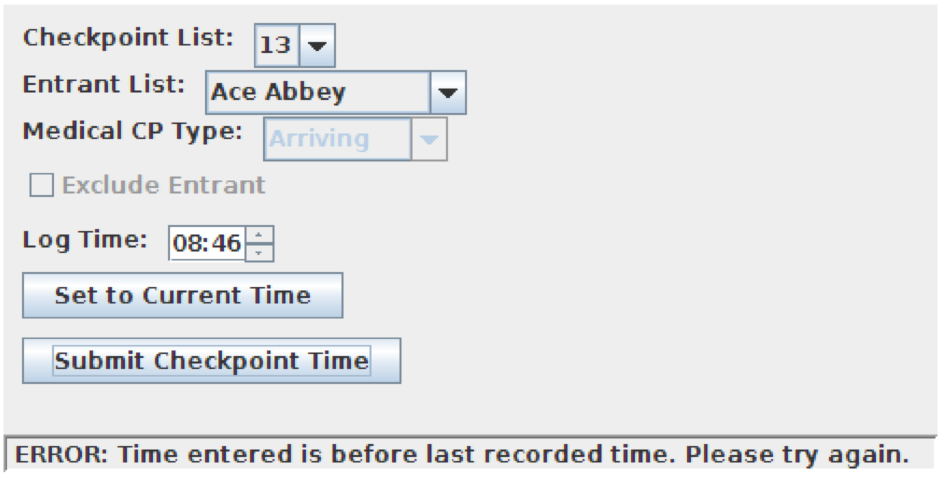
Entrant 7 has been excluded for medical reasons.

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:



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

8.13 File Lock Access Prevention

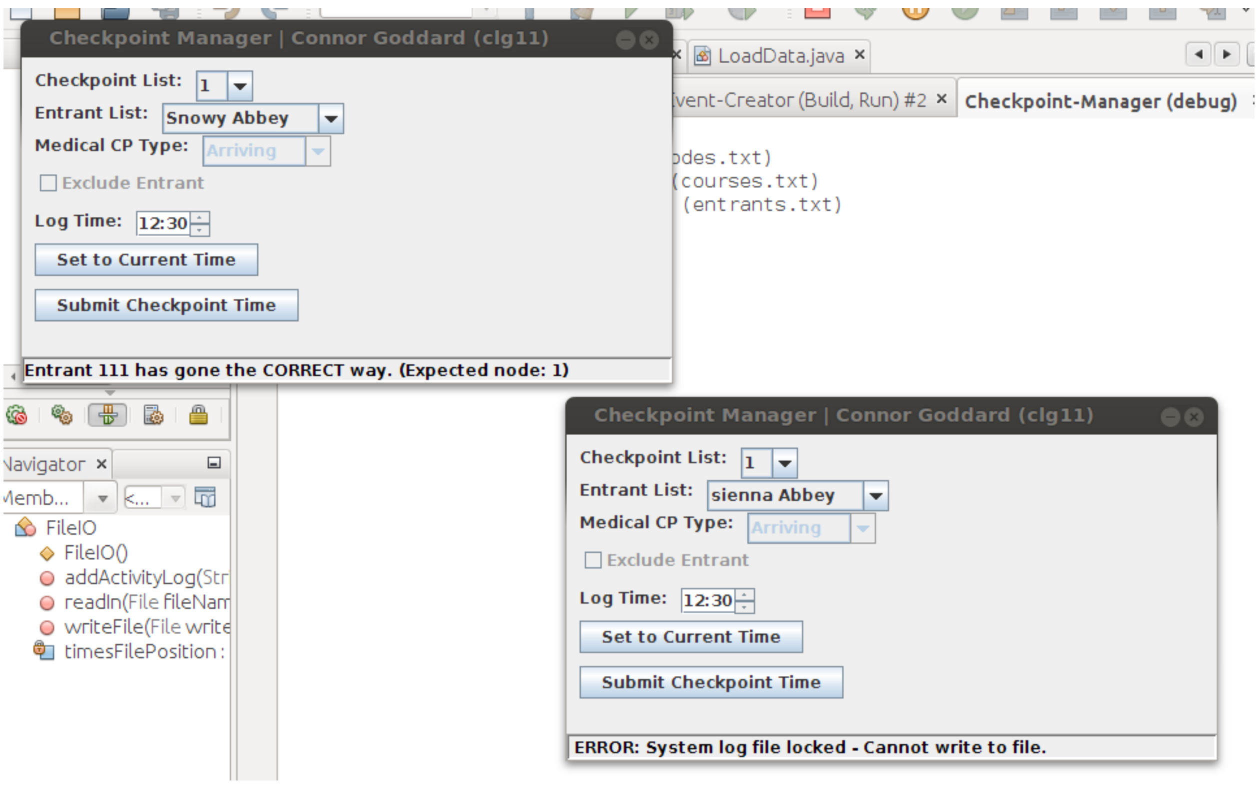
Input:

```
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.

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

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

                  fw.write(output);

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

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

              return true;

          }
```

Output:



Checkpoint Manager | Connor Goddard (clg11)

Checkpoint List: 1
Entrant List: Snowy Abbey
Medical CP Type: Arriving
Exclude Entrant
Log Time: 12:30

```
Entrant 111 has gone the CORRECT way. (Expected node: 1)
```

Checkpoint Manager | Connor Goddard (clg11)

Checkpoint List: 1
Entrant List: sienna Abbey
Medical CP Type: Arriving
Exclude Entrant
Log Time: 12:30

```
ERROR: System log file locked - Cannot write to file.
```

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

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


CLEAN SUCCESSFUL (total time: 59ms)

"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-conf
make[1]: Entering directory ‘/home/connor/Git/Endurance-Race-Tracker/Event-Manager’
"/usr/bin/make"  -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/event-manager
make[2]: Entering directory ‘/home/connor/Git/Endurance-Race-Tracker/Event-Manager’
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/course.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/display.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/entrant.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/event.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/fileIO.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/main.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/node.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/process.o.d
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
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/track.o.d
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
mkdir -p dist/Debug/GNU-Linux-x86
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
make[2]: Leaving directory ‘/home/connor/Git/Endurance-Race-Tracker/Event-Manager’
make[1]: Leaving directory ‘/home/connor/Git/Endurance-Race-Tracker/Event-Manager’


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 seperate applications that form the "runners and riders" system.

13.1 Event Creator (C++)

13.2 Checkpoint Manager (Java)

13.3 Event Manager (C)