

CS22510 Assignment 1
Runners and Riders
"Out and About"

Chris Savill
`chs17@aber.ac.uk`

March 20, 2013

Contents

1	Description of three programs	3
1.1	Event Creation Program	3
1.2	Checkpoint Manager Program	3
1.3	Event Manager Program	3
2	Code for the Event Creation Program	3
3	Clean build and compilation of Event Creation Program	17
4	Run through of Event Creation Program	18
5	Files created by execution of Event Creation Program	18
6	Code for Checkpoint Manager Program	18
7	Clean build and compilation of Checkpoint Program	45
8	Run through of Checkpoint Manager Program	46
9	Files created by execution of Event Creation Program	46
10	Clean build and compilation of Event Manager Program	46
11	Run through of Event Manager Program	48
12	Results list produced at the end of an event	64
13	Log file contents	64

1 Description of three programs

1.1 Event Creation Program

1.2 Checkpoint Manager Program

1.3 Event Manager Program

2 Code for the Event Creation Program

Listing 1: Header file for non-class specific functions.

```
1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: creator.h
4   * Description: Header file for the starter function declarations.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #ifndef CREATOR_H
10 #define CREATOR_H
11
12 #include <memory>
13 #include "event.h"
14
15 bool get_acceptance(); //Function to get the user's input for accepting or
    rejecting their inputs.
16 bool checkCourseExists(char letter, Event *event); //Member function that
    checks if the letter given be the user matches any of the course letters.
17 void ecp_menu(Event *event); //Function that launches the event creation
    program menu.
18
19 #endif /* CREATOR_H */
```

Listing 2: Main method and menu file.

```
1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: competitor.cpp
4   * Description: cpp file that contains function definitions for the start-up
    of the event creation program.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #include "creator.h"
10 #include <iostream>
11 #include <cstdlib>
12 #include <limits>
13
14 using namespace std;
15
16 /* Main function that just calls a function that takes over. */
17 int main(int argc, char** argv) {
18     Event *event = new Event();
19     ecp_menu(event);
20
21     return 0;
22 }
```

```

23
24  /* Function to get the user's input for accepting or rejecting their inputs.
    */
25  bool get_acceptance() {
26      char option;
27
28      do {
29          cout << "If yes press 'y' then 'Enter'" << endl << "If no press 'n'
              then 'Enter'" << endl;
30          cin.clear();
31          option = cin.get();
32          cin.ignore(numeric_limits<streamsize>::max(), '\n');
33
34          if (option == 'y') return true;
35          else if (option == 'n') return false;
36          else cout << "Invalid option selected" << endl;
37      } while (option != 'y' && option != 'n');
38  }
39
40  /* Function that displays the main menu for the event creation program. */
41  void ecp_menu(Event *event) {
42      int option; //Field to store the user's option input.
43
44      do {
45          cout << "*****"
              << endl;
46          cout << "*   Runners and Riders Event Creation Program Main Menu   *"
              << endl;
47          cout << "*****"
              << endl;
48          cout << "1. Add Competitor to Event" << endl;
49          cout << "2. Add Course to Event" << endl;
50          cout << "3. Export Event to File" << endl;
51          cout << "4. Export Competitors to File" << endl;
52          cout << "5. Export Courses to File" << endl;
53          cout << "6. Exit Event Creation Program" << endl;
54          cout << "*****"
              << endl << endl;
55
56          cout << "Please enter in an option from the above an press 'Enter':
              ";
57          cin.clear();
58          cin >> option;
59          cin.ignore();
60
61          switch (option) {
62              case 1:
63                  event->add_competitor();
64                  break;
65              case 2:
66                  event->add_course();
67                  break;
68              case 3:
69                  event->export_event();
70                  break;
71              case 4:
72                  event->export_competitors();

```

```

73         break;
74     case 5:
75         event->export_courses();
76         break;
77     case 6:
78         delete(event);
79         cout << "Exiting program..." << endl << endl;
80         break;
81     default:
82         cout << "Please enter in a valid option." << endl << endl;
83     }
84 } while (option != 6);
85 }

```

Listing 3: Header file Event class.

```

1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: event.h
4   * Description: Header file for the Event class.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #ifndef EVENT_H
10 #define EVENT_H
11
12 #include <memory>
13 #include "competitor.h"
14 #include "course.h"
15 #include <vector>
16 #include <cstdlib>
17 #include <iostream>
18
19 #define MAX_EVENT_NAME_LENGTH 79
20 #define MAX_DATE_LENGTH 19
21
22 class Competitor;
23 class Course;
24
25 class Event {
26 private:
27     std::string name; //Name of the event.
28     std::string date; //Date of the event.
29     std::string start_time; //Start time of the event.
30     std::vector<Competitor*> *competitors; //Array of competitors to take
        part in the event.
31     std::vector<Course*> *courses; //Array of courses that are part of an
        event.
32
33     void set_name(); //Member function to get the user to input the events
        name.
34     void set_date(); //Member function to get the user to input the date of
        the event.
35     void set_start_time(); //Member function to get the user to input the
        start time of the event.
36
37 public:
38     Event();
39     ~Event();
40     std::vector<Course*>* getCourses(); //Member function that returns a
        pointer to the vector of courses.

```

```

41     void add_competitor(); //Member function that will handle adding a
        competitor to the event.
42     void add_course(); //Member function that will handle adding a course to
        the event.
43     void export_event(); //Member function that will handle exporting the
        name, date and start_time of the event to a '.txt' file.
44     void export_competitors(); //Member function that will handle the
        exporting of the array of competitors to a '.txt' file.
45     void export_courses(); //Member function that will handle the exporting
        of the array of courses to a '.txt' file.
46 };
47
48 #endif /* EVENT_H */

```

Listing 4: Cpp file for Event class.

```

1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: event.cpp
4   * Description: cpp file that contains member function definitions for the
        event class.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #include "event.h"
10 #include "creator.h"
11 #include <iostream>
12 #include <stdlib.h>
13 #include <fstream>
14 #include <sstream>
15 #include <limits>
16
17 using namespace std;
18
19 /* Member function that returns a pointer to the vector of courses. */
20 vector<Course*> Event::getCourses() {
21     return courses;
22 }
23
24 /* Member function to get the user to input the events name. */
25 void Event::set_name() {
26     bool name_chosen = false;
27     string name;
28
29     do {
30         do {
31             cout << "Please enter in the name for the event (no more than 79
                characters): ";
32             cin.clear();
33             getline(cin, name);
34         } while (name.length() > MAX_EVENT_NAME_LENGTH);
35
36         cout << endl << endl << "Are you happy with the name: '" << name <<
            "'?" << endl;
37         name_chosen = get_acceptance();
38     } while (name_chosen == false);
39
40     this->name = name;
41 }
42
43 /* Member function to get the user to input the date of the event. */
44 void Event::set_date() {

```

```

45     bool date_chosen = false;
46     string date;
47
48     do {
49         do {
50             cout << endl << endl << "Please enter in the date for the event
                    (no more than 19 characters): ";
51             cin.clear();
52             getline(cin, date);
53         } while (date.length() > MAX_DATE_LENGTH);
54
55         cout << endl << endl << "Are you happy with the date: '" << date <<
                    "'?" << endl;
56         date_chosen = get_acceptance();
57     } while (date_chosen == false);
58
59     this->date = date;
60 }
61
62 /* Member function to get the user to input the start time of the event. */
63 void Event::set_start_time() {
64     bool start_time_chosen = false;
65     bool valid_hours = false;
66     bool valid_minutes = false;
67     char input[3];
68     int hours;
69     int minutes;
70     string start_time;
71     string string_hours;
72     string string_minutes;
73
74     do {
75         do {
76             cout << endl << endl << "Please enter in the start time for the
                    event with the 24 hour format 'HH:MM', hours first: ";
77             cin.clear();
78             cin >> input;
79             cin.ignore(numeric_limits<streamsize>::max(), '\n');
80             cout << endl;
81
82             if (isdigit(input[0]) && isdigit(input[1])) { //Ensures the
                    input has 2 digits.
83                 hours = atoi(input); //Converts the digits into an int and
                    stores it in hours.
84
85                 if (hours <= 23 && hours >= 00) { //Makes sure that the
                    hours are in 24-hour format.
86                     cout << "Valid hours entered." << endl << endl;
87                     valid_hours = true;
88                 }
89             } else cout << "Invalid hours entered, please enter in a value
                    between 00 and 23 inclusive." << endl << endl;
90         } while (valid_hours == false);
91
92         do {
93             cout << endl << endl << "Please now enter in the minutes: ";
94             cin.clear();
95             cin >> input;
96             cin.ignore(numeric_limits<streamsize>::max(), '\n');
97             cout << endl;
98
99             if (isdigit(input[0]) && isdigit(input[1])) {
100                 minutes = atoi(input);

```

```

101
102         if (minutes <= 59 && minutes >= 00) { //Makes sure minutes
103             are valid.
104             cout << "Valid minutes entered." << endl << endl;
105             valid_minutes = true;
106         }
107     } else cout << "Invalid minutes entered, please enter in a value
108         between 00 and 59 inclusive." << endl << endl;
109 } while (valid_minutes == false);
110
111     cout << endl << endl << "Are you happy with the start time: '" <<
112         hours << ":" << minutes << "'?" << endl;
113     start_time_chosen = get_acceptance();
114 } while (start_time_chosen == false);
115
116     ostringstream string_retriever; //Converts ints into strings.
117     string_retriever << hours;
118     string_hours = string_retriever.str();
119     string_retriever.str(""); //Clears the string stream.
120     string_retriever << minutes;
121     string_minutes = string_retriever.str();
122
123     start_time = string_hours + ":" + string_minutes; //Concatenates the
124         final time into HH:MM format.
125     this->start_time = start_time;
126 }
127
128 /* Member function that will handle adding a competitor to the event.
129 * @param number The current competitor number.
130 */
131 void Event::add_competitor() {
132     if (courses->empty()) cout << "No courses exist for competitor course
133         selection. Please create a course first." << endl << endl;
134     else {
135         Competitor *competitor = new Competitor((competitors->size() + 1),
136             this);
137         competitors->push_back(competitor);
138         cout << "New competitor added to event." << endl << endl;
139         cout << "Competitor number: " << competitors->back()->get_number();
140         cout << "Competitor name: " << competitors->back()->get_name() <<
141             endl;
142         cout << "Course: " << competitors->back()->get_course() << endl;
143     }
144 }
145
146 /* Member function that will handle adding a course to the event. */
147 void Event::add_course() {
148     Course *course = new Course(this);
149     courses->push_back(course);
150     cout << "New course added to event." << endl << endl;
151     cout << "Course letter: " << courses->back()->get_letter() << endl;
152     cout << "Number of course nodes: " << courses->back()->
153         get_number_of_nodes() << endl;
154     cout << "Nodes: " << courses->back()->get_node(0);
155
156     for (int counter = 1; counter < courses->back()->get_number_of_nodes();
157         counter++) {
158         cout << ", " << courses->back()->get_node(counter);
159     }
160
161     cout << endl << endl;
162 }

```



```

155  /* Member function that will handle exporting the name, date and start_time
      of the event to a '.txt' file. */
156  void Event::export_event() {
157      ofstream competitors_file;
158      competitors_file.open("name.txt", ios::out);
159
160      if (competitors_file.is_open()) {
161          competitors_file << this->name << "\n" << this->date << "\n" << this
              ->start_time;
162          competitors_file.close();
163          cout << "Event successfully exported to 'name.txt'." << endl << endl
              ;
164      } else cout << "File 'name.txt' could not be written." << endl;
165  }
166
167  /* Member function that will handle the exporting of the array of
      competitors to a '.txt' file. */
168  void Event::export_competitors() {
169      if (competitors->empty()) cout << "No competitors to export. Exporting
          cancelled." << endl << endl;
170      else {
171          ofstream competitors_file;
172          competitors_file.open("entrants.txt", ios::out);
173
174          if (competitors_file.is_open()) {
175              for (int counter = 0; counter < this->competitors->size();
                  counter++) {
176                  competitors_file << this->competitors->at(counter)->
                      get_number() << " " << this->competitors->at(counter)->
                      get_course()
177                      << " " << this->competitors->at(counter)->get_name()
                      << "\n";
178              }
179
180              competitors_file.close();
181              cout << "Competitors successfully exported to 'entrants.txt'."
                  << endl << endl;
182          } else cout << "File 'entrants.txt' could not be written." << endl;
183      }
184  }
185
186  /* Member function that will handle the exporting of the array of courses to
      a '.txt' file. */
187  void Event::export_courses() {
188      if (courses->empty()) cout << "No courses to export. Exporting cancelled
          ." << endl << endl;
189      else {
190          ofstream courses_file;
191          courses_file.open("courses.txt", ios::out);
192
193          if (courses_file.is_open()) {
194              for (int counter = 0; counter < this->courses->size(); counter
                  ++){
195                  courses_file << this->courses->at(counter)->get_letter() <<
                      " " << this->courses->at(counter)->get_number_of_nodes();
196
197                  for (int counter2 = 0; counter2 < this->courses->at(counter)
                      ->get_number_of_nodes(); counter2++) {
198                      courses_file << " " << this->courses->at(counter)->
                          get_node(counter2);
199                  }
200                  courses_file << "\n";
201              }

```

```

202
203         courses_file.close();
204         cout << "Courses successfully exported to 'courses.txt'." <<
                endl << endl;
205     } else cout << "File 'courses.txt' could not be written." << endl;
206 }
207 }
208
209 /* Constructor for Event class. */
210 Event::Event() {
211     competitors = new vector<Competitor* > ();
212     courses = new vector<Course* > ();
213     set_name();
214     set_date();
215     set_start_time();
216
217     cout << "Event name: " << this->name << endl;
218     cout << "Event date: " << this->date << endl;
219     cout << "Event start time: " << this->start_time << endl << endl;
220 }
221
222 /* Destructor for Event class. */
223 Event::~~Event() {
224     delete(competitors);
225     delete(courses);
226 }

```

Listing 5: Header file for Course class.

```

1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: course.h
4   * Description: Header file for the Course class.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #ifndef COURSE_H
10 #define COURSE_H
11
12 #include <memory>
13 #include <vector>
14
15 class Event;
16
17 class Course {
18 private:
19     char letter; //The courses unique identification letter for an event.
20     int number_of_nodes; //The number of nodes the course contains.
21     std::vector<int> *nodes; //An array of nodes that are contained in the
        course.
22     std::vector<int> *nodes_available; //An array of nodes that are
        available to select from, read in from the 'nodes.txt' file.
23
24     void set_letter(Event *event); //Member function that will set the
        letter of the course.
25     void set_number_of_nodes(); //Member function that will set the number
        of nodes of the course.
26     bool read_nodes_available(); //Member function that reads in the nodes
        from the 'nodes.txt' file and adds them to the nodes available array.
27     void add_node(); //Member function that adds a new node to the course.
28     bool duplicated_last_node(int number); //Member function to check if the
        new node being selected matches the last node added.

```

```

29     bool check_node_exists(int number); //Member function that checks that
        the node being added exists in the array of nodes available.
30
31 public:
32     char get_letter(); //Member function to return a course's letter.
33     int get_number_of_nodes(); //Member function to return a course's number
        of nodes.
34     int get_node(int index); //Member function to return a node from the
        course's vector of nodes.
35     Course(Event *event);
36     ~Course();
37 };
38
39 #endif /* COURSE_H */

```

Listing 6: Cpp file for Course class.

```

1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: course.cpp
4   * Description: cpp file that contains member function definitions for the
        course class.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #include "course.h"
10 #include "creator.h"
11 #include <iostream>
12 #include <fstream>
13 #include <sstream>
14 #include <limits>
15
16 using namespace std;
17
18 /* Member function to return a course's letter. */
19 char Course::get_letter() {
20     return this->letter;
21 }
22
23 /* Member function to return a course's number of nodes. */
24 int Course::get_number_of_nodes() {
25     return this->number_of_nodes;
26 }
27
28 /* Member function to return a node from the course's vector of nodes. */
29 int Course::get_node(int index) {
30     return this->nodes->at(index);
31 }
32
33 /* Member function that checks if the letter given by the user matches any
        of the course letters. */
34 bool checkCourseExists(char letter, Event *event) {
35     for (int counter = 0; counter < event->getCourses()->size(); counter++)
36     {
37         if (letter == event->getCourses()->at(counter)->get_letter()) return
            true; //Checks if letter matches any of the course letters.
38     }
39     return false; //Return false if no match found.
40 }
41
42 /* Member function that will set the letter of the course. */

```

```

43 void Course::set_letter(Event *event) {
44     bool valid_letter = false;
45     bool letter_chosen = false;
46     char letter;
47
48     do {
49         do {
50             cout << endl << endl << "Please enter in the course letter for
                    the course: ";
51             cin.clear();
52             letter = cin.get();
53             cin.ignore(numeric_limits<streamsize>::max(), '\n');
54
55             if (isalpha(letter) && !checkCourseExists(letter, event))
                    valid_letter = true; //Checks that character entered is a
                    letter and that it does not match any course letters.
56             else {
57                 cout << "Please enter in a valid course letter that does not
                        already exist in this event, a-z or A-Z." << endl <<
                        endl;
58                 valid_letter = false;
59             }
60         } while (valid_letter == false);
61
62         cout << endl << "Are you happy with the course letter: '" << letter
                << "'?" << endl;
63         letter_chosen = get_acceptance();
64     } while (letter_chosen == false);
65
66     this->letter = letter;
67 }
68
69 /* Member function that will set the number of nodes of the course. */
70 void Course::set_number_of_nodes() {
71     bool number_chosen = false;
72     int number;
73
74     do {
75         cout << endl << endl << "Please enter in the number of nodes for
                    this course: ";
76         cin.clear();
77         cin >> number;
78         cin.ignore(numeric_limits<streamsize>::max(), '\n');
79
80         cout << endl << endl << "Are you happy with the number of nodes: '"
                << number << "'?" << endl;
81         number_chosen = get_acceptance();
82     } while (number_chosen == false && number > 0);
83
84     this->number_of_nodes = number;
85 }
86
87 /* Member function that reads in the nodes from the 'nodes.txt' file and
    adds them to the nodes available array. */
88 bool Course::read_nodes_available() {
89     ifstream nodes_file;
90     string input;
91     int node_number;
92
93     nodes_file.open("nodes.txt", ios::in);
94
95     if (nodes_file.is_open()) {
96         while (getline(nodes_file, input)) { //Keep reading until EOF

```

```

100     }
101
102     nodes_file.close();
103     cout << "Nodes from 'nodes.txt' read in successfully." << endl;
104     cout << "Nodes read in: " << nodes_available->at(0);
105     for (int counter = 1; counter < nodes_available->size(); counter++)
106         cout << ", " << nodes_available->at(counter);
107     cout << endl << endl;
108 } else cout << "File 'nodes.txt' could not be opened. Please check file
109     is in correct directory and permissions." << endl;
110 }
111
112 /* Member function that adds a new node to the course. */
113 void Course::add_node() {
114     bool number_chosen = false;
115     string input;
116     int number = 0;
117
118     do {
119         do {
120             cout << "Please enter in the node number you wish to add to the
121                 course: ";
122             getline(cin, input);
123             stringstream int_retriever(input);
124             int_retriever >> number;
125         } while (duplicated_last_node(number) || !check_node_exists(number))
126             ; //Makes sure that the number entered doesn't match the last
127             number entered and that it does exist.
128
129         cout << endl << endl << "Are you happy with the node number: '" <<
130             number << "'?" << endl;
131         number_chosen = get_acceptance();
132     } while (number_chosen == false);
133
134     this->nodes->push_back(number);
135 }
136
137 /* Member function to check if the new node being selected matches the last
138     node added. */
139 bool Course::duplicated_last_node(int number) {
140     if (!nodes->empty()) { //Only checks if there are nodes present.
141         if (number == nodes->back()) {
142             cout << "Node matches last node. Please choose a different node
143                 number to add." << endl;
144             return true;
145         }
146     }
147
148     return false; //Returns false if the number entered and the last number
149     entered don't match.
150 }
151
152 /* Member function that checks that the node being added exists in the array
153     of node available. */
154 bool Course::check_node_exists(int number) {
155     for (int counter = 0; counter < this->nodes_available->size(); counter
156         ++){
157         if (number == this->nodes_available->at(counter)) return true;

```

```

147     }
148
149     cout << "Node does not exist, please choose a different node number to
        add." << endl;
150     return false; //Returns false if the number entered does not exist in
        the vector of nodes available.
151 }
152
153 /* Constructor for Course class. */
154 Course::Course(Event *event) {
155     this->nodes = new vector<int>();
156     this->nodes_available = new vector<int>();
157
158     if (read_nodes_available()) {
159         set_letter(event);
160         set_number_of_nodes();
161
162         for (int counter = 0; counter < number_of_nodes - 1; counter++) {
163             add_node();
164         }
165
166         nodes->push_back(nodes->front()); //Adds the last node, matching the
            first node to the course.
167     } else cout << "Nodes could not be read in from 'nodes.txt' file. Course
        creation cancelled." << endl << endl;
168 }
169
170 /* Destructor for Course class. */
171 Course::~~Course() {
172     delete(nodes);
173     delete(nodes_available);
174 }

```

Listing 7: Header file for Competitor class.

```

1  /*
2  * Author: Chris Savill, chs17@aber.ac.uk
3  * File Name: competitor.h
4  * Description: Header file for the Competitor class.
5  * First Created: 11/03/2013
6  * Last Modified: 14/03/2013
7  */
8
9  #ifndef COMPETITOR_H
10 #define COMPETITOR_H
11
12 #include <memory>
13 #include <string>
14
15 #define MAX_COMPETITOR_NAME_LENGTH 51 //Includes null terminator \0.
16
17 class Event;
18
19 class Competitor {
20 private:
21     int number; //The competitor's unique identification number for an event
22     .
23     std::string name; //The competitor's name.
24     char course; //The course letter the competitor is entering in for.
25
26     void set_number(int number); //Member function that will set the number
        of the competitor.

```

```

26     void set_name(); //Member function that will set the name of the
        competitor.
27     void set_course(Event *event); //Member function that will set the
        course letter for the competitor.
28
29 public:
30     int get_number(); //Member function to return a competitor's number.
31     std::string get_name(); //Member function to return a competitor's name.
32     char get_course(); //Member function to return a competitor's course.
33     Competitor(int number, Event *event);
34 };
35
36 #endif /* COMPETITOR_H */

```

Listing 8: Cpp file for Competitor class.

```

1  /*
2   * Author: Chris Savill, chs17@aber.ac.uk
3   * File Name: competitor.cpp
4   * Description: cpp file that contains member function definitions for the
        competitor class.
5   * First Created: 11/03/2013
6   * Last Modified: 14/03/2013
7   */
8
9  #include "competitor.h"
10 #include "creator.h"
11 #include <ctype.h>
12 #include <iostream>
13 #include <limits>
14
15 using namespace std;
16
17 /* Member function to return a competitor's number. */
18 int Competitor::get_number() {
19     return this->number;
20 }
21
22 /* Member function to return a competitor's name. */
23 string Competitor::get_name() {
24     return this->name;
25 }
26
27 /* Member function to return a competitor's course. */
28 char Competitor::get_course() {
29     return this->course;
30 }
31
32 /* Member function that will set the number of the competitor.
33  * @param number The number for the competitor.
34  */
35 void Competitor::set_number(int number) {
36     this->number = number;
37 }
38
39 /* Member function that will set the name of the competitor. */
40 void Competitor::set_name() {
41     bool name_chosen = false;
42     string name;
43
44     do {
45         do {

```

```

46         cout << endl << endl << "Please enter in the name for the
           competitor (no more than 50 characters): ";
47         getline(cin, name);
48     } while (name.length() > MAX_COMPETITOR_NAME_LENGTH);
49
50     cout << endl << endl << "Are you happy with the name: '" << name <<
       "'?" << endl;
51
52     name_chosen = get_acceptance();
53
54     } while (name_chosen == false);
55
56     this->name = name;
57 }
58
59 /* Member function that will set the course letter for the competitor. */
60 void Competitor::set_course(Event *event) {
61     bool valid_letter = false;
62     bool letter_chosen = false;
63     char letter;
64
65     do {
66         do {
67             cout << endl << endl << "List of courses available for the
               competitor to enter on: " << event->getCourses()->front()->
               get_letter();
68
69             if (event->getCourses()->size() > 1) { //Only prints out other
               courses if the size of the vector > 1.
70                 for (int counter = 1; counter < event->getCourses()->size();
                       counter++)
71                     cout << ", " << event->getCourses()->at(counter)->
                       get_letter();
72             }
73
74             cout << endl << endl << "Please enter in the letter of the
               course that the competitor is entering: ";
75             cin.clear(); //Resets the input stream flags.
76             letter = cin.get(); //Gets a single character.
77             cin.ignore(numeric_limits<streamsize>::max(), '\n'); //Clears
               the input stream.
78
79             if (isalpha(letter) && checkCourseExists(letter, event))
               valid_letter = true; //Makes sure character is a letter and
               that it corresponds to a course that exists.
80             else {
81                 cout << "Please enter in a valid course letter." << endl <<
                   endl;
82                 valid_letter = false;
83             }
84         } while (valid_letter == false);
85
86         cout << endl << "Are you happy with the course letter: '" << letter
           << "'?" << endl;
87         letter_chosen = get_acceptance();
88     } while (letter_chosen == false);
89
90     this->course = letter;
91 }
92
93 /* Constructor for Competitor class.
94 * @param number The number for the new competitor.
95 */

```



```

96 | Competitor::Competitor(int number, Event *event) {
97 |     set_number(number);
98 |     cout << "Competitor number: " << this->number << endl;
99 |     set_name();
100 |     cout << "Competitor name: " << this->name << endl;
101 |     set_course(event);
102 |     cout << "Competitor course:" << this-> course << endl;
103 | }

```

3 Clean build and compilation of Event Creation Program

```

"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-
conf
make[1]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Creation_Program'
rm -f -r build/Debug
rm -f dist/Debug/GNU-Linux-x86/event_creation_program
make[1]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Creation_Program'

```

CLEAN SUCCESSFUL (total time: 217ms)

```

"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-
conf
make[1]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Creation_Program'
"/usr/bin/make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/
event_creation_program
make[2]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Creation_Program'
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/main.o.d
g++ -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/main.o.d -o build/Debug/
GNU-Linux-x86/main.o main.cpp
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/course.o.d
g++ -c -g -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/event.o.d
g++ -c -g -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/competitor.o.d
g++ -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/competitor.o.d -o build/
Debug/GNU-Linux-x86/competitor.o competitor.cpp
mkdir -p dist/Debug/GNU-Linux-x86
g++ -o dist/Debug/GNU-Linux-x86/event_creation_program build/Debug/GNU-
Linux-x86/main.o build/Debug/GNU-Linux-x86/course.o build/Debug/GNU-Linux
-x86/event.o build/Debug/GNU-Linux-x86/competitor.o
make[2]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Creation_Program'
make[1]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Creation_Program'

```

BUILD SUCCESSFUL (total time: 5s)

4 Run through of Event Creation Program

5 Files created by execution of Event Creation Program

6 Code for Checkpoint Manager Program

Listing 9: Launcher class.

```
1  /* File Name: Launcher.java
2  * Description: Launcher class which handles the initial launching of the
   Checkpoint Manager Program.
3  * First Created: 15/03/2013
4  * Last Modified: 19/03/2013
5  */
6  package Data_Structures;
7
8  import GUI.TypeWindow;
9  import java.io.IOException;
10 import javax.swing.JOptionPane;
11
12 /**
13 * @author Chris Savill, chs17@aber.ac.uk
14 */
15 public class Launcher {
16
17     /**
18     * Main method that checks that the right number of arguments were
       received
19     * and calls methods to load the file required and launch the GUI.
20     *
21     * @param args String array of arguments, should be a list of file names
       *
22     */
23     public static void main(String[] args) throws IOException {
24         if (args.length < 4) {
25             JOptionPane.showMessageDialog(null, "Invalid number of file
                names supplied required for program to run.\n\n"
26                 + "File names required for:\nFile containing nodes\nFile
                containing courses\nFile containing entrants\n"
27                 + "File to retrieve time records and write time records
                to.\n\n"
28                 + "Now exiting program.");
29         } else {
30             Event event = new Event(args);
31
32             if (event.loadCycle(args)) {
33                 JOptionPane.showMessageDialog(null, "Data files loaded
                successfully.");
34                 TypeWindow typeWindow = new TypeWindow(event);
35             } else {
36                 System.out.print("Exiting Program...\n");
37             }
38         }
39     }
40 }
```

Listing 10: Event class.

```
1 /* File Name: Manager.java
```

```

2  * Description: Event class which stores all members and functions
   pertaining to an event.
3  * First Created: 15/03/2013
4  * Last Modified: 18/03/2013
5  */
6  package Data_Structures;
7
8  import File_Handling.FileHandler;
9  import java.io.IOException;
10 import java.util.ArrayList;
11 import java.util.Date;
12
13 /**
14  * @author Chris Savill, chs17@aber.ac.uk
15  */
16 public class Event {
17
18     private ArrayList<Competitor> competitors; //Array list of competitors
        in an event.
19     private ArrayList<Node> nodes; //Array list of nodes in an event.
20     private ArrayList<Node> checkpoints; //Array list of nodes that are of
        type "CP" or "MC".
21     private ArrayList<Course> courses; //Array list of courses in an event.
22     private ArrayList<Record> records; //Array list of records logged.
23     private int lastLineRead;
24     private Date lastRecordedTime;
25     private boolean timeFileExists;
26     private String[] fileNames;
27
28     /**
29      * Method to return array list of competitors.
30      *
31      * @return The array list of competitors.
32      */
33     public ArrayList<Competitor> getCompetitors() {
34         return competitors;
35     }
36
37     /**
38      * Method to return array list of nodes.
39      *
40      * @return The array list of nodes.
41      */
42     public ArrayList<Node> getNodes() {
43         return nodes;
44     }
45
46     /**
47      * Method to return array list of checkpoints.
48      *
49      * @return The array list of checkpoints (non-junction nodes).
50      */
51     public ArrayList<Node> getCheckpoints() {
52         return checkpoints;
53     }
54
55     /**
56      * Method to return array list of courses.
57      *
58      * @return The array list of courses.
59      */
60     public ArrayList<Course> getCourses() {
61         return courses;

```

```

62     }
63
64     /**
65      * Method to return array list of records.
66      *
67      * @return The array list of records.
68      */
69     public ArrayList<Record> getRecords() {
70         return records;
71     }
72
73     /**
74      * Method to get the last line read number.
75      *
76      * @return The line read from the times file.
77      */
78     public int getLastLineRead() {
79         return lastLineRead;
80     }
81
82     /**
83      * Method to return the array of file names.
84      *
85      * @return The string array of file names.
86      */
87     public String[] getFileNames() {
88         return fileNames;
89     }
90
91     /**
92      * Method to set the last line read number.
93      *
94      * @param lineNumber The line read from the times file.
95      */
96     public void setLastLineRead(int lineNumber) {
97         this.lastLineRead = lineNumber;
98     }
99
100    /**
101     * Method to set the last time recorded.
102     *
103     * @param time The last time recorded.
104     */
105    public void setLastRecordedTime(Date time) {
106        this.lastRecordedTime = time;
107    }
108
109    /**
110     * Method to call a series of methods to load in the data required by
111     * the
112     * program.
113     *
114     * @param args The list of filenames to load the required data into the
115     * system.
116     * @return Successful/Unsuccessful.
117     */
118    public boolean loadCycle(String[] fileNames) throws IOException {
119        this.fileNames = fileNames;
120
121        FileHandler fileReader = new FileHandler();
122
123        if (fileReader.readNodes(fileNames[0], this)) {
124            if (fileReader.readCourses(fileNames[1], this)) {

```

```

124         if (fileReader.readCompetitors(fileNames[2], this)) {
125             return true;
126         } else {
127             System.out.print("Failed to load competitors. Program
128                               Exiting.\n");
129         }
130     } else {
131         System.out.print("Failed to load courses. Program Exiting.\n
132                           ");
133     }
134     } else {
135         System.out.print("Failed to load nodes. Program Exiting.\n");
136     }
137     return false;
138 }
139
140 /**
141  * Method that checks if the node number passed in exists in the array
142  * list
143  * of nodes loaded in.
144  *
145  * @param number The number to be compared with.
146  * @return True if node exists else false.
147  */
148 public boolean checkNodeExists(int number) {
149     for (int counter = 0; counter < nodes.size(); counter++) {
150         if (number == nodes.get(counter).getNumber()) {
151             return true;
152         } //Nodes exists.
153     }
154     return false; //Returns false if the node number passed in does not
155                   //exist in the array list of nodes.
156 }
157
158 /**
159  * Method that checks if the course letter passed in exists in the array
160  * list of courses loaded in.
161  *
162  * @param letter The letter to be compared with.
163  * @return True if course exists else false.
164  */
165 public boolean checkCourseExists(char letter) {
166     for (int counter = 0; counter < courses.size(); counter++) {
167         if (letter == courses.get(counter).getLetter()) {
168             return true;
169         } //Course exists.
170     }
171     return false; //Returns false if the course letter passed in does
172                   //not exist in the array list of courses.
173 }
174
175 /**
176  * Method to let the know event instance know that a time file does now
177  * exist.
178  *
179  */
180 public void setTimesFilesExistsTrue() {
181     timeFileExists = true;
182 }

```

```

182  /**
183   * Method to find a competitor and return it.
184   *
185   * @param competitorNumber The number of the competitor being looked for
186   *
187   * @return The competitor matched.
188   */
189  public Competitor retrieveCompetitor(int competitorNumber) {
190      for (Competitor competitor : competitors) {
191          if (competitor.getNumber() == competitorNumber) {
192              return competitor;
193          }
194      }
195      return null;
196  }
197
198  /**
199   * Method to find a course and return it.
200   *
201   * @param courseLetter The course being looked for.
202   *
203   * @return The course matched.
204   */
205  public Course retrieveCourse(char courseLetter) {
206      for (Course course : courses) {
207          if (course.getLetter() == courseLetter) {
208              return course;
209          }
210      }
211      return null;
212  }
213
214  /**
215   * Method to retrieve the checkpoint number.
216   *
217   * @param type The type of the checkpoint.
218   * @param listIndex The index of the list element.
219   * @param numberOfElements The size of the list.
220   *
221   * @return The checkpoint number being looked for.
222   */
223  public int retrieveCheckpointNumber(String type, int listIndex, int
224      numberOfElements) {
225      int[] checkpointArray = new int[numberOfElements];
226      int arrayIndex = 0;
227
228      for (int counter = 0; counter < checkpoints.size(); counter++) {
229          if (checkpoints.get(counter).getType().equals(type)) {
230              checkpointArray[arrayIndex++] = checkpoints.get(counter).
231                  getNumber();
232          }
233      }
234
235      return checkpointArray[listIndex];
236  }
237
238  /**
239   * Method to check if the new record is valid.
240   *
241   * @param checkpoint The checkpoint number.
242   * @param status The status.
243   * @param competitorNumber The competitor's number.
244   * @param time The time of the record.
245   *
246   * @return True is record is valid, else false.
247   */

```

```

242 public boolean checkNewRecord(int checkpoint, int status, int
243     competitorNumber, Date time) {
244     Competitor competitor = retrieveCompetitor(competitorNumber);
245
246     if (timeFileExists != false) {
247         if (time.before(lastRecordedTime)) {
248             System.out.println("\nInvalid time.");
249             return false;
250         }
251     }
252
253     if (competitor.getStatus() == 'I' || competitor.getStatus() == 'E')
254     {
255         System.out.println("\nCompetitor already excluded.");
256         return false; //Should not be updated as competitor already
257             excluded.
258     } else if (status == 2 || status == 3) {
259         if (competitor.getStatus() != 'A') {
260             System.out.println("\nCompetitor hasn't arrived at a medical
261                 checkpoint yet.");
262             return false; //Competitor cannot be departing or be exclude
263                 from a medical checkpoint they haven't arrived at.
264         } else {
265             return true;
266         }
267     } else if (status == 0) {
268         if (competitor.getStatus() != 'A') {
269             return true;
270         } else {
271             System.out.println("\nCompetitor is still being examined at
272                 a medical checkpoint.");
273             return false; //Competitor cannot be at a time checkpoint
274                 when should be at a medical checkpoint being examined.
275         }
276     } else if (status == 1) {
277         return true;
278     }
279
280     return false;
281 }
282
283 /**
284  * Method to determine the final status to be written to the time record
285  * file.
286  * @param checkpoint The checkpoint number.
287  * @param status The status.
288  * @param competitorNumber The competitor's number.
289  * @return The final status for the record.
290  */
291 public char determineFinalStatus(int checkpoint, int status, int
292     competitorNumber) {
293     Competitor competitor = retrieveCompetitor(competitorNumber);
294
295     if (competitor.getStatus() == 'N') {
296         if (checkpoint != competitor.getCheckpoints()[competitor.
297             getCheckpointIndex()]) {
298             return 'I';
299         } else if (status == 0) {
300             return 'T';
301         } else if (status == 1) {
302             return 'A';
303         }
304     } else if (competitor.getStatus() == 'A') {

```

```

295         if (status == 2) {
296             return 'D';
297         } else if (status == 3) {
298             return 'E';
299         }
300     } else if (checkpoint != competitor.getCheckpoints()[competitor.
        getCheckpointIndex() + 1]) {
301         return 'I';
302     } else {
303         if (status == 0) {
304             return 'T';
305         } else if (status == 1) {
306             return 'A';
307         } else if (status == 2) {
308             return 'D';
309         } else if (status == 3) {
310             return 'E';
311         }
312     }
313
314     System.out.print("\n\nInvalid final status, returning 'I'.\n");
315     return 'I';
316 }
317
318 /**
319  * Constructor to initialise the event.
320  */
321 public Event(String[] fileNames) {
322     competitors = new ArrayList<Competitor>();
323     nodes = new ArrayList<Node>();
324     checkpoints = new ArrayList<Node>();
325     courses = new ArrayList<Course>();
326     records = new ArrayList<Record>();
327     lastLineRead = 0;
328     timeFileExists = false;
329 }
330 }

```

Listing 11: Node class.

```

1  /* File Name: Node.java
2   * Description: Node class which stores all members and functions pertaining
        to a node.
3   * First Created: 15/03/2013
4   * Last Modified: 15/03/2013
5   */
6  package Data_Structures;
7
8  /**
9   * @author Chris Savill, chs17@aber.ac.uk
10  */
11 public class Node {
12
13     private int number;
14     private String type;
15
16     /**
17      * Constructor to initialise Node.
18      *
19      * @param number The number of the node.
20      * @param type The type of the node.
21      */
22     public Node(int number, String type) {

```



```

23         this.number = number;
24         this.type = type;
25     }
26
27     /**
28      * Method to return the node's number.
29      *
30      * @return The node number.
31      */
32     public int getNumber() {
33         return number;
34     }
35
36     /**
37      * Method to return the node's type.
38      * @return The type of the node.
39      */
40     public String getType() {
41         return type;
42     }
43 }

```

Listing 12: Course class.

```

1  /* File Name: Couse.java
2   * Description: Course class which stores all members and functions
   * pertaining to a course.
3   * First Created: 15/03/2013
4   * Last Modified: 17/03/2013
5   */
6  package Data_Structures;
7
8  /**
9   * @author Chris Savill, chs17@aber.ac.uk
10  */
11 public class Course {
12
13     private char letter;
14     private int numberOfNodes;
15     private int[] nodes;
16
17     /**
18      * Constructor to initialise course.
19      *
20      * @param letter The course letter identifier.
21      * @param numberOfNodes The number of nodes the course contains.
22      * @param nodes The array of nodes the course contains.
23      */
24     public Course(char letter, int numberOfNodes, int[] nodes) {
25         this.letter = letter;
26         this.numberOfNodes = numberOfNodes;
27         this.nodes = nodes;
28     }
29
30     /**
31      * Method to return the course letter.
32      */
33     public char getLetter() {
34         return letter;
35     }
36
37     /**
38      * Method to return the number of nodes the course contains.

```

```

39     */
40     public int getNumberOfNodes() {
41         return numberOfNodes;
42     }
43
44     /**
45      * Method to return the array of nodes the course contains.
46      */
47     public int[] getNodes() {
48         return nodes;
49     }
50 }

```

Listing 13: Competitor class.

```

1  /* File Name: Competitor.java
2   * Description: Competitor class which stores all members and functions
   * pertaining to a competitor.
3   * First Created: 15/03/2013
4   * Last Modified: 18/03/2013
5   */
6  package Data_Structures;
7
8  import java.util.ArrayList;
9
10 /**
11  * @author Chris Savill, chs17@aber.ac.uk
12  */
13 public class Competitor {
14
15     private String name;
16     private int number;
17     private char course;
18     private char status;
19     private int[] checkpoints;
20     private int checkpointIndex;
21
22     /**
23      * Constructor to initialise competitor.
24      *
25      * @param number The competitor's number.
26      * @param course The competitor's course.
27      * @param name The competitor's name.
28      */
29     public Competitor(int number, char course, String name, Event event) {
30         this.number = number;
31         this.course = course;
32         this.name = name;
33         this.checkpoints = setCheckpoints(event);
34         this.checkpointIndex = 0;
35         this.status = 'N'; //Not started yet.
36     }
37
38     /**
39      * Method to return the competitor's number.
40      *
41      * @return The number of the competitor.
42      */
43     public int getNumber() {
44         return number;
45     }
46
47     /**

```

```

48      * Method to return the course the competitor is entered on.
49      *
50      * @return The course the competitor entered in on.
51      */
52      public char getCourse() {
53          return course;
54      }
55
56      /**
57       * Method to return the competitor's name.
58       *
59       * @return The name of the competitor.
60       */
61      public String getName() {
62          return name;
63      }
64
65      /**
66       * Method to return the competitor's status.
67       *
68       * @return The status of the competitor.
69       */
70      public char getStatus() {
71          return status;
72      }
73
74      /**
75       * Method to return the index of the last checkpoint the competitor
76       * arrived
77       * at.
78       *
79       * @return The index of the last checkpoint the competitor arrived at.
80       */
81      public int getCheckpointIndex() {
82          return checkpointIndex;
83      }
84
85      /**
86       * Method to return the int array of checkpoints.
87       *
88       * @return The int array of checkpoints.
89       */
90      public int[] getCheckpoints() {
91          return checkpoints;
92      }
93
94      /**
95       * Method to get the nodes which are recordable checkpoints (non-
96       * junction
97       * nodes).
98       *
99       * @param event The event instance.
100      * @return The int array of checkpoints.
101      */
102      private int[] setCheckpoints(Event event) {
103          ArrayList<Integer> checkpointsList = new ArrayList<Integer>();
104          Course courseReference = event.retrieveCourse(course);
105
106          for (int counter = 0; counter < courseReference.getNumberOfNodes();
              counter++) {
              for (int counter2 = 0; counter2 < event.getNodes().size();
                  counter2++) {
                  if ((!event.getNodes().get(counter2).getType().equals("JN"))

```

```

107         && (event.getNodes().get(counter2).getNumber() ==
108             courseReference.getNodes()[counter])) {
109             checkpointsList.add(event.getNodes().get(counter2).
110                 getNumber());
111             break;
112         }
113     }
114
115     int[] intList = new int[checkpointsList.size()];
116
117     for (int counter = 0; counter < checkpointsList.size(); counter++) {
118         intList[counter] = checkpointsList.get(counter).intValue();
119     }
120
121     return intList;
122 }
123
124 /**
125  * Method to set the status of the competitor.
126  *
127  * @param status The current status of the competitor.
128  */
129 public void setStatus(char status) {
130     this.status = status;
131 }
132
133 /**
134  * Method to increment the checkpoint index by 1.
135  */
136 public void incrementCheckpointIndex() {
137     checkpointIndex++;
138 }
139 }

```

Listing 14: Record class.

```

1  /* File Name: Record.java
2   * Description: Record class which stores all members and functions
3   * pertaining to checking a competitor in at a checkpoint.
4   * First Created: 15/03/2013
5   * Last Modified: 17/03/2013
6   */
7  package Data_Structures;
8
9  import java.util.Date;
10
11 /**
12  * @author Chris Savill, chs17@aber.ac.uk
13  */
14 public class Record {
15
16     private Event event;
17     private char competitorStatus;
18     private int checkpoint;
19     private int competitorNumber;
20     private Date time;
21
22     /**
23      * Constructor to initialise record data when read in from file.
24      *
25      * @param checkpoint The number of the checkpoint.
26      * @param competitorNumber The number of the competitor.

```

```

26     * @param time The time of the record.
27     */
28     public Record(char status, int checkpoint, int competitorNumber, Date
        time) {
29         this.competitorStatus = status;
30         this.checkpoint = checkpoint;
31         this.competitorNumber = competitorNumber;
32         this.time = time;
33     }
34
35     /**
36     * Constructor to initialise record data when recorded through GUI.
37     *
38     * @param checkpoint The number of the checkpoint.
39     * @param competitorNumber The number of the competitor.
40     * @param time The time of the record.
41     */
42     public Record(int checkpoint, char status, int competitorNumber, Date
        time) {
43         this.competitorStatus = status;
44         this.checkpoint = checkpoint;
45         this.competitorNumber = competitorNumber;
46         this.time = time;
47     }
48
49     /**
50     * Method to return the status of the competitor as marked by the
51     * checkpoint.
52     *
53     * @return The status of the competitor.
54     */
55     public char getCompetitorStatus() {
56         return competitorStatus;
57     }
58
59     /**
60     * Method to return the checkpoint number being recorded.
61     *
62     * @return The checkpoint number.
63     */
64     public int getCheckpointNumber() {
65         return checkpoint;
66     }
67
68     /**
69     * Method to return the competitor number being recorded.
70     *
71     * @return The competitor number.
72     */
73     public int getCompetitorNumber() {
74         return competitorNumber;
75     }
76
77     /**
78     * Method to return the time being recorded.
79     *
80     * @return The time of the record.
81     */
82     public Date getTime() {
83         return time;
84     }
85 }

```

Listing 15: FileHandler class.

```

1  /* File Name: FileHandler.java
2   * Description: FileHandler class which stores methods to handle the reading
       of files.
3   * First Created: 15/03/2013
4   * Last Modified: 18/03/2013
5   */
6  package File_Handling;
7
8  import Data_Structures.Competitor;
9  import Data_Structures.Course;
10 import Data_Structures.Event;
11 import Data_Structures.Node;
12 import Data_Structures.Record;
13 import java.io.BufferedReader;
14 import java.io.FileNotFoundException;
15 import java.io.FileReader;
16 import java.io.FileWriter;
17 import java.io.IOException;
18 import java.io.RandomAccessFile;
19 import java.nio.channels.FileChannel;
20 import java.nio.channels.FileLock;
21 import java.text.ParseException;
22 import java.text.SimpleDateFormat;
23 import java.util.Date;
24 import java.util.logging.Level;
25 import java.util.logging.Logger;
26
27 /**
28  * @author Chris Savill, chs17@aber.ac.uk
29  */
30 public class FileHandler {
31
32     /**
33      * Method to read in all the details for the nodes pertaining to an
           event.
34      *
35      * @param fileName The file name required to access the file needed.
36      * @param event The event instance.
37      * @return True if file loaded successfully, else false if it fails at
           any
38      * point.
39      */
40     public boolean readNodes(String fileName, Event event) throws
       IOException {
41         String input;
42         int nodeNumber;
43         String nodeType;
44         String[] subStrings;
45         String pattern = "(\\d+\\s+([A-Z]{2}))$"; //Regular expression for
           nodes file.
46
47         try {
48             BufferedReader reader = new BufferedReader(new FileReader(
               fileName));
49
50             while ((input = reader.readLine()) != null) {
51                 if (input.matches(pattern)) { //Checks to make sure the line
                   is in the right format.
52                     subStrings = input.split("\\s+"); //Gets rid of
                   whitespace and separates the two sides into two
                   substrings.

```

```

53         nodeNumber = Integer.parseInt(subStrings[0]); //
           Retrieves the node number by parsing the string into
           an int.
54         nodeType = subStrings[1]; //Retrieves the node type.
55
56         Node node = new Node(nodeNumber, nodeType); //Creates
           new node with parameters read in.
57         event.getNodes().add(node); //Adds new node to array
           list of nodes.
58
59         if (node.getType().equals("CP") || node.getType().equals
           ("MC")) {
60             event.getCheckpoints().add(node); //Adds new node to
           array list of checkpoints if the node is of type
           "CP or "MC".
61         }
62     } else {
63         System.out.print("Invalid line format. Cancelling
           loading of nodes.\n\n");
64         reader.close();
65         return false;
66     }
67 }
68
69 if (!event.getNodes().isEmpty()) {
70     System.out.print("Loading in of nodes successful.\n\n");
71     reader.close();
72     return true;
73 } else {
74     System.out.print("Loading in of nodes unsuccessful. No nodes
           in file.\n\n");
75     reader.close();
76     return false;
77 }
78 } catch (FileNotFoundException ex) {
79     Logger.getLogger(FileHandler.class.getName()).log(Level.SEVERE,
           null, ex);
80 }
81
82 System.out.print("Could not open file that contains nodes.\n\n");
83 return false;
84 }
85
86 /**
87  * Method to read in all the details for the courses pertaining to an
           event.
88  *
89  * @param fileName The file name required to access the file needed.
90  * @param event The event instance.
91  * @return True if file loaded successfully, else false if it fails at
           any
92  * point.
93  */
94 public boolean readCourses(String fileName, Event event) throws
           IOException {
95     String input;
96     char courseLetter;
97     int numberOfNodes;
98     int[] nodes;
99     String[] subStrings;
100     String pattern = "(([A-Za-z]+)((\\s+\\d+)+)$)"; //Regular expression
           for courses file.
101

```

```

102     try {
103         BufferedReader reader = new BufferedReader(new FileReader(
104             fileName));
105
106         while ((input = reader.readLine()) != null) {
107             if (input.matches(pattern)) { //Checks to make sure the line
108                 is in the right format.
109                 subStrings = input.split("\\s+"); //Gets rid of
110                 whitespace and separates the strings into substrings.
111                 courseLetter = subStrings[0].charAt(0); //Retrieves the
112                 course letter.
113                 numberOfNodes = Integer.parseInt(subStrings[1]);
114                 nodes = new int[numberOfNodes];
115
116                 for (int counter = 0; counter < numberOfNodes; counter
117                     ++){
118                     if (event.checkNodeExists(Integer.parseInt(
119                         subStrings[counter + 2]))) {
120                         nodes[counter] = Integer.parseInt(subStrings[
121                             counter + 2]);
122                     } else {
123                         System.out.print("Invalid node in course file
124                             found. Cancelling loading of courses\n\n");
125                         reader.close();
126                         return false;
127                     }
128                 }
129
130                 Course course = new Course(courseLetter, numberOfNodes,
131                     nodes); //Creates new course with parameters read in.
132                 event.getCourses().add(course); //Adds new course to
133                 array list of courses.
134             } else {
135                 System.out.print("Invalid line format. Cancelling
136                     loading of courses\n\n");
137                 reader.close();
138                 return false;
139             }
140         }
141
142         if (!event.getCourses().isEmpty()) {
143             System.out.print("Loading in of courses successful.\n\n");
144             reader.close();
145             return true;
146         } else {
147             System.out.print("Loading in of courses unsuccessful. No
148                 courses in file.\n\n");
149             reader.close();
150             return false;
151         }
152     } catch (FileNotFoundException ex) {
153         Logger.getLogger(FileHandler.class.getName()).log(Level.SEVERE,
154             null, ex);
155     }
156
157     System.out.print("Could not open file that contains courses.\n\n");
158     return false;
159 }
160
161 /**
162  * Method to read in all the details for the competitors pertaining to
163  * an
164  * event.

```



```

151 *
152 * @param fileName The file name required to access the file needed.
153 * @param event The event instance.
154 * @return True if file loaded successfully, else false if it fails at
155 * any
156 * point.
157 */
158 public boolean readCompetitors(String fileName, Event event) throws
159 IOException {
160     String input;
161     int competitorNumber;
162     char courseLetter;
163     String[] subStrings;
164     String competitorName;
165     String pattern = "(\\d+\\s+[A-Za-z](\\s+[A-Za-z]{1}[a-z]+)+)$"; //
166     Regular expression for competitors file.
167
168     try {
169         BufferedReader reader = new BufferedReader(new FileReader(
170             fileName));
171
172         while ((input = reader.readLine()) != null) {
173             if (input.matches(pattern)) { //Checks to make sure the line
174                 is in the right format.
175                 subStrings = input.split("\\s+"); //Gets rid of
176                 whitespace and separates the strings into substrings.
177                 competitorNumber = Integer.parseInt(subStrings[0]); //
178                 Retrieves the competitor number by parsing the string
179                 into an int.
180
181                 if (event.checkCourseExists(subStrings[1].charAt(0))) {
182                     courseLetter = subStrings[1].charAt(0); //Retrieves
183                     the course the competitor is entering in on.
184                 } else {
185                     System.out.print("Invalid course in competitor file
186                     found. Cancelling loading of competitors.\n\n");
187                     reader.close();
188                     return false;
189                 }
190
191                 competitorName = subStrings[2];
192
193                 if (subStrings.length > 3) {
194                     for (int counter = 3; counter < subStrings.length;
195                         counter++) {
196                         competitorName += " " + subStrings[counter]; //
197                         Concatenates name substrings together.
198                     }
199                 }
200
201                 Competitor competitor = new Competitor(competitorNumber,
202                     courseLetter, competitorName, event); //Creates new
203                 competitor with parameters read in.
204                 event.getCompetitors().add(competitor); //Adds new
205                 competitor to array list of competitors.
206             } else {
207                 System.out.print("Invalid line format. Cancelling
208                 loading of competitors.\n\n");
209                 reader.close();
210                 return false;
211             }
212         }
213     }
214 }

```

```

198         if (!event.getCompetitors().isEmpty()) {
199             System.out.print("Loading in of competitors successful.\n\n"
200                 );
201             reader.close();
202             return true;
203         } else {
204             System.out.print("Loading in of competitors unsuccessful. No
205                 competitors in file.\n\n");
206             reader.close();
207             return false;
208         }
209     } catch (FileNotFoundException ex) {
210         Logger.getLogger(FileHandler.class
211             .getName()).log(Level.SEVERE, null, ex);
212     }
213
214     System.out.print("Could not open file that contains competitors.\n\n
215         ");
216     return false;
217 }
218
219 /**
220  * Method to read in all the details for the checkpoint times pertaining
221  * to
222  * an event.
223  *
224  * @param fileName The file name required to access the file needed.
225  * @param event The event instance.
226  * @return True if file loaded successfully, else false if it fails at
227  * any
228  * point.
229  */
230 public boolean readTimes(String fileName, Event event) throws
231     IOException, ParseException {
232     String input;
233     int currentLineNumber = 0;
234     int lastLineNumber = event.getLastLineRead();
235     char competitorStatus;
236     int competitorNumber;
237     int nodeNumber;
238     String[] subStrings;
239     String pattern = "([A-Z{1}]((\\s+\\d+){2})\\s
240         +[0-2{1}][0-9{1}]:[0-5{1}][0-9{1}]$)"; //Regular expression for
241         times file.
242     SimpleDateFormat formatter = new SimpleDateFormat("HH:mm");
243     Date time;
244
245     event.getRecords().clear(); //Empties array list.
246
247     try {
248         FileChannel channel = new RandomAccessFile(fileName, "rw").
249             getChannel(); //Creates a channel for the file.
250         FileLock lock = channel.lock(); //Blocks/Halts thread until lock
251             aquired.
252
253         BufferedReader reader = new BufferedReader(new FileReader("
254             cp_times.txt"));
255
256         while ((input = reader.readLine()) != null) {
257             currentLineNumber++;
258             if (currentLineNumber > lastLineNumber) {
259                 if (input.matches(pattern)) { //Checks to make sure the
260                     line is in the right format.

```

```

249         subStrings = input.split("[\\s+]"); //Gets rid of
           whitespace and separates the strings into
           substrings.
250         competitorStatus = subStrings[0].charAt(0); //
           Retrieves competitor status.
251         nodeNumber = Integer.parseInt(subStrings[1]); //
           Retrieves the node number by parsing the string
           into an int.
252         competitorNumber = Integer.parseInt(subStrings[2]);
           //Retrieves the competitor number by parsing the
           string into an int.
253         time = formatter.parse(subStrings[3]); //Retrieves
           the time being recorded and formats it into 24
           hour HH:MM.

254
255         Competitor competitor = event.retrieveCompetitor(
           competitorNumber);
256         if (competitor.getStatus() == 'T') {
257             competitor.incrementCheckpointIndex(); //
           Increments the competitor's checkpoint index
           by 1.
258         }
259
260         Record record = new Record(competitorStatus,
           nodeNumber, competitorNumber, time); //Creates
           new record with parameters read in.
261         event.getRecords().add(record); //Adds new record to
           array list of records.
262         competitor.setStatus(competitorStatus); //Updates
           competitor's status.

263
264         event.setLastLineRead(currentLineNumber);
265         event.setLastRecordedTime(time);
266     } else {
267         System.out.print("Invalid line format. Cancelling
           loading of times.\n\n");
268         reader.close();
269         lock.release();
270         channel.close();
271         return false;
272     }
273 }
274 }
275
276     event.setTimesFilesExistsTrue(); //Lets the event instance know
           that an event does exist.
277     reader.close(); //Closes reader.
278     lock.release(); //Releases file lock.
279     channel.close(); //Closes channel ensuring lock release and
           release of resources.
280     return true;
281 } catch (FileNotFoundException ex) {
282     System.out.print("Could not open file that contains times.\n\n");
           ;
283 }
284 return false;
285 }
286
287 /**
288  * Method to write a record on a line in the time records file.
289  *
290  * @param fileName The file name required to access the file needed.
291  * @param record The record to be written.

```

```

292     * @return True if file written to successfully, else false if it fails
293     * at
294     */
295     public boolean appendTimeRecord(String fileName, Record record) {
296         SimpleDateFormat formatter = new SimpleDateFormat("HH:mm");
297
298         try {
299             FileChannel channel = new RandomAccessFile(fileName, "rw").
300                 getChannel(); //Creates a channel for the file.
301             FileLock lock = channel.lock();
302
303             FileWriter writer = new FileWriter(fileName, true); //True sets
304                 append mode.
305             writer.write(record.getCompetitorStatus() + " " + record.
306                 getCheckpointNumber()
307                 + " " + record.getCompetitorNumber() + " " + formatter.
308                 format(record.getTime()) + "\n");
309             writer.close();
310             lock.release();
311             channel.close();
312             return true;
313         } catch (IOException ex) {
314             System.out.print("\nCould not open file for writing.\n\n");
315         }
316         return false;
317     }
318 }

```

Listing 16: TypeWindow class.

```

1  /* File Name: TypeWindow.java
2   * Description: TypeWindow GUI class using swing.
3   * First Created: 17/03/2013
4   * Last Modified: 18/03/2013
5   */
6  package GUI;
7
8  import Data_Structures.Event;
9  import java.awt.BorderLayout;
10 import java.awt.Dimension;
11 import java.awt.event.ActionEvent;
12 import java.awt.event.ActionListener;
13 import javax.swing.ButtonGroup;
14 import javax.swing.ImageIcon;
15 import javax.swing.JButton;
16 import javax.swing.JFrame;
17 import javax.swing.JLabel;
18 import javax.swing.JPanel;
19 import javax.swing.JRadioButton;
20 import javax.swing.border.EmptyBorder;
21
22 /**
23  * @author Chris Savill, chs17@aber.ac.uk
24  */
25 public class TypeWindow extends JFrame implements ActionListener {
26
27     private Event event;
28     private boolean medicalSelected;
29     private JFrame typeFrame;
30     private JPanel typePanel, bottomPanel;
31     private JLabel typeLabel;
32     private JRadioButton time, medical;

```

```

33 private ButtonGroup typeGroup;
34 private JButton next;
35
36 /**
37  * Constructor for TypeWindow GUI class that sets up and launches GUI.
38  *
39  * @param event The event instance.
40  */
41 public TypeWindow(Event event) {
42     this.event = event;
43     medicalSelected = false;
44
45     //Setup frame:
46     typeFrame = new JFrame("Checkpoint Type Selection");
47     typeFrame.setPreferredSize(new Dimension(300, 200));
48     typeFrame.setLocation(400, 200);
49     typeFrame.setLayout(new BorderLayout());
50     typeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Sets the
        default close operation
51     typeFrame.setIconImage(new ImageIcon("horse.jpg").getImage()); //
        Loads an image and sets it as the frame icon
52     //////////////////////////////////////
53
54     //Setup panels:
55     typePanel = new JPanel(new BorderLayout()); //Creates new JPanel.
56     typePanel.setBorder(new EmptyBorder(25, 25, 25, 25)); //Sets an
        invisible border to simulate a padding effect
57     typeFrame.add(typePanel, BorderLayout.NORTH); //Adds panel to frame
        and places it in NORTH container.
58     bottomPanel = new JPanel();
59     typeFrame.add(bottomPanel, BorderLayout.SOUTH); //Adds panel to
        frame and places it in SOUTH container.
60     //////////////////////////////////////
61
62     //Setup checkpoint panel components:
63     typeLabel = new JLabel("Select Checkpoint Type Below: ");
64     typePanel.add(typeLabel, BorderLayout.NORTH);
65
66     time = new JRadioButton("Time Checkpoint");
67     time.setActionCommand("time");
68     time.addActionListener(this);
69     time.setSelected(true); //Defaults this button to be selected.
70     typePanel.add(time, BorderLayout.CENTER);
71     medical = new JRadioButton("Medical Checkpoint");
72     medical.setActionCommand("medical");
73     medical.addActionListener(this);
74     medical.setSelected(false);
75     typePanel.add(medical, BorderLayout.SOUTH);
76
77     typeGroup = new ButtonGroup(); //Creates a group for the radio
        buttons to prevent both from being selected.
78     typeGroup.add(time);
79     typeGroup.add(medical);
80     //////////////////////////////////////
81
82     //Setup bottom panel components:
83     next = new JButton("Next");
84     next.setPreferredSize(new Dimension(100, 50));
85     bottomPanel.add(next);
86     next.addActionListener(this);
87     //////////////////////////////////////
88
89     //Finialise frame setup:

```

```

90         typeFrame.pack();
91         typeFrame.setVisible(true); //Makes the frame visible
92         //////////////////////////////////////
93     }
94
95     /**
96      * Method to handle actions performed.
97      *
98      * @param evt The event triggered.
99      */
100    @Override
101    public void actionPerformed(ActionEvent evt) {
102        String actionCommand = evt.getActionCommand();
103
104        switch (actionCommand) {
105            case "Next":
106                if (medicalSelected == true) {
107                    typeFrame.setVisible(false);
108                    SelectionWindow selectionWindow = new SelectionWindow(
109                        event, "MC", typeFrame);
110                } else {
111                    typeFrame.setVisible(false);
112                    SelectionWindow selectionWindow = new SelectionWindow(
113                        event, "CP", typeFrame);
114                }
115
116                typeFrame.dispose();
117                this.dispose();
118                break;
119            case "time":
120                medicalSelected = false;
121                break;
122            case "medical":
123                medicalSelected = true;
124                break;
125        }
126    }
127 }

```

Listing 17: SelectionWindow class.

```

1  /* File Name: SelectionWindow.java
2   * Description: SelectionWindow GUI class using swing.
3   * First Created: 16/03/2013
4   * Last Modified: 17/03/2013
5   */
6  package GUI;
7
8  import Data_Structures.Competitor;
9  import Data_Structures.Event;
10 import Data_Structures.Node;
11 import java.awt.BorderLayout;
12 import java.awt.Color;
13 import java.awt.Dimension;
14 import java.awt.event.ActionEvent;
15 import java.awt.event.ActionListener;
16 import javax.swing.DefaultListModel;
17 import javax.swing.ImageIcon;
18 import javax.swing.JButton;
19 import javax.swing.JFrame;
20 import javax.swing.JLabel;
21 import javax.swing.JList;
22 import javax.swing.JOptionPane;

```

```

23 import javax.swing.JPanel;
24 import javax.swing.JScrollPane;
25 import javax.swing.ScrollPaneConstants;
26 import javax.swing.border.EmptyBorder;
27 import javax.swing.border.LineBorder;
28 import javax.swing.event.ListSelectionEvent;
29 import javax.swing.event.ListSelectionListener;
30
31 /**
32  * @author Chris Savill, chs17@aber.ac.uk
33  */
34 public class SelectionWindow extends JFrame implements ActionListener,
    ListSelectionListener {
35
36     private Event event;
37     private int checkpoint;
38     private String type;
39     private int competitor;
40     private boolean checkpointSelected = false;
41     private boolean competitorSelected = false;
42     private JFrame selectionFrame, typeFrame;
43     private JPanel checkpointPanel, competitorPanel, bottomPanel;
44     private JLabel checkpointLabel, competitorLabel;
45     private DefaultListModel checkpointListModel, competitorListModel;
46     private JList checkpointList, competitorList;
47     private JScrollPane checkpointListScrollBar, competitorListScrollBar;
48     private JButton next;
49
50     /**
51      * Constructor for SelectionWindow GUI class, sets up and runs GUI.
52      * @param event The event instance.
53      * @param type The type of the checkpoint.
54      * @param typeFrame The JFrame this transitioned from.
55      */
56     public SelectionWindow(Event event, String type, JFrame typeFrame) {
57         typeFrame.dispose();
58         this.typeFrame = typeFrame;
59         this.event = event;
60         this.type = type;
61
62         //Setup frame:
63         selectionFrame = new JFrame("Checkpoint and Competitor Selection");
64         selectionFrame.setLocation(400, 200);
65         selectionFrame.setLayout(new BorderLayout());
66         selectionFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //
        Sets the default close operation
67         selectionFrame.setIconImage(new ImageIcon("horse.jpg").getImage());
        //Loads an image and sets it as the frame icon
68         //////////////////////////////////////
69
70         //Setup panels:
71         checkpointPanel = new JPanel(new BorderLayout()); //Creates new
        JPanel.
72         checkpointPanel.setBorder(new EmptyBorder(10, 25, 10, 25)); //Sets
        an invisible border to simulate a padding effect
73         selectionFrame.add(checkpointPanel, BorderLayout.WEST); //Adds panel
        to frame and places it in WEST container.
74         competitorPanel = new JPanel(new BorderLayout());
75         competitorPanel.setBorder(new EmptyBorder(10, 25, 10, 25));
76         selectionFrame.add(competitorPanel, BorderLayout.EAST); //Adds panel
        to frame and places it in EAST container.
77         bottomPanel = new JPanel();
78         selectionFrame.add(bottomPanel, BorderLayout.SOUTH); //Adds panel to

```



```

79         frame and places it in SOUTH container.
80         //////////////////////////////////////
81         //Setup checkpoint panel components:
82         checkpointLabel = new JLabel("Select Checkpoint Below: ");
83         checkpointPanel.add(checkpointLabel, BorderLayout.NORTH);
84
85         checkpointListModel = new DefaultListModel();
86         checkpointList = new JList(checkpointListModel);
87         checkpointList.setBorder(new LineBorder(Color.BLACK));
88         checkpointPanel.add(checkpointList, BorderLayout.CENTER);
89         checkpointList.addListSelectionListener(this);
90
91         checkpointListScrollBar = new JScrollPane(checkpointList);
92         checkpointListScrollBar.setPreferredSize(new Dimension(50, 100));
93         checkpointListScrollBar.setVerticalScrollBarPolicy(
94             JScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED); //Adds
95             vertical scrollbar to JList
96         checkpointListScrollBar.setHorizontalScrollBarPolicy(
97             JScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED); //Adds
98             horizontal scrollbar to JList
99         checkpointPanel.add(checkpointListScrollBar);
100         //////////////////////////////////////
101
102         //Setup competitor panel components:
103         competitorLabel = new JLabel("Select Competitor Below: ");
104         competitorPanel.add(competitorLabel, BorderLayout.NORTH);
105
106         competitorListModel = new DefaultListModel();
107         competitorList = new JList(competitorListModel);
108         competitorList.setBorder(new LineBorder(Color.BLACK));
109         competitorPanel.add(competitorList, BorderLayout.CENTER);
110         competitorList.addListSelectionListener(this);
111
112         competitorListScrollBar = new JScrollPane(competitorList);
113         competitorListScrollBar.setPreferredSize(new Dimension(400, 300));
114         competitorListScrollBar.setVerticalScrollBarPolicy(
115             JScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED); //Adds
116             vertical scrollbar to JList
117         competitorListScrollBar.setHorizontalScrollBarPolicy(
118             JScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED); //Adds
119             horizontal scrollbar to JList
120         competitorPanel.add(competitorListScrollBar);
121         //////////////////////////////////////
122
123         //Setup bottom panel components:
124         next = new JButton("Next");
125         next.setPreferredSize(new Dimension(100, 50));
126         bottomPanel.add(next);
127         next.addActionListener(this);
128         //////////////////////////////////////
129
130         //Finialise frame setup:
131         addCheckpoints();
132         addCompetitors();
133         selectionFrame.pack();
134         selectionFrame.setVisible(true); //Makes the frame visible
135         //////////////////////////////////////
136     }
137
138     /**
139      * Method that adds the checkpoint checkpoints to the checkpoint JList
140      */

```



```

133 public void addCheckpoints() {
134     checkpointListModel.removeAllElements();
135
136     for (Node currentCheckpoint : event.getCheckpoints()) {
137         if (currentCheckpoint.getType().equals(type)) {
138             checkpointListModel.addElement(currentCheckpoint.getNumber()
139                 + ": " + currentCheckpoint.getType());
140         }
141     }
142
143     /**
144      * Method that adds the competitors to the competitor JList
145      */
146     public void addCompetitors() {
147         competitorListModel.removeAllElements();
148
149         for (Competitor currentCompetitor : event.getCompetitors()) {
150             competitorListModel.addElement("Competitor: " +
151                 currentCompetitor.getNumber()
152                 + " Course: " + currentCompetitor.getCourse() + "
153                 Name: " + currentCompetitor.getName());
154         }
155     }
156
157     /**
158      * Method to handle actions performed.
159      * @param evt The event triggered.
160      */
161     @Override
162     public void actionPerformed(ActionEvent evt) {
163         String actionCommand = evt.getActionCommand();
164
165         if (actionCommand.equals("Next")) {
166             if (checkpointSelected == true && competitorSelected == true) {
167                 selectionFrame.setVisible(false);
168                 TimeWindow timeWindow = new TimeWindow(event, checkpoint,
169                     type, competitor, selectionFrame, typeFrame);
170                 selectionFrame.dispose();
171                 this.dispose();
172             } else {
173                 JOptionPane.showMessageDialog(selectionFrame, "Please select
174                     both a checkpoint and competitor.");
175             }
176         }
177     }
178
179     /**
180      * Method to handle values changing in a JList.
181      * @param evt The event triggered.
182      */
183     @Override
184     public void valueChanged(ListSelectionEvent evt) {
185         if (!evt.getValueIsAdjusting()) {
186             JList list = (JList) evt.getSource();
187
188             if (list.equals(checkpointList)) {
189                 checkpoint = event.retrieveCheckpointNumber(type, list.

```

```

190         getSelectedIndex()).getNumber();
191         competitorSelected = true;
192     }
193 }
194 }

```

Listing 18: TimeWindow class.

```

1  /* File Name: TimeWindow.java
2   * Description: TimeWindow GUI class using swing.
3   * First Created: 16/03/2013
4   * Last Modified: 17/03/2013
5   */
6  package GUI;
7
8  import Data_Structures.Event;
9  import Data_Structures.Record;
10 import File_Handling.FileHandler;
11 import java.awt.BorderLayout;
12 import java.awt.Dimension;
13 import java.awt.event.ActionEvent;
14 import java.awt.event.ActionListener;
15 import java.io.IOException;
16 import java.text.ParseException;
17 import java.util.Calendar;
18 import java.util.Date;
19 import java.util.logging.Level;
20 import java.util.logging.Logger;
21 import javax.swing.ImageIcon;
22 import javax.swing.JButton;
23 import javax.swing.JFrame;
24 import javax.swing.JLabel;
25 import javax.swing.JOptionPane;
26 import javax.swing.JPanel;
27 import javax.swing.JSpinner;
28 import javax.swing.SpinnerDateModel;
29 import javax.swing.border.EmptyBorder;
30
31 /**
32  * @author Chris Savill, chs17@aber.ac.uk
33  */
34 public class TimeWindow extends JFrame implements ActionListener {
35
36     private Event event;
37     private FileHandler fileHandler;
38     private int checkpoint;
39     private String type;
40     private int competitor;
41     private int status;
42     private JFrame timeFrame, typeFrame;
43     private JPanel timePanel, bottomPanel;
44     private JLabel timeLabel;
45     private JButton submit;
46     private Date date;
47     private SpinnerDateModel spinnerModel;
48     private JSpinner spinner;
49     private JSpinner.DateEditor dateEditor;
50
51     /**
52      * Constructor for TimeWindow GUI class that sets up and launches the
53      * GUI.
54      */

```

```

54      * @param event The event instance.
55      * @param checkpoint The checkpoint number.
56      * @param type The checkpoint type.
57      * @param competitor The competitor number.
58      * @param selectionFrame The JFrame this transitioned from.
59      * @param typeFrame The JFrame that is reopened after this JFrame closes
60      */
61      public TimeWindow(Event event, int checkpoint, String type, int
62          competitor, JFrame selectionFrame, JFrame typeFrame) {
63          selectionFrame.dispose();
64
65          this.typeFrame = typeFrame;
66          this.event = event;
67          this.checkpoint = checkpoint;
68          this.type = type;
69          this.competitor = competitor;
70          fileHandler = new FileHandler();
71
72          //Setup frame:
73          timeFrame = new JFrame("Time Of Record");
74
75          if (type.equals("MC")) {
76              status = getMedicalOptions();
77          } else {
78              status = 0; //Comeptitor status not a medical related status.
79          }
80
81          timeFrame.setLocation(400, 200);
82          timeFrame.setLayout(new BorderLayout());
83          timeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Sets the
84              default close operation
85          timeFrame.setIconImage(new ImageIcon("horse.jpg").getImage()); //
86              Loads an image and sets it as the frame icon
87          ///////////////////////////////////////////////////
88
89          //Setup panels:
90          timePanel = new JPanel(new BorderLayout()); //Creates new JPanel.
91          timePanel.setBorder(new EmptyBorder(10, 25, 10, 25)); //Sets an
92              invisible border to simulate a padding effect
93          timeFrame.add(timePanel, BorderLayout.WEST); //Adds panel to frame
94              and places it in WEST container.
95          bottomPanel = new JPanel();
96          timeFrame.add(bottomPanel, BorderLayout.SOUTH); //Adds panel to
97              frame and places it in SOUTH container.
98          ///////////////////////////////////////////////////
99
100         //Setup checkpoint panel components:
101         timeLabel = new JLabel("Select Time Below: ");
102         timePanel.add(timeLabel, BorderLayout.NORTH);
103
104         date = new Date();
105         spinnerModel = new SpinnerDateModel(date, null, null, Calendar.
106             HOUR_OF_DAY);
107         spinner = new JSpinner(spinnerModel);
108         dateEditor = new JSpinner.DateEditor(spinner, "HH:mm"); //24-hour
109             format.
110         spinner.setEditor(dateEditor);
111         timePanel.add(spinner, BorderLayout.CENTER);
112         ///////////////////////////////////////////////////
113
114         //Setup bottom panel components:
115         submit = new JButton("Submit Checkpoint Record");

```

```

108         submit.setPreferredSize(new Dimension(225, 30));
109         bottomPanel.add(submit);
110         submit.addActionListener(this);
111         //////////////////////////////////////////
112
113         //Finialise frame setup:
114         timeFrame.pack();
115         timeFrame.setVisible(true); //Makes the frame visible
116         //////////////////////////////////////////
117     }
118
119     /**
120      * Method to handle actions performed.
121      *
122      * @param evt The event triggered.
123      */
124     @Override
125     public void actionPerformed(ActionEvent evt) {
126         String actionCommand = evt.getActionCommand();
127
128         if (actionCommand.equals("Submit Checkpoint Record")) {
129             try {
130                 if (!fileHandler.readTimes(event.getFileNames()[3], event))
131                     {
132                         JOptionPane.showMessageDialog(timeFrame, "Failed to load
133                             time records from file.");
134                     }
135             } catch (IOException | ParseException ex) {
136                 Logger.getLogger(TimeWindow.class.getName()).log(Level.
137                     SEVERE, null, ex);
138             }
139
140             if (event.checkNewRecord(checkpoint, status, competitor, (Date)
141                 spinner.getValue())) {
142                 char finalStatus = event.determineFinalStatus(checkpoint,
143                     status, competitor);
144
145                 Record record = new Record(checkpoint, finalStatus,
146                     competitor, (Date) spinner.getValue());
147                 event.getRecords().add(record);
148
149                 fileHandler.appendTimeRecord(event.getFileNames()[3], record
150                     );
151                 JOptionPane.showMessageDialog(timeFrame, "Time record
152                     succesfully added.");
153             } else {
154                 JOptionPane.showMessageDialog(timeFrame, "Non-valid record.
155                     Record will not added.");
156             }
157
158             timeFrame.dispose(); //Closes frame and releases resources.
159             this.dispose(); //Releases resources.
160             TypeWindow typeFrame = new TypeWindow(event);
161         }
162     }
163
164     /**
165      * Method to get the user to select the status of the competitor at the
166      * medical checkpoint.
167      *
168      * @return The status of the competitor at the medical checkpoint.
169      */

```

```

162     public int getMedicalOptions() {
163         String[] options = new String[]{"Arriving", "Departing", "Excluded"};
164     }
165
166     int selection = JOptionPane.showOptionDialog(timeFrame, "Is the
167         competitor being marked as 'Arriving',"
168         + " 'Departing' or as 'Excluded' on medical grounds?", "
169         Medical Marking", JOptionPane.DEFAULT_OPTION,
170         JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
171
172     if (selection == 0) {
173         return 1; //Competitor status to be set to arriving at medical
174         checkpoint.
175     } else if (selection == 1) {
176         return 2; //Competitor status to be set to departing medical
177         checkpoint.
178     } else if (selection == 2) {
179         return 3; //Competitor status to be set to excluded based on
180         medical grounds.
181     }
182
183     return 0;
184 }
185 }

```

7 Clean build and compilation of Checkpoint Program

```

ant -f /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program clean jar
init:
deps-clean:
Updating property file: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build/built-clean.properties
Deleting directory /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build
clean:
init:
deps-jar:
Created dir: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build
Updating property file: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build/built-jar.properties
Created dir: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build/classes
Created dir: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build/empty
Created dir: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build/generated-sources/ap-source-output
Compiling 10 source files to /home/clsavill/GitHub/Runners_and_Riders_3_Part
/Checkpoint_Manager_Program/build/classes
Note: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/src/GUI/SelectionWindow.java uses unchecked or
unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
compile:
Created dir: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/dist
Copying 1 file to /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/build
Nothing to copy.
Building jar: /home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/dist/Checkpoint_Manager_Program.jar

```

```
To run this application from the command line without Ant, try:
java -jar "/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Checkpoint_Manager_Program/dist/Checkpoint_Manager_Program.jar"
jar:
BUILD SUCCESSFUL (total time: 2 seconds)
```

8 Run through of Checkpoint Manager Program

9 Files created by execution of Event Creation Program

10 Clean build and compilation of Event Manager Program

```
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .clean-
conf
make[1]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Manager_Program'
rm -f -r build/Debug
rm -f dist/Debug/GNU-Linux-x86/event_manager_program
make[1]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Manager_Program'
```

CLEAN SUCCESSFUL (total time: 57ms)

```
"/usr/bin/make" -f nbproject/Makefile-Debug.mk QMAKE= SUBPROJECTS= .build-
conf
make[1]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Manager_Program'
"/usr/bin/make" -f nbproject/Makefile-Debug.mk dist/Debug/GNU-Linux-x86/
event_manager_program
make[2]: Entering directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part
/Event_Manager_Program'
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/loader.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/loader.o.d -o build/
Debug/GNU-Linux-x86/loader.o loader.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/logger.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/logger.o.d -o build/
Debug/GNU-Linux-x86/logger.o logger.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/updater.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/updater.o.d -o build/
Debug/GNU-Linux-x86/updater.o updater.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/courses.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/courses.o.d -o build/
Debug/GNU-Linux-x86/courses.o courses.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/competitors.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/competitors.o.d -o build
/Debug/GNU-Linux-x86/competitors.o competitors.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/nodes.o.d
gcc -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/nodes.o.d -o build/Debug
/GNU-Linux-x86/nodes.o nodes.c
mkdir -p build/Debug/GNU-Linux-x86
rm -f build/Debug/GNU-Linux-x86/main.o.d
```

```

gcc      -c -g -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/event.o.d
gcc      -c -g -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/tracks.o.d
gcc      -c -g -MMD -MP -MF build/Debug/GNU-Linux-x86/tracks.o.d -o build/
Debug/GNU-Linux-x86/tracks.o tracks.c
mkdir -p dist/Debug/GNU-Linux-x86
gcc      -o dist/Debug/GNU-Linux-x86/event_manager_program build/Debug/GNU-
Linux-x86/loader.o build/Debug/GNU-Linux-x86/logger.o build/Debug/GNU-
Linux-x86/updater.o build/Debug/GNU-Linux-x86/courses.o build/Debug/GNU-
Linux-x86/competitors.o build/Debug/GNU-Linux-x86/nodes.o build/Debug/GNU
-Linux-x86/main.o build/Debug/GNU-Linux-x86/event.o build/Debug/GNU-Linux
-x86/tracks.o
make[2]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Manager_Program'
make[1]: Leaving directory '/home/clsavill/GitHub/Runners_and_Riders_3_Part/
Event_Manager_Program'

```

BUILD SUCCESSFUL (total time: 857ms)

11 Run through of Event Manager Program

Event Monitoring Program Launching...

Please enter in the file path and name of the event file: Mission_Files/event_3/name.txt

Endurance Horse Race - The Main Event

27th June 2012

07:30

Event file loaded in successfully.

Event loading finished.

Please enter in the file path and name of the nodes file: Mission_Files/event_3/nodes.txt

Head Node: Number: 1, Type: 0 = CP

Node: Number: 2, Type: 1 = JN

Node: Number: 3, Type: 1 = JN

Node: Number: 4, Type: 0 = CP

Node: Number: 5, Type: 0 = CP

Node: Number: 6, Type: 1 = JN

Node: Number: 7, Type: 0 = CP

Node: Number: 8, Type: 1 = JN

Node: Number: 9, Type: 0 = CP

Node: Number: 10, Type: 1 = JN

Node: Number: 11, Type: 1 = JN

Node: Number: 12, Type: 1 = JN

Node: Number: 13, Type: 0 = CP

Node: Number: 14, Type: -13 = MC

Node: Number: 15, Type: 1 = JN

Node: Number: 16, Type: 1 = JN

Node: Number: 17, Type: 0 = CP

Node: Number: 18, Type: 1 = JN

Nodes file loaded in successfully.

Node loading finished.

Please enter in the file path and name of the tracks file: Mission_Files/event_3/tracks.txt

Head Track: Number: 1, Start: 1, End: 2, Max Time: 20
 Track: Number: 2, Start: 2, End: 3, Max Time: 10
 Track: Number: 3, Start: 3, End: 4, Max Time: 11
 Track: Number: 4, Start: 4, End: 5, Max Time: 15
 Track: Number: 5, Start: 5, End: 6, Max Time: 12
 Track: Number: 6, Start: 6, End: 8, Max Time: 10
 Track: Number: 7, Start: 6, End: 7, Max Time: 8
 Track: Number: 8, Start: 7, End: 10, Max Time: 12
 Track: Number: 9, Start: 8, End: 10, Max Time: 10
 Track: Number: 10, Start: 8, End: 9, Max Time: 5
 Track: Number: 11, Start: 3, End: 9, Max Time: 18
 Track: Number: 12, Start: 9, End: 12, Max Time: 20
 Track: Number: 13, Start: 2, End: 13, Max Time: 30
 Track: Number: 14, Start: 12, End: 13, Max Time: 5
 Track: Number: 15, Start: 10, End: 11, Max Time: 15
 Track: Number: 16, Start: 11, End: 12, Max Time: 5
 Track: Number: 17, Start: 11, End: 14, Max Time: 12
 Track: Number: 18, Start: 14, End: 15, Max Time: 15
 Track: Number: 19, Start: 15, End: 16, Max Time: 8
 Track: Number: 20, Start: 16, End: 17, Max Time: 8
 Track: Number: 21, Start: 17, End: 18, Max Time: 7
 Track: Number: 22, Start: 15, End: 18, Max Time: 5

Tracks file loaded in successfully.
 Track loading finished.

Please enter in the file path and name of the courses file: Mission_Files/event_3/courses.txt

Head Course: ID: A, Number of Nodes: 21, Nodes: [1,2,3,4,5,6,7,10,11,14,15,16,17,18,15,14,11,12,13,2,1]

Course: ID: B, Number of Nodes: 15, Nodes: [1,2,3,4,5,6,7,10,11,14,11,12,13,2,1]

Course: ID: C, Number of Nodes: 13, Nodes: [1,2,3,4,5,6,7,10,11,12,13,2,1]

Course: ID: D, Number of Nodes: 11, Nodes: [1,2,3,4,5,6,8,9,3,2,1]

Course: ID: E, Number of Nodes: 11, Nodes: [1,2,3,9,8,10,11,12,13,2,1]

Course: ID: F, Number of Nodes: 8, Nodes: [1,2,3,9,12,13,2,1]

Courses file loaded in successfully.
Course loading finished.

Please enter in the file path and name of the competitors file: Mission_Files/event_3/entrants.txt

Head Competitor: Number: 1, Course: E, Name: Ace Abbey
Competitor: Number: 3, Course: A, Name: Ace Fudge
Competitor: Number: 4, Course: C, Name: Amber Abbey
Competitor: Number: 5, Course: E, Name: Amber Fudge
Competitor: Number: 6, Course: D, Name: April Abbey
Competitor: Number: 7, Course: B, Name: April Fudge
Competitor: Number: 8, Course: F, Name: Ash Abbey
Competitor: Number: 9, Course: D, Name: Ash Fudge
Competitor: Number: 10, Course: A, Name: Asti Abbey
Competitor: Number: 11, Course: A, Name: Asti Fudge
Competitor: Number: 12, Course: C, Name: Autumn Abbey
Competitor: Number: 13, Course: B, Name: Autumn Fudge
Competitor: Number: 14, Course: A, Name: Barfields Marco Abbey
Competitor: Number: 16, Course: F, Name: Barfields Marco Fudge
Competitor: Number: 17, Course: B, Name: Basil Abbey
Competitor: Number: 18, Course: A, Name: Basil Fudge
Competitor: Number: 19, Course: C, Name: Beatrice Abbey
Competitor: Number: 20, Course: A, Name: Beatrice Fudge
Competitor: Number: 22, Course: D, Name: Beau Abbey
Competitor: Number: 23, Course: C, Name: Beau Fudge
Competitor: Number: 24, Course: B, Name: Bella Abbey
Competitor: Number: 26, Course: F, Name: Bella Fudge
Competitor: Number: 27, Course: F, Name: Black Jack Abbey
Competitor: Number: 28, Course: A, Name: Black Jack Fudge
Competitor: Number: 30, Course: B, Name: Blue Abbey
Competitor: Number: 31, Course: B, Name: Blue Fudge
Competitor: Number: 32, Course: A, Name: Bobby Abbey
Competitor: Number: 34, Course: E, Name: Bobby Fudge
Competitor: Number: 35, Course: C, Name: Bubbles Abbey
Competitor: Number: 36, Course: D, Name: Bubbles Fudge
Competitor: Number: 38, Course: A, Name: Captain Abbey
Competitor: Number: 39, Course: B, Name: Captain Fudge
Competitor: Number: 40, Course: D, Name: Chalkie Abbey
Competitor: Number: 41, Course: F, Name: Chalkie Fudge

Competitor: Number: 42, Course: E, Name: Copper Abbey
Competitor: Number: 44, Course: B, Name: Copper Fudge
Competitor: Number: 45, Course: C, Name: Diamond Abbey
Competitor: Number: 46, Course: B, Name: Diamond Fudge
Competitor: Number: 47, Course: E, Name: Dinky Abbey
Competitor: Number: 48, Course: F, Name: Dinky Fudge
Competitor: Number: 49, Course: B, Name: Ebony Abbey
Competitor: Number: 50, Course: C, Name: Ebony Fudge
Competitor: Number: 51, Course: C, Name: Ginger Abbey
Competitor: Number: 52, Course: F, Name: Ginger Fudge
Competitor: Number: 53, Course: A, Name: Goldie Abbey
Competitor: Number: 55, Course: E, Name: Goldie Fudge
Competitor: Number: 56, Course: F, Name: Honey Abbey
Competitor: Number: 57, Course: C, Name: Honey Fudge
Competitor: Number: 58, Course: A, Name: Izzy Abbey
Competitor: Number: 59, Course: A, Name: Izzy Fudge
Competitor: Number: 60, Course: A, Name: Jasmine Abbey
Competitor: Number: 61, Course: F, Name: Jasmine Fudge
Competitor: Number: 62, Course: D, Name: Lady Abbey
Competitor: Number: 64, Course: B, Name: Lady Fudge
Competitor: Number: 65, Course: C, Name: Lady Tara Abbey
Competitor: Number: 66, Course: B, Name: Lady Tara Fudge
Competitor: Number: 67, Course: B, Name: Lemon Abbey
Competitor: Number: 68, Course: E, Name: Lemon Fudge
Competitor: Number: 69, Course: F, Name: Lord Abbey
Competitor: Number: 70, Course: E, Name: Lord Fudge
Competitor: Number: 71, Course: A, Name: Lucky Abbey
Competitor: Number: 74, Course: E, Name: Lucky Fudge
Competitor: Number: 76, Course: D, Name: Lord Abbey
Competitor: Number: 77, Course: B, Name: Lord Fudge
Competitor: Number: 78, Course: F, Name: Maddy Abbey
Competitor: Number: 79, Course: A, Name: Maddy Fudge
Competitor: Number: 80, Course: D, Name: Magic Abbey
Competitor: Number: 81, Course: D, Name: Magic Fudge
Competitor: Number: 83, Course: A, Name: Major Abbey
Competitor: Number: 85, Course: A, Name: Major Fudge
Competitor: Number: 86, Course: B, Name: Mattie Abbey
Competitor: Number: 87, Course: A, Name: Mattie Fudge
Competitor: Number: 89, Course: B, Name: Prince Abbey
Competitor: Number: 90, Course: A, Name: Prince Fudge

Competitor: Number: 91, Course: B, Name: Princess Abbey
Competitor: Number: 92, Course: B, Name: Princess Fudge
Competitor: Number: 93, Course: D, Name: Rosie Abbey
Competitor: Number: 94, Course: B, Name: Rosie Fudge
Competitor: Number: 95, Course: F, Name: Ruby Abbey
Competitor: Number: 97, Course: C, Name: Ruby Fudge
Competitor: Number: 98, Course: C, Name: Sapphire Abbey
Competitor: Number: 100, Course: F, Name: Sapphire Fudge
Competitor: Number: 101, Course: C, Name: Scarlet Abbey
Competitor: Number: 102, Course: F, Name: Scarlet Fudge
Competitor: Number: 103, Course: D, Name: sienna Abbey
Competitor: Number: 106, Course: B, Name: sienna Fudge
Competitor: Number: 107, Course: F, Name: Silver Abbey
Competitor: Number: 108, Course: A, Name: Silver Fudge
Competitor: Number: 109, Course: A, Name: Smokey Abbey
Competitor: Number: 110, Course: D, Name: Smokey Fudge
Competitor: Number: 111, Course: E, Name: Snowy Abbey
Competitor: Number: 113, Course: C, Name: Snowy Fudge
Competitor: Number: 114, Course: A, Name: sonic Abbey
Competitor: Number: 115, Course: D, Name: sonic Fudge
Competitor: Number: 117, Course: A, Name: Summer Abbey
Competitor: Number: 118, Course: E, Name: Summer Fudge
Competitor: Number: 121, Course: B, Name: Tango Abbey
Competitor: Number: 122, Course: A, Name: Tango Fudge
Competitor: Number: 123, Course: B, Name: Topaz Abbey
Competitor: Number: 124, Course: F, Name: Topaz Fudge
Competitor: Number: 126, Course: D, Name: Zizou Abbey
Competitor: Number: 127, Course: F, Name: Zizou Fudge

Competitors file loaded in successfully.
Competitor loading finished.
Loading Cycle Finished.
Press enter to continue.

```
===== MAIN MENU =====  
|  
| 1: Query competitor for current location/status. |  
| 2: Display how many competitors have not started yet. |  
| 3: Display how many competitors are out on the courses. |
```

```
| 4: Display how many competitors have completed their course successfully. |
| 5: Read in a file of times at which competitors have reached time checkpoints. |
| 6: Display the result times for the successfully completed. |
| 7: Display the competitors who have been excluded. |
| 8: Exit program. |
| |
```

Please select from one of the options above (number): 5

Please enter in the file path and name of the time record file: Mission_Files/event_3/cp_times_1.txt

End of file reached.
Loading of times files complete.
Time record loading finished.

Press enter to continue.

```
===== MAIN MENU =====
|
| 1: Query competitor for current location/status. |
| 2: Display how many competitors have not started yet. |
| 3: Display how many competitors are out on the courses. |
| 4: Display how many competitors have completed their course successfully. |
| 5: Read in a file of times at which competitors have reached time checkpoints. |
| 6: Display the result times for the successfully completed. |
| 7: Display the competitors who have been excluded. |
| 8: Exit program. |
| |
```

Please select from one of the options above (number): 3

Printing competitors that are out on a course...

```
=====
| Number | Name | Course | Last Recorded Checkpoint | Presumed Location |
=====
| 001 | Ace Abbey | E | 13 | TN - 01 |
```

	003		Ace Fudge		A		14		TN - 18	
	004		Amber Abbey		C		13		TC - 13	
	005		Amber Fudge		E		13		TN - 13	
	006		April Abbey		D		09		TN - 02	
	007		April Fudge		B		14		A - 14	
	008		Ash Abbey		F		13		TN - 13	
	009		Ash Fudge		D		09		TN - 02	
	010		Asti Abbey		A		07		TN - 15	
	011		Asti Fudge		A		07		TN - 15	
	012		Autumn Abbey		C		07		TN - 15	
	013		Autumn Fudge		B		07		TN - 08	
	014		Barfields Marco Abbey		A		07		TN - 08	
	016		Barfields Marco Fudge		F		13		TN - 13	
	017		Basil Abbey		B		07		TN - 08	
	018		Basil Fudge		A		05		TN - 07	
	019		Beatrice Abbey		C		05		TN - 07	
	020		Beatrice Fudge		A		05		TN - 05	
	022		Beau Abbey		D		05		TN - 05	
	023		Beau Fudge		C		05		TN - 05	
	024		Bella Abbey		B		04		TN - 04	
	026		Bella Fudge		F		09		TN - 12	
	027		Black Jack Abbey		F		09		TN - 12	
	028		Black Jack Fudge		A		04		TN - 04	
	030		Blue Abbey		B		04		TN - 04	
	031		Blue Fudge		B		01		TN - 03	
	032		Bobby Abbey		A		04		TC - 04	
	034		Bobby Fudge		E		01		TN - 11	
	035		Bubbles Abbey		C		01		TN - 02	
	036		Bubbles Fudge		D		01		TN - 02	
	038		Captain Abbey		A		01		TN - 02	
	039		Captain Fudge		B		01		TN - 01	
	040		Chalkie Abbey		D		01		TN - 01	
	041		Chalkie Fudge		F		01		TN - 01	
	042		Copper Abbey		E		01		TN - 01	
	044		Copper Fudge		B		01		TN - 01	
	045		Diamond Abbey		C		01		TN - 01	
	046		Diamond Fudge		B		01		TN - 01	

=====
 Key: NS = Not Started, TC = Time Checkpoint, TN = Track Number,

A = Medical Checkpoint, D = Departed Medical Checkpoint.

Number of Competitors out on course: 38 out of 102

Current Event Time: 9:26.

Press enter to continue.

```
===== MAIN MENU =====
|
| 1: Query competitor for current location/status.
| 2: Display how many competitors have not started yet.
| 3: Display how many competitors are out on the courses.
| 4: Display how many competitors have completed their course successfully.
| 5: Read in a file of times at which competitors have reached time checkpoints.
| 6: Display the result times for the successfully completed.
| 7: Display the competitors who have been excluded.
| 8: Exit program.
|
=====
```

Please select from one of the options above (number): 5

Please enter in the file path and name of the time record file: Mission_Files/event_3/cp_times_2.txt

End of file reached.

Loading of times files complete.

Time record loading finished.

Press enter to continue.

```
===== MAIN MENU =====
|
| 1: Query competitor for current location/status.
| 2: Display how many competitors have not started yet.
| 3: Display how many competitors are out on the courses.
| 4: Display how many competitors have completed their course successfully.
| 5: Read in a file of times at which competitors have reached time checkpoints.
|
```

```
| 6: Display the result times for the successfully completed. |
| 7: Display the competitors who have been excluded.         |
| 8: Exit program.                                           |
|                                                             |
```

```
=====
Please select from one of the options above (number): 5
```

```
Please enter in the file path and name of the time record file:  Mission_Files/event_3/cp_times_3.txt
```

```
End of file reached.
Loading of times files complete.
Time record loading finished.
```

```
Press enter to continue.
```

```
===== MAIN MENU =====
|
| 1: Query competitor for current location/status.           |
| 2: Display how many competitors have not started yet.     |
| 3: Display how many competitors are out on the courses.   |
| 4: Display how many competitors have completed their course successfully. |
| 5: Read in a file of times at which competitors have reached time checkpoints. |
| 6: Display the result times for the successfully completed. |
| 7: Display the competitors who have been excluded.         |
| 8: Exit program.                                           |
|                                                             |
=====
```

```
Please select from one of the options above (number): 2
```

```
Printing competitors that have not yet started...
```

```
=====
| Number | Name                               | Course | Location |
=====
| 095    | Ruby Abbey                        | F      | NS       |
| 097    | Ruby Fudge                       | C      | NS       |
| 098    | Sapphire Abbey                   | C      | NS       |
```


	100		Sapphire Fudge		F		NS	
	101		Scarlet Abbey		C		NS	
	102		Scarlet Fudge		F		NS	
	103		sienna Abbey		D		NS	
	106		sienna Fudge		B		NS	
	107		Silver Abbey		F		NS	
	108		Silver Fudge		A		NS	
	109		Smokey Abbey		A		NS	
	110		Smokey Fudge		D		NS	
	111		Snowy Abbey		E		NS	
	113		Snowy Fudge		C		NS	
	114		sonic Abbey		A		NS	
	115		sonic Fudge		D		NS	
	117		Summer Abbey		A		NS	
	118		Summer Fudge		E		NS	
	121		Tango Abbey		B		NS	
	122		Tango Fudge		A		NS	
	123		Topaz Abbey		B		NS	
	124		Topaz Fudge		F		NS	
	126		Zizou Abbey		D		NS	
	127		Zizou Fudge		F		NS	

=====
Key: NS = Not Started.

Number of Competitors not started yet: 24 out of 102

Current Event Time: 11:39.

Press enter to continue.

===== MAIN MENU =====

- | | | |
|--|--|--|
| | | |
| | 1: Query competitor for current location/status. | |
| | 2: Display how many competitors have not started yet. | |
| | 3: Display how many competitors are out on the courses. | |
| | 4: Display how many competitors have completed their course successfully. | |
| | 5: Read in a file of times at which competitors have reached time checkpoints. | |
| | 6: Display the result times for the successfully completed. | |

```
| 7: Display the competitors who have been excluded. |
| 8: Exit program. |
| |
```

=====

Please select from one of the options above (number): 3

Printing competitors that are out on a course...

Number	Name	Course	Last Recorded Checkpoint	Presumed Location
010	Asti Abbey	A	13	TN - 01
011	Asti Fudge	A	13	TN - 01
014	Barfields Marco Abbey	A	13	TN - 13
018	Basil Fudge	A	13	TN - 13
020	Beatrice Fudge	A	13	TN - 13
028	Black Jack Fudge	A	14	TN - 20
032	Bobby Abbey	A	14	TN - 17
038	Captain Abbey	A	17	TN - 22
039	Captain Fudge	B	13	TN - 13
044	Copper Fudge	B	14	TN - 14
045	Diamond Abbey	C	13	TN - 01
049	Ebony Abbey	B	14	TN - 17
050	Ebony Fudge	C	13	TN - 13
051	Ginger Abbey	C	13	TN - 13
052	Ginger Fudge	F	13	TN - 01
055	Goldie Fudge	E	13	TN - 13
056	Honey Abbey	F	13	TN - 01
057	Honey Fudge	C	07	TN - 16
058	Izzy Abbey	A	07	TN - 17
060	Jasmine Abbey	A	07	TN - 15
061	Jasmine Fudge	F	13	TN - 13
062	Lady Abbey	D	09	TN - 11
064	Lady Fudge	B	07	TN - 08
065	Lady Tara Abbey	C	07	TN - 08
066	Lady Tara Fudge	B	07	TN - 08
067	Lemon Abbey	B	07	TN - 08
068	Lemon Fudge	E	09	TN - 15
069	Lord Abbey	F	13	TN - 13

070	Lord Fudge	E	09	TN - 15
071	Lucky Abbey	A	05	TN - 05
074	Lucky Fudge	E	09	TN - 09
076	Lord Abbey	D	05	TC - 05
077	Lord Fudge	B	04	TN - 04
078	Maddy Abbey	F	09	TN - 12
079	Maddy Fudge	A	04	TN - 04
080	Magic Abbey	D	01	TN - 03
081	Magic Fudge	D	01	TN - 03
083	Major Abbey	A	01	TN - 03
085	Major Fudge	A	01	TN - 02
086	Mattie Abbey	B	01	TN - 02
087	Mattie Fudge	A	01	TN - 02
089	Prince Abbey	B	01	TN - 01
090	Prince Fudge	A	01	TN - 01
091	Princess Abbey	B	01	TN - 01
092	Princess Fudge	B	01	TN - 01
093	Rosie Abbey	D	01	TN - 01
094	Rosie Fudge	B	01	TN - 01

=====
 Key: NS = Not Started, TC = Time Checkpoint, TN = Track Number,
 A = Medical Checkpoint, D = Departed Medical Checkpoint.

Number of Competitors out on course: 47 out of 102

Current Event Time: 11:39.

Press enter to continue.

===== MAIN MENU =====

1:	Query competitor for current location/status.
2:	Display how many competitors have not started yet.
3:	Display how many competitors are out on the courses.
4:	Display how many competitors have completed their course successfully.
5:	Read in a file of times at which competitors have reached time checkpoints.
6:	Display the result times for the successfully completed.
7:	Display the competitors who have been excluded.

| 8: Exit program. |

|

=====

Please select from one of the options above (number): 4

Printing competitors that have finished...

=====								
	Number		Name		Course		Location	
=====								
	001		Ace Abbey		E		CC	
	003		Ace Fudge		A		CC	
	004		Amber Abbey		C		CC	
	005		Amber Fudge		E		CC	
	006		April Abbey		D		CC	
	007		April Fudge		B		CC	
	008		Ash Abbey		F		CC	
	009		Ash Fudge		D		CC	
	012		Autumn Abbey		C		CC	
	013		Autumn Fudge		B		CC	
	016		Barfields Marco Fudge		F		CC	
	017		Basil Abbey		B		CC	
	019		Beatrice Abbey		C		CC	
	022		Beau Abbey		D		CC	
	024		Bella Abbey		B		CC	
	026		Bella Fudge		F		CC	
	027		Black Jack Abbey		F		CC	
	030		Blue Abbey		B		CC	
	031		Blue Fudge		B		CC	
	034		Bobby Fudge		E		CC	
	035		Bubbles Abbey		C		CC	
	040		Chalkie Abbey		D		CC	
	042		Copper Abbey		E		CC	
	047		Dinky Abbey		E		CC	
	048		Dinky Fudge		F		CC	
=====								

Number of Competitors completed course successfully: 25 out of 102

Current Event Time: 11:39.

Press enter to continue.

```
===== MAIN MENU =====
|
| 1: Query competitor for current location/status.
| 2: Display how many competitors have not started yet.
| 3: Display how many competitors are out on the courses.
| 4: Display how many competitors have completed their course successfully.
| 5: Read in a file of times at which competitors have reached time checkpoints.
| 6: Display the result times for the successfully completed.
| 7: Display the competitors who have been excluded.
| 8: Exit program.
|
=====
```

Please select from one of the options above (number): 6

Printing results...

```
=====
| Number | Name | Status | Time |
=====
| 001 | Ace Abbey | CC | 02:04 |
| 003 | Ace Fudge | CC | 03:52 |
| 004 | Amber Abbey | CC | 02:37 |
| 005 | Amber Fudge | CC | 02:11 |
| 006 | April Abbey | CC | 02:03 |
| 007 | April Fudge | CC | 02:46 |
| 008 | Ash Abbey | CC | 01:56 |
| 009 | Ash Fudge | CC | 01:58 |
| 012 | Autumn Abbey | CC | 02:30 |
| 013 | Autumn Fudge | CC | 02:53 |
| 016 | Barfields Marco Fudge | CC | 01:55 |
| 017 | Basil Abbey | CC | 02:49 |
| 019 | Beatrice Abbey | CC | 02:27 |
| 022 | Beau Abbey | CC | 02:02 |
| 024 | Bella Abbey | CC | 02:54 |
```

	026		Bella Fudge		CC		01:49	
	027		Black Jack Abbey		CC		01:49	
	030		Blue Abbey		CC		02:43	
	031		Blue Fudge		CC		02:44	
	034		Bobby Fudge		CC		02:03	
	035		Bubbles Abbey		CC		02:32	
	040		Chalkie Abbey		CC		02:03	
	042		Copper Abbey		CC		02:05	
	047		Dinky Abbey		CC		02:10	
	048		Dinky Fudge		CC		01:54	

Number of Competitors completed course successfully: 25 out of 102

Current Event Time: 11:39.

Press enter to continue.

===== MAIN MENU =====

|

| 1: Query competitor for current location/status.

| 2: Display how many competitors have not started yet.

| 3: Display how many competitors are out on the courses.

| 4: Display how many competitors have completed their course successfully.

| 5: Read in a file of times at which competitors have reached time checkpoints.

| 6: Display the result times for the successfully completed.

| 7: Display the competitors who have been excluded.

| 8: Exit program.

|

=====

Please select from one of the options above (number): 7

Printing excluded...

	Number		Name		Status		At Time	
	023		Beau Fudge		EI		09:49	

	036		Bubbles Fudge		EI		09:57	
	041		Chalkie Fudge		EI		11:05	
	046		Diamond Fudge		EI		11:13	
	059		Izzy Fudge		EI		11:10	

=====
Number of Competitors excluded: 5 out of 102

Key: EI = Excluded for taking an Incorrect Route, EM = Excluded for Medical Safety Reasons.

Current Event Time: 11:39.

Press enter to continue.

===== MAIN MENU =====

	1: Query competitor for current location/status.	
	2: Display how many competitors have not started yet.	
	3: Display how many competitors are out on the courses.	
	4: Display how many competitors have completed their course successfully.	
	5: Read in a file of times at which competitors have reached time checkpoints.	
	6: Display the result times for the successfully completed.	
	7: Display the competitors who have been excluded.	
	8: Exit program.	

=====

Please select from one of the options above (number): 8

Exiting Program...

RUN SUCCESSFUL (total time: 2m 5s)

- 12 Results list produced at the end of an event
- 13 Log file contents