CS22510 Assignment 1
Runners and Riders
"Out and About"

Chris Savill
chs17@aber.ac.uk

March 19, 2013

Contents

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.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: creator.h
 * Description: Header file for the starter function declarations.
 * First Created: 11/03/2013
 * Last Modified: 14/03/2013
 */

#ifndef CREATOR_H
#define CREATOR_H

#include <memory>
#include "event.h"

bool get_acceptance(); //Function to get the user's input for accepting or rejecting their inputs.
bool checkCourseExists(char letter, Event *event); //Member function that checks if the letter given be the user
    matches any of the course letters.
void ecp_menu(Event *event); //Function that launches the event creation program menu.

#endif  /* CREATOR_H */
```

Listing 2: Main method and menu file.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: competitor.cpp
 * Description: cpp file that contains function definitions for the start-up of the event creation program.
 * First Created: 11/03/2013
 * Last Modified: 14/03/2013
 */

#include "creator.h"
#include <iostream>
#include <cstdlib>
```

```cpp
#include <limits>

using namespace std;

/* Main function that just calls a function that takes over. */
int main(int argc, char** argv) {
    Event *event = new Event();
    ecp_menu(event);

    return 0;
}

/* Function to get the user's input for accepting or rejecting their inputs. */
bool get_acceptance() {
    char option;

    do {
        cout << "If yes press 'y' then 'Enter'" << endl << "If no press 'n' then 'Enter'" << endl;
        cin.clear();
        option = cin.get();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');

        if (option == 'y') return true;
        else if (option == 'n') return false;
        else cout << "Invalid option selected" << endl;
    } while (option != 'y' && option != 'n');
}

/* Function that displays the main menu for the event creation program. */
void ecp_menu(Event *event) {
    int option; //Field to store the user's option input.

    do {
        cout << "*********************************************************" << endl;
        cout << "*  Runners and Riders Event Creation Program Main Menu  *" << endl;
        cout << "*********************************************************" << endl;
        cout << "*              1. Add Competitor to Event               *" << endl;
        cout << "*              2. Add Course to Event                   *" << endl;
        cout << "*              3. Export Event to File                  *" << endl;
        cout << "*              4. Export Competitors to File            *" << endl;
```

```
        cout << "*              5. Export Courses to File                *" << endl;
        cout << "*              6. Exit Event Creation Program           *" << endl;
        cout << "*********************************************************" << endl << endl;

        cout << "Please enter in an option from the above an press 'Enter': ";
        cin.clear();
        cin >> option;
        cin.ignore();

        switch (option) {
            case 1:
                event->add_competitor();
                break;
            case 2:
                event->add_course();
                break;
            case 3:
                event->export_event();
                break;
            case 4:
                event->export_competitors();
                break;
            case 5:
                event->export_courses();
                break;
            case 6:
                delete(event);
                cout << "Exiting program..." << endl << endl;
                break;
            default:
                cout << "Please enter in a valid option." << endl << endl;
        }
    } while (option != 6);
}
```

Listing 3: Header file Event class.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: event.h
```

```
 * Description: Header file for the Event class.
 * First Created: 11/03/2013
 * Last Modified: 14/03/2013
 */

#ifndef EVENT_H
#define EVENT_H

#include <memory>
#include "competitor.h"
#include "course.h"
#include <vector>
#include <cstdlib>
#include <iostream>

#define MAX_EVENT_NAME_LENGTH 79
#define MAX_DATE_LENGTH 19

class Competitor;
class Course;

class Event {
private:
    std::string name; //Name of the event.
    std::string date; //Date of the event.
    std::string start_time; //Start time of the event.
    std::vector<Competitor*> *competitors; //Array of competitors to take part in the event.
    std::vector<Course*> *courses; //Array of courses that are part of an event.

    void set_name(); //Member function to get the user to input the events name.
    void set_date(); //Member function to get the user to input the date of the event.
    void set_start_time(); //Member function to get the user to input the start time of the event.

public:
    Event();
    ~Event();
    std::vector<Course*>* getCourses(); //Member function that returns a pointer to the vector of courses.
    void add_competitor(); //Member function that will handle adding a competitor to the event.
    void add_course(); //Member function that will handle adding a course to the event.
    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 {
```

```
        do {
            cout << "Please enter in the name for the event (no more than 79 characters): ";
            cin.clear();
            getline(cin, name);
        } while (name.length() > MAX_EVENT_NAME_LENGTH);

        cout << endl << endl << "Are you happy with the name: '" << name << "'?" << endl;
        name_chosen = get_acceptance();
    } while (name_chosen == false);

    this->name = name;
}

/* Member function to get the user to input the date of the event. */
void Event::set_date() {
    bool date_chosen = false;
    string date;

    do {
        do {
            cout << endl << endl << "Please enter in the date for the event (no more than 19 characters): ";
            cin.clear();
            getline(cin, date);
        } while (date.length() > MAX_DATE_LENGTH);

        cout << endl << endl << "Are you happy with the date: '" << date << "'?" << endl;
        date_chosen = get_acceptance();
    } while (date_chosen == false);

    this->date = date;
}

/* Member function to get the user to input the start time of the event. */
void Event::set_start_time() {
    bool start_time_chosen = false;
    bool valid_hours = false;
    bool valid_minutes = false;
    char input[3];
    int hours;
    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 are valid.
103                     cout << "Valid minutes entered." << endl << endl;
104                     valid_minutes = true;
105                 }
106             } else cout << "Invalid minutes entered, please enter in a value between 00 and 59 inclusive." << endl
                    << endl;
```

```
107         } while (valid_minutes == false);
108
109         cout << endl << endl << "Are you happy with the start time: '" << hours << ":" << minutes << "'?" << endl;
110         start_time_chosen = get_acceptance();
111     } while (start_time_chosen == false);
112
113     ostringstream string_retriever; //Converts ints into strings.
114     string_retriever << hours;
115     string_hours = string_retriever.str();
116     string_retriever.str(""); //Clears the string stream.
117     string_retriever << minutes;
118     string_minutes = string_retriever.str();
119
120     start_time = string_hours + ":" + string_minutes; //Concatenates the final time into HH:MM format.
121     this->start_time = start_time;
122 }
123
124 /* Member function that will handle adding a competitor to the event.
125  * @param number The current competitor number.
126  */
127 void Event::add_competitor() {
128     if (courses->empty()) cout << "No courses exist for competitor course selection. Please create a course first."
        << endl << endl;
129     else {
130         Competitor *competitor = new Competitor((competitors->size() + 1), this);
131         competitors->push_back(competitor);
132         cout << "New competitor added to event." << endl << endl;
133         cout << "Competitor number: " << competitors->back()->get_number();
134         cout << "Competitor name: " << competitors->back()->get_name() << endl;
135         cout << "Course: " << competitors->back()->get_course() << endl;
136     }
137 }
138
139 /* Member function that will handle adding a course to the event. */
140 void Event::add_course() {
141     Course *course = new Course(this);
142     courses->push_back(course);
143     cout << "New course added to event." << endl << endl;
144     cout << "Course letter: " << courses->back()->get_letter() << endl;
145     cout << "Number of course nodes: " << courses->back()->get_number_of_nodes() << endl;
```

```
146     cout << "Nodes: " << courses->back()->get_node(0);
147 
148     for (int counter = 1; counter < courses->back()->get_number_of_nodes(); counter++) {
149         cout << ", " << courses->back()->get_node(counter);
150     }
151 
152     cout << endl << endl;
153 }
154 
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() {
```

```
    delete(competitors);
    delete(courses);
}
```

Listing 5: Header file for Course class.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: course.h
 * Description: Header file for the Course class.
 * First Created: 11/03/2013
 * Last Modified: 14/03/2013
 */

#ifndef COURSE_H
#define COURSE_H

#include <memory>
#include <vector>

class Event;

class Course {
private:
    char letter; //The courses unique identification letter for an event.
    int number_of_nodes; //The number of nodes the course contains.
    std::vector<int> *nodes; //An array of nodes that are contained in the course.
    std::vector<int> *nodes_available; //An array of nodes that are available to select from, read in from the 'nodes.txt' file.

    void set_letter(Event *event); //Member function that will set the letter of the course.
    void set_number_of_nodes(); //Member function that will set the number of nodes of the course.
    bool read_nodes_available(); //Member function that reads in the nodes from the 'nodes.txt' file and adds them to the nodes available array.
    void add_node(); //Member function that adds a new node to the course.
    bool duplicated_last_node(int number); //Member function to check if the new node being selected matches the last node added.
    bool check_node_exists(int number); //Member function that checks that the node being added exists in the array of nodes available.

```

```
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. */
```

```
int Course::get_node(int index) {
    return this->nodes->at(index);
}

/* Member function that checks if the letter given be the user matches any of the course letters. */
bool checkCourseExists(char letter, Event *event) {
    for (int counter = 0; counter < event->getCourses()->size(); counter++) {
        if (letter == event->getCourses()->at(counter)->get_letter()) return true; //Checks if letter matches any of
            the course letters.
    }

    return false; //Return false if no match found.
}

/* Member function that will set the letter of the course. */
void Course::set_letter(Event *event) {
    bool valid_letter = false;
    bool letter_chosen = false;
    char letter;

    do {
        do {
            cout << endl << endl << "Please enter in the course letter for the course: ";
            cin.clear();
            letter = cin.get();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');

            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.
            else {
                cout << "Please enter in a valid course letter that does not already exist in this event, a-z or A-Z
                    ." << endl << endl;
                valid_letter = false;
            }
        } while (valid_letter == false);

        cout << endl << "Are you happy with the course letter: '" << letter << "'?" << endl;
        letter_chosen = get_acceptance();
    } while (letter_chosen == false);

```

```
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 reached.
97              stringstream int_retriever(input); //Retrieves int from the string stream.
98              int_retriever >> node_number; //Stores the int in node_number.
99              this->nodes_available->push_back(node_number);
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++) cout << ", " << nodes_available->at(
```

```
            counter);
106         cout << endl << endl;
107     } else cout << "File 'nodes.txt' could not be opened. Please check file is in correct directory and permissions.
        " << endl;
108 }
109
110 /* Member function that adds a new node to the course. */
111 void Course::add_node() {
112     bool number_chosen = false;
113     string input;
114     int number = 0;
115
116     do {
117         do {
118             cout << "Please enter in the node number you wish to add to the course: ";
119             getline(cin, input);
120             stringstream int_retriever(input);
121             int_retriever >> number;
122         } while (duplicated_last_node(number) || !check_node_exists(number)); //Makes sure that the number entered
            doesn't match the last number entered and that it does exist.
123
124         cout << endl << endl << "Are you happy with the node number: '" << number << "'?" << endl;
125         number_chosen = get_acceptance();
126     } while (number_chosen == false);
127
128     this->nodes->push_back(number);
129 }
130
131 /* Member function to check if the new node being selected matches the last node added. */
132 bool Course::duplicated_last_node(int number) {
133     if (!nodes->empty()) { //Only checks if there are nodes present.
134         if (number == nodes->back()) {
135             cout << "Node matches last node. Please choose a different node number to add." << endl;
136             return true;
137         }
138     }
139
140     return false; //Returns false if the number entered and the last number entered don't match.
141 }
142
```

```
/* Member function that checks that the node being added exists in the array of node available. */
bool Course::check_node_exists(int number) {
    for (int counter = 0; counter < this->nodes_available->size(); counter++) {
        if (number == this->nodes_available->at(counter)) return true;
    }

    cout << "Node does not exist, please choose a different node number to add." << endl;
    return false; //Returns false if the number entered does not exist in the vector of nodes available.
}

/* Constructor for Course class. */
Course::Course(Event *event) {
    this->nodes = new vector<int>();
    this->nodes_available = new vector<int>();

    if (read_nodes_available()) {
        set_letter(event);
        set_number_of_nodes();

        for (int counter = 0; counter < number_of_nodes - 1; counter++) {
            add_node();
        }

        nodes->push_back(nodes->front()); //Adds the last node, matching the first node to the course.
    } else cout << "Nodes could not be read in from 'nodes.txt' file. Course creation cancelled." << endl << endl;
}

/* Destructor for Course class. */
Course::~Course() {
    delete(nodes);
    delete(nodes_available);
}
```


Listing 7: Header file for Competitor class.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: competitor.h
 * Description: Header file for the Competitor class.
 * First Created: 11/03/2013
```

```
 * Last Modified: 14/03/2013
 */

#ifndef COMPETITOR_H
#define COMPETITOR_H

#include <memory>
#include <string>

#define MAX_COMPETITOR_NAME_LENGTH 51 //Includes null terminator \0.

class Event;

class Competitor {
private:
    int number; //The competitor's unique identification number for an event.
    std::string name; //The competitor's name.
    char course; //The course letter the competitor is entering in for.

    void set_number(int number); //Member function that will set the number of the competitor.
    void set_name(); //Member function that will set the name of the competitor.
    void set_course(Event *event); //Member function that will set the course letter for the competitor.

public:
    int get_number(); //Member function to return a competitor's number.
    std::string get_name(); //Member function to return a competitor's name.
    char get_course(); //Member function to return a competitor's course.
    Competitor(int number, Event *event);
};

#endif  /* COMPETITOR_H */
```

Listing 8: Cpp file for Competitor class.

```
/*
 * Author: Chris Savill, chs17@aber.ac.uk
 * File Name: competitor.cpp
 * Description: cpp file that contains member function definitions for the competitor class.
 * First Created: 11/03/2013
 * Last Modified: 14/03/2013
```

```
 */

#include "competitor.h"
#include "creator.h"
#include <ctype.h>
#include <iostream>
#include <limits>

using namespace std;

/* Member function to return a competitor's number. */
int Competitor::get_number() {
    return this->number;
}

/* Member function to return a competitor's name. */
string Competitor::get_name() {
    return this->name;
}

/* Member function to return a competitor's course. */
char Competitor::get_course() {
    return this->course;
}

/* Member function that will set the number of the competitor.
 * @param number The number for the competitor.
 */
void Competitor::set_number(int number) {
    this->number = number;
}

/* Member function that will set the name of the competitor. */
void Competitor::set_name() {
    bool name_chosen = false;
    string name;

    do {
        do {
            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 }
```

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.");
```

```
                TypeWindow typeWindow = new TypeWindow(event);
            } else {
                System.out.print("Exiting Program...\n");
            }
        }
    }
}
```

Listing 10: Event class.

```
/* File Name: Manager.java
 * Description: Event class which stores all members and functions pertaining to an event.
 * First Created: 15/03/2013
 * Last Modified: 18/03/2013
 */
package Data_Structures;

import File_Handling.FileHandler;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Date;

/**
 * @author Chris Savill, chs17@aber.ac.uk
 */
public class Event {

    private ArrayList<Competitor> competitors; //Array list of competitors in an event.
    private ArrayList<Node> nodes; //Array list of nodes in an event.
    private ArrayList<Node> checkpoints; //Array list of nodes that are of type "CP" or "MC".
    private ArrayList<Course> courses; //Array list of courses in an event.
    private ArrayList<Record> records; //Array list of records logged.
    private int lastLineRead;
    private Date lastRecordedTime;
    private boolean timeFileExists;
    private String[] fileNames;

    /**
     * Method to return array list of competitors.
     *
```

```
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 the
```

```
111         * program.
112         *
113         * @param args The list of filenames to load the required data into the
114         * system.
115         * @return Successful/Unsuccessful.
116         */
117        public boolean loadCycle(String[] fileNames) throws IOException {
118            this.fileNames = fileNames;
119
120            FileHandler fileReader = new FileHandler();
121
122            if (fileReader.readNodes(fileNames[0], this)) {
123                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 Exiting.\n");
128                    }
129                } else {
130                    System.out.print("Failed to load courses. Program Exiting.\n");
131                }
132            } else {
133                System.out.print("Failed to load nodes. Program Exiting.\n");
134            }
135
136            return false;
137        }
138
139        /**
140         * Method that checks if the node number passed in exists in the array list
141         * of nodes loaded in.
142         *
143         * @param number The number to be compared with.
144         * @return True if node exists else false.
145         */
146        public boolean checkNodeExists(int number) {
147            for (int counter = 0; counter < nodes.size(); counter++) {
148                if (number == nodes.get(counter).getNumber()) {
149                    return true;
150                } //Nodes exists.
```

```
151             }
152
153             return false; //Returns false if the node number passed in does not exist in the array list of nodes.
154         }
155
156         /**
157          * Method that checks if the course letter passed in exists in the array
158          * list of courses loaded in.
159          *
160          * @param letter The letter to be compared with.
161          * @return True if course exists else false.
162          */
163         public boolean checkCourseExists(char letter) {
164             for (int counter = 0; counter < courses.size(); counter++) {
165                 if (letter == courses.get(counter).getLetter()) {
166                     return true;
167                 } //Course exists.
168             }
169
170             return false; //Returns false if the course letter passed in does not exist in the array list of courses.
171         }
172
173         /**
174          * Method to let the know event instance know that a time file does now
175          * exist.
176          *
177          */
178         public void setTimesFilesExistsTrue() {
179             timeFileExists = true;
180         }
181
182         /**
183          * Method to find a competitor and return it.
184          *
185          * @param competitorNumber The number of the competitor being looked for.
186          * @return The competitor matched.
187          */
188         public Competitor retrieveCompetitor(int competitorNumber) {
189             for (Competitor competitor : competitors) {
190                 if (competitor.getNumber() == competitorNumber) {
```

```
                return competitor;
            }
        }
        return null;
    }

    /**
     * Method to find a course and return it.
     *
     * @param courseLetter The course being looked for.
     * @return The course matched.
     */
    public Course retrieveCourse(char courseLetter) {
        for (Course course : courses) {
            if (course.getLetter() == courseLetter) {
                return course;
            }
        }
        return null;
    }

    /**
     * Method to retrieve the checkpoint number.
     *
     * @param type The type of the checkpoint.
     * @param listIndex The index of the list element.
     * @param numberOfElements The size of the list.
     * @return The checkpoint number being looked for.
     */
    public int retrieveCheckpointNumber(String type, int listIndex, int numberOfElements) {
        int[] checkpointArray = new int[numberOfElements];
        int arrayIndex = 0;

        for (int counter = 0; counter < checkpoints.size(); counter++) {
            if (checkpoints.get(counter).getType().equals(type)) {
                checkpointArray[arrayIndex++] = checkpoints.get(counter).getNumber();
            }
        }

        return checkpointArray[listIndex];
```

```
231     }
232
233     /**
234      * Method to check if the new record is valid.
235      *
236      * @param checkpoint The checkpoint number.
237      * @param status The status.
238      * @param competitorNumber The competitor's number.
239      * @param time The time of the record.
240      * @return True is record is valid, else false.
241      */
242     public boolean checkNewRecord(int checkpoint, int status, int competitorNumber, Date time) {
243         Competitor competitor = retrieveCompetitor(competitorNumber);
244
245         if (timeFileExists != false) {
246             if (time.before(lastRecordedTime)) {
247                 System.out.println("\nInvalid time.");
248                 return false;
249             }
250         }
251
252         if (competitor.getStatus() == 'I' || competitor.getStatus() == 'E') {
253             System.out.println("\nCompetitor already excluded.");
254             return false; //Should not be updated as competitor already excluded.
255         } else if (status == 2 || status == 3) {
256             if (competitor.getStatus() != 'A') {
257                 System.out.println("\nCompetitor hasn't arrived at a medical checkpoint yet.");
258                 return false; //Competitor cannot be departing or be exclude from a medical checkpoint they haven't
                        arrived at.
259             } else {
260                 return true;
261             }
262         } else if (status == 0) {
263             if (competitor.getStatus() != 'A') {
264                 return true;
265             } else {
266                 System.out.println("\nCompetitor is still being examined at a medical checkpoint.");
267                 return false; //Competitor cannot be at a time checkpoint when should be at a medical checkpoint
                        being examined.
268             }
```

```
269             } else if (status == 1) {
270                 return true;
271             }
272 
273             return false;
274         }
275 
276         /**
277          * Method to determine the final status to be written to the time record file.
278          * @param checkpoint The checkpoint number.
279          * @param status The status.
280          * @param competitorNumber The competitor's number.
281          * @return The final status for the record.
282          */
283         public char determineFinalStatus(int checkpoint, int status, int competitorNumber) {
284             Competitor competitor = retrieveCompetitor(competitorNumber);
285 
286             if (competitor.getStatus() == 'N') {
287                 if (checkpoint != competitor.getCheckpoints()[competitor.getCheckpointIndex()]) {
288                     return 'I';
289                 } else if (status == 0) {
290                     return 'T';
291                 } else if (status == 1) {
292                     return 'A';
293                 }
294             } 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';
```

```
            } else if (status == 3) {
                return 'E';
            }
        }

        System.out.print("\n\nInvalid final status, returning 'I'.\n");
        return 'I';
    }

    /**
     * Constructor to initialise the event.
     */
    public Event(String[] fileNames) {
        competitors = new ArrayList<Competitor>();
        nodes = new ArrayList<Node>();
        checkpoints = new ArrayList<Node>();
        courses = new ArrayList<Course>();
        records = new ArrayList<Record>();
        lastLineRead = 0;
        timeFileExists = false;
    }
}
```


Listing 11: Node class.

```
/* File Name: Node.java
 * Description: Node class which stores all members and functions pertaining to a node.
 * First Created: 15/03/2013
 * Last Modified: 15/03/2013
 */
package Data_Structures;

/**
 * @author Chris Savill, chs17@aber.ac.uk
 */
public class Node {

    private int number;
    private String type;

```

```
    /**
     * Constructor to initialise Node.
     *
     * @param number The number of the node.
     * @param type The type of the node.
     */
    public Node(int number, String type) {
        this.number = number;
        this.type = type;
    }

    /**
     * Method to return the node's number.
     *
     * @return The node number.
     */
    public int getNumber() {
        return number;
    }

    /**
     * Method to return the node's type.
     * @return The type of the node.
     */
    public String getType() {
        return type;
    }
}
```

Listing 12: Course class.

```
/* File Name: Couse.java
 * Description: Course class which stores all members and functions pertaining to a course.
 * First Created: 15/03/2013
 * Last Modified: 17/03/2013
 */
package Data_Structures;

/**
 * @author Chris Savill, chs17@aber.ac.uk
```

```
 */
public class Course {

    private char letter;
    private int numberOfNodes;
    private int[] nodes;

    /**
     * Constructor to initialise course.
     *
     * @param letter The course letter identifier.
     * @param numberOfNodes The number of nodes the course contains.
     * @param nodes The array of nodes the course contains.
     */
    public Course(char letter, int numberOfNodes, int[] nodes) {
        this.letter = letter;
        this.numberOfNodes = numberOfNodes;
        this.nodes = nodes;
    }

    /**
     * Method to return the course letter.
     */
    public char getLetter() {
        return letter;
    }

    /**
     * Method to return the number of nodes the course contains.
     */
    public int getNumberOfNodes() {
        return numberOfNodes;
    }

    /**
     * Method to return the array of nodes the course contains.
     */
    public int[] getNodes() {
        return nodes;
    }
```

```
}
```

Listing 13: Competitor class.

```
/* File Name: Competitor.java
 * Description: Competitor class which stores all members and functions pertaining to a competitor.
 * First Created: 15/03/2013
 * Last Modified: 18/03/2013
 */
package Data_Structures;

import java.util.ArrayList;

/**
 * @author Chris Savill, chs17@aber.ac.uk
 */
public class Competitor {

    private String name;
    private int number;
    private char course;
    private char status;
    private int[] checkpoints;
    private int checkpointIndex;

    /**
     * Constructor to initialise competitor.
     *
     * @param number The competitor's number.
     * @param course The competitor's course.
     * @param name The competitor's name.
     */
    public Competitor(int number, char course, String name, Event event) {
        this.number = number;
        this.course = course;
        this.name = name;
        this.checkpoints = setCheckpoints(event);
        this.checkpointIndex = 0;
        this.status = 'N'; //Not started yet.
    }
```

```
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 arrived
76       * at.
```

```
77          *
78          * @return The index of the last checkpoint the competitor arrived at.
79          */
80         public int getCheckpointIndex() {
81             return checkpointIndex;
82         }
83 
84         /**
85          * Method to return the int array of checkpoints.
86          *
87          * @return The int array of checkpoints.
88          */
89         public int[] getCheckpoints() {
90             return checkpoints;
91         }
92 
93         /**
94          * Method to get the nodes which are recordable checkpoints (non-junction
95          * nodes).
96          *
97          * @param event The event instance.
98          * @return The int array of checkpoints.
99          */
100        private int[] setCheckpoints(Event event) {
101            ArrayList<Integer> checkpointsList = new ArrayList<Integer>();
102            Course courseReference = event.retrieveCourse(course);
103
104            for (int counter = 0; counter < courseReference.getNumberOfNodes(); counter++) {
105                for (int counter2 = 0; counter2 < event.getNodes().size(); counter2++) {
106                    if ((!event.getNodes().get(counter2).getType().equals("JN"))
107                            && (event.getNodes().get(counter2).getNumber() == courseReference.getNodes()[counter])) {
108                        checkpointsList.add(event.getNodes().get(counter2).getNumber());
109                        break;
110                    }
111                }
112            }
113
114            int[] intList = new int[checkpointsList.size()];
115
116            for (int counter = 0; counter < checkpointsList.size(); counter++) {
```

```
117             intList[counter] = checkpointsList.get(counter).intValue();
118         }
119
120         return intList;
121     }
122
123     /**
124      * Method to set the status of the competitor.
125      *
126      * @param status The current status of the competitor.
127      */
128     public void setStatus(char status) {
129         this.status = status;
130     }
131
132     /**
133      * Method to increment the checkpoint index by 1.
134      */
135     public void incrementCheckpointIndex() {
136         checkpointIndex++;
137     }
138 }
```

Listing 14: Record class.

```
1  /* File Name: Record.java
2   * Description: Record class which stores all members and functions pertaining to checking a competitor in at a
       checkpoint.
3   * First Created: 15/03/2013
4   * Last Modified: 17/03/2013
5   */
6  package Data_Structures;
7
8  import java.util.Date;
9
10 /**
11  * @author Chris Savill, chs17@aber.ac.uk
12  */
13 public class Record {
14
```

```
15      private Event event;
16      private char competitorStatus;
17      private int checkpoint;
18      private int competitorNumber;
19      private Date time;
20
21      /**
22       * Constructor to initialise record data when read in from file.
23       *
24       * @param checkpoint The number of the checkpoint.
25       * @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(fileName));
104
105             while ((input = reader.readLine()) != null) {
106                 if (input.matches(pattern)) { //Checks to make sure the line is in the right format.
107                     subStrings = input.split("\\s+"); //Gets rid of whitespace and separates the strings into
                            substrings.
108                     courseLetter = subStrings[0].charAt(0); //Retrieves the course letter.
109                     numberOfNodes = Integer.parseInt(subStrings[1]);
110                     nodes = new int[numberOfNodes];
111
112                     for (int counter = 0; counter < numberOfNodes; counter++) {
113                         if (event.checkNodeExists(Integer.parseInt(subStrings[counter + 2]))) {
114                             nodes[counter] = Integer.parseInt(subStrings[counter + 2]);
115                         } else {
116                             System.out.print("Invalid node in course file found. Cancelling loading of courses\n\n")
                                ;
117                             reader.close();
118                             return false;
119                         }
120                     }
121
```

```
122                     Course course = new Course(courseLetter, numberOfNodes, nodes); //Creates new course with
                            parameters read in.
123                     event.getCourses().add(course); //Adds new course to array list of courses.
124                 } else {
125                     System.out.print("Invalid line format. Cancelling loading of courses\n\n");
126                     reader.close();
127                     return false;
128                 }
129             }
130 
131             if (!event.getCourses().isEmpty()) {
132                 System.out.print("Loading in of courses successful.\n\n");
133                 reader.close();
134                 return true;
135             } else {
136                 System.out.print("Loading in of courses unsuccessful. No courses in file.\n\n");
137                 reader.close();
138                 return false;
139             }
140         } catch (FileNotFoundException ex) {
141             Logger.getLogger(FileHandler.class.getName()).log(Level.SEVERE, null, ex);
142         }
143 
144         System.out.print("Could not open file that contains courses.\n\n");
145         return false;
146     }
147 
148     /**
149      * Method to read in all the details for the competitors pertaining to an
150      * 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 any
155      * point.
156      */
157     public boolean readCompetitors(String fileName, Event event) throws IOException {
158         String input;
159         int competitorNumber;
160         char courseLetter;
```

```
161         String[] subStrings;
162         String competitorName;
163         String pattern = "(\\d+\\s+[A-Za-z]((\\s+[A-Za-z]{1}[a-z]+)+)$)"; //Regular expression for competitors file.
164 
165         try {
166             BufferedReader reader = new BufferedReader(new FileReader(fileName));
167 
168             while ((input = reader.readLine()) != null) {
169                 if (input.matches(pattern)) { //Checks to make sure the line is in the right format.
170                     subStrings = input.split("\\s+"); //Gets rid of whitespace and separates the strings into
                            substrings.
171                     competitorNumber = Integer.parseInt(subStrings[0]); //Retrieves the competitor number by parsing
                             the string into an int.
172 
173                     if (event.checkCourseExists(subStrings[1].charAt(0))) {
174                         courseLetter = subStrings[1].charAt(0); //Retrieves the course the competitor is entering in
                                on.
175                     } else {
176                         System.out.print("Invalid course in competitor file found. Cancelling loading of competitors
                            .\n\n");
177                         reader.close();
178                         return false;
179                     }
180 
181                     competitorName = subStrings[2];
182 
183                     if (subStrings.length > 3) {
184                         for (int counter = 3; counter < subStrings.length; counter++) {
185                             competitorName += " " + subStrings[counter]; //Concatanates name substrings together.
186                         }
187                     }
188 
189                     Competitor competitor = new Competitor(competitorNumber, courseLetter, competitorName, event);
                            //Creates new competitor with parameters read in.
190                     event.getCompetitors().add(competitor); //Adds new competitor to array list of competitors.
191                 } else {
192                     System.out.print("Invalid line format. Cancelling loading of competitors.\n\n");
193                     reader.close();
194                     return false;
195                 }
```

```
196             }
197
198             if (!event.getCompetitors().isEmpty()) {
199                 System.out.print("Loading in of competitors successful.\n\n");
200                 reader.close();
201                 return true;
202             } else {
203                 System.out.print("Loading in of competitors unsuccessful. No competitors in file.\n\n");
204                 reader.close();
205                 return false;
206             }
207         } catch (FileNotFoundException ex) {
208             Logger.getLogger(FileHandler.class
209                     .getName()).log(Level.SEVERE, null, ex);
210         }
211
212         System.out.print("Could not open file that contains competitors.\n\n");
213         return false;
214     }
215
216     /**
217      * Method to read in all the details for the checkpoint times pertaining to
218      * an event.
219      *
220      * @param fileName The file name required to access the file needed.
221      * @param event The event instance.
222      * @return True if file loaded successfully, else false if it fails at any
223      * point.
224      */
225     public boolean readTimes(String fileName, Event event) throws IOException, ParseException {
226         String input;
227         int currentLineNumber = 0;
228         int lastLineNumber = event.getLastLineRead();
229         char competitorStatus;
230         int competitorNumber;
231         int nodeNumber;
232         String[] subStrings;
233         String pattern = "([A-Z{1}]((\\s+\\d+){2})\\s+[0-2{1}][0-9{1}]:[0-5{1}][0-9{1}]$)"; //Regular expression for
                times file.
234         SimpleDateFormat formatter = new SimpleDateFormat("HH:mm");
```

```
235             Date time;
236
237             event.getRecords().clear(); //Empties array list.
238
239             try {
240                 FileChannel channel = new RandomAccessFile(fileName, "rw").getChannel(); //Creates a channel for the
                        file.
241                 FileLock lock = channel.lock(); //Blocks/Halts thread until lock aquired.
242
243                 BufferedReader reader = new BufferedReader(new FileReader("cp_times.txt"));
244
245                 while ((input = reader.readLine()) != null) {
246                     currentLineNumber++;
247                     if (currentLineNumber > lastLineNumber) {
248                         if (input.matches(pattern)) { //Checks to make sure the 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 24hour HH:MM.
254
255                             Competitor competitor = event.retrieveCompetitor(competitorNumber);
256                             if (competitor.getStatus() == 'T') {
257                                 competitor.incrementCheckpointIndex(); //Increments the competitor's checkpoint intdex
                                        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");
```

```
                        reader.close();
                        lock.release();
                        channel.close();
                        return false;
                    }
                }
            }

            event.setTimesFilesExistsTrue(); //Lets the event instance know that an event does exist.
            reader.close(); //Closes reader.
            lock.release(); //Releases file lock.
            channel.close(); //Closes channel ensuring lock release and release of resources.
            return true;
        } catch (FileNotFoundException ex) {
            System.out.print("Could not open file that contains times.\n\n");
        }
        return false;
    }

    /**
     * Method to write a record on a line in the time records file.
     *
     * @param fileName The file name required to access the file needed.
     * @param record The record to be written.
     * @return True if file written to successfully, else false if it fails at
     * any point.
     */
    public boolean appendTimeRecord(String fileName, Record record) {
        SimpleDateFormat formatter = new SimpleDateFormat("HH:mm");

        try {
            FileChannel channel = new RandomAccessFile(fileName, "rw").getChannel(); //Creates a channel for the
                file.
            FileLock lock = channel.lock();

            FileWriter writer = new FileWriter(fileName, true); //True sets append mode.
            writer.write(record.getCompetitorStatus() + " " + record.getCheckpointNumber()
                    + " " + record.getCompetitorNumber() + " " + formatter.format(record.getTime()) + "\n");
            writer.close();
            lock.release();
```

```
            channel.close();
            return true;
        } catch (IOException ex) {
            System.out.print("\nCould not open file for writing.\n\n");
        }
        return false;
    }
}
```

Listing 16: TypeWindow class.

```
/* File Name: TypeWindow.java
 * Description: TypeWindow GUI class using swing.
 * First Created: 17/03/2013
 * Last Modified: 18/03/2013
 */
package GUI;

import Data_Structures.Event;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.border.EmptyBorder;

/**
 * @author Chris Savill, chs17@aber.ac.uk
 */
public class TypeWindow extends JFrame implements ActionListener {

    private Event event;
    private boolean medicalSelected;
    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);
```

```
                    SelectionWindow selectionWindow = new SelectionWindow(event, "MC", typeFrame);
                } else {
                    typeFrame.setVisible(false);
                    SelectionWindow selectionWindow = new SelectionWindow(event, "CP", typeFrame);
                }

                typeFrame.dispose();
                this.dispose();
                break;
            case "time":
                medicalSelected = false;
                break;
            case "medical":
                medicalSelected = true;
                break;
        }
    }
}
```

Listing 17: SelectionWindow class.


```
/* File Name: SelectionWindow.java
 * Description: SelectionWindow GUI class using swing.
 * First Created: 16/03/2013
 * Last Modified: 17/03/2013
 */
package GUI;

import Data_Structures.Competitor;
import Data_Structures.Event;
import Data_Structures.Node;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.DefaultListModel;
import javax.swing.ImageIcon;
import javax.swing.JButton;
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 EASTcontainer
                .
77          bottomPanel = new JPanel();
78          selectionFrame.add(bottomPanel, BorderLayout.SOUTH); //Adds panel to frame and places it in SOUTH container.
79          ////////////////////////////////////////////////////////////////
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(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED); //Adds
                vertical scrollbar to JList
94          checkpointListScrollBar.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED); //
```

```
            Adds horizontal scrollbar to JList
        checkpointPanel.add(checkpointListScrollBar);
        ////////////////////////////////////////////////////////////////

        //Setup competitor panel components:
        competitorLabel = new JLabel("Select Competitor Below: ");
        competitorPanel.add(competitorLabel, BorderLayout.NORTH);

        competitorListModel = new DefaultListModel();
        competitorList = new JList(competitorListModel);
        competitorList.setBorder(new LineBorder(Color.BLACK));
        competitorPanel.add(competitorList, BorderLayout.CENTER);
        competitorList.addListSelectionListener(this);

        competitorListScrollBar = new JScrollPane(competitorList);
        competitorListScrollBar.setPreferredSize(new Dimension(400, 300));
        competitorListScrollBar.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED); //Adds
            vertical scrollbar to JList
        competitorListScrollBar.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED); //
            Adds horizontal scrollbar to JList
        competitorPanel.add(competitorListScrollBar);
        ////////////////////////////////////////////////////////////////

        //Setup bottom panel components:
        next = new JButton("Next");
        next.setPreferredSize(new Dimension(100, 50));
        bottomPanel.add(next);
        next.addActionListener(this);
        ////////////////////////////////////////////////////////////////

        //Finialise frame setup:
        addCheckpoints();
        addCompetitors();
        selectionFrame.pack();
        selectionFrame.setVisible(true); //Makes the frame visible
        ////////////////////////////////////////////////////////////////
    }

    /**
     * Method that adds the checkpoint checkpoints to the checkpoint JList
```

```
132         */
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() + ": " + currentCheckpoint.getType());
139                 }
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: " + currentCompetitor.getNumber()
151                         + "   Course: " + currentCompetitor.getCourse() + "   Name: " + currentCompetitor.getName());
152             }
153         }
154 
155         /**
156          * Method to handle actions performed.
157          * @param evt The event triggered.
158          */
159         @Override
160         public void actionPerformed(ActionEvent evt) {
161             String actionCommand = evt.getActionCommand();
162 
163             if (actionCommand.equals("Next")) {
164                 if (checkpointSelected == true && competitorSelected == true) {
165                     selectionFrame.setVisible(false);
166                     TimeWindow timeWindow = new TimeWindow(event, checkpoint, type, competitor, selectionFrame,
                            typeFrame);
167                     selectionFrame.dispose();
168                     this.dispose();
169                 } else {
170                     JOptionPane.showMessageDialog(selectionFrame, "Please select both a checkpoint and competitor.");
```

```
171                 }
172             }
173         }
174 
175         /**
176          * Method to handle values changing in a JList.
177          * @param evt The event triggered.
178          */
179         @Override
180         public void valueChanged(ListSelectionEvent evt) {
181 
182             if (!evt.getValueIsAdjusting()) {
183                 JList list = (JList) evt.getSource();
184 
185                 if (list.equals(checkpointList)) {
186                     checkpoint = event.retrieveCheckpointNumber(type, list.getSelectedIndex(), list.getModel().getSize()
                        );
187                     checkpointSelected = true;
188                 } else if (list.equals(competitorList)) {
189                     competitor = event.getCompetitors().get(list.getSelectedIndex()).getNumber();
190                     competitorSelected = true;
191                 }
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 GUI.
```

```
53          *
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 competitor, JFrame selectionFrame, JFrame
               typeFrame) {
62             selectionFrame.dispose();
63
64             this.typeFrame = typeFrame;
65             this.event = event;
66             this.checkpoint = checkpoint;
67             this.type = type;
68             this.competitor = competitor;
69             fileHandler = new FileHandler();
70
71             //Setup frame:
72             timeFrame = new JFrame("Time Of Record");
73
74             if (type.equals("MC")) {
75                 status = getMedicalOptions();
76             } else {
77                 status = 0; //Comeptitor status not a medical related status.
78             }
79
80             timeFrame.setLocation(400, 200);
81             timeFrame.setLayout(new BorderLayout());
82             timeFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //Sets the default close operation
83             timeFrame.setIconImage(new ImageIcon("horse.jpg").getImage()); //Loads an image and sets it as the frame
                   icon
84             ////////////////////////////////////////////////////////////
85
86             //Setup panels:
87             timePanel = new JPanel(new BorderLayout()); //Creates new JPanel.
88             timePanel.setBorder(new EmptyBorder(10, 25, 10, 25));  //Sets an invisible border to simulate a padding
                   effect
89             timeFrame.add(timePanel, BorderLayout.WEST); //Adds panel to frame and places it in WEST container.
```

```
90          bottomPanel = new JPanel();
91          timeFrame.add(bottomPanel, BorderLayout.SOUTH); //Adds panel to frame and places it in SOUTH container.
92          ////////////////////////////////////////////////////////////////
93
94          //Setup checkpoint panel components:
95          timeLabel = new JLabel("Select Time Below: ");
96          timePanel.add(timeLabel, BorderLayout.NORTH);
97
98          date = new Date();
99          spinnerModel = new SpinnerDateModel(date, null, null, Calendar.HOUR_OF_DAY);
100         spinner = new JSpinner(spinnerModel);
101         dateEditor = new JSpinner.DateEditor(spinner, "HH:mm"); //24-hour format.
102         spinner.setEditor(dateEditor);
103         timePanel.add(spinner, BorderLayout.CENTER);
104         ////////////////////////////////////////////////////////////////
105
106         //Setup bottom panel components:
107         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                     JOptionPane.showMessageDialog(timeFrame, "Failed to load time records from file.");
132                 }
133             } catch (IOException | ParseException ex) {
134                 Logger.getLogger(TimeWindow.class.getName()).log(Level.SEVERE, null, ex);
135             }
136 
137             if (event.checkNewRecord(checkpoint, status, competitor, (Date) spinner.getValue())) {
138                 char finalStatus = event.determineFinalStatus(checkpoint, status, competitor);
139 
140                 Record record = new Record(checkpoint, finalStatus, competitor, (Date) spinner.getValue());
141                 event.getRecords().add(record);
142 
143                 fileHandler.appendTimeRecord(event.getFileNames()[3], record);
144                 JOptionPane.showMessageDialog(timeFrame, "Time record succesfully added.");
145             } else {
146                 JOptionPane.showMessageDialog(timeFrame, "Non-valid record. Record will not added.");
147             }
148 
149             timeFrame.dispose(); //Closes frame and releases resourses.
150             this.dispose(); //Releases resources.
151             TypeWindow typeFrame = new TypeWindow(event);
152 
153         }
154     }
155 
156     /**
157      * Method to get the user to select the status of the competitor at the
158      * medical checkpoint.
159      *
160      * @return The status of the competitor at the medical checkpoint.
161      */
162     public int getMedicalOptions() {
163         String[] options = new String[]{"Arriving", "Departing", "Excluded"};
164 
165         int selection = JOptionPane.showOptionDialog(timeFrame, "Is the competitor being marked as 'Arriving',"
166                 + " 'Departing' or as 'Excluded' on medical grounds?", "Medical Marking", JOptionPane.DEFAULT_OPTION
                    ,
167                 JOptionPane.PLAIN_MESSAGE, null, options, options[0]);
168 
```

```
169             if (selection == 0) {
170                 return 1; //Competitor status to be set to arriving at medical checkpoint.
171             } else if (selection == 1) {
172                 return 2; //Competitor status to be set to departing medical checkpoint.
173             } else if (selection == 2) {
174                 return 3; //Competitor status to be set to excluded based on medical grounds.
175             }
176 
177             return 0;
178         }
179 }
```