BSc (Hons) Artificial Intelligence and Data Science

Module number: CM1601

Module title: Programming Fundamentals

Module coordinator: Iresh Bandara

Semester: 2

Deadline:- April 10th

Assessment type: Individual coursework

Student ID (IIT): 20211100

Student ID (RGU): 2236741

Student Name: Abdul Najib Zahid Ali

Executive Summary

This report includes the code and explanation of a GUI application for a rally cross championship. A flow chart depicting the flow of the program along with test plans and test cases and junit tests are also included in this report.

Contents Page

Table of Contents

Table of Figures.

1. Flow charts

1.1 Flow chart for add function.

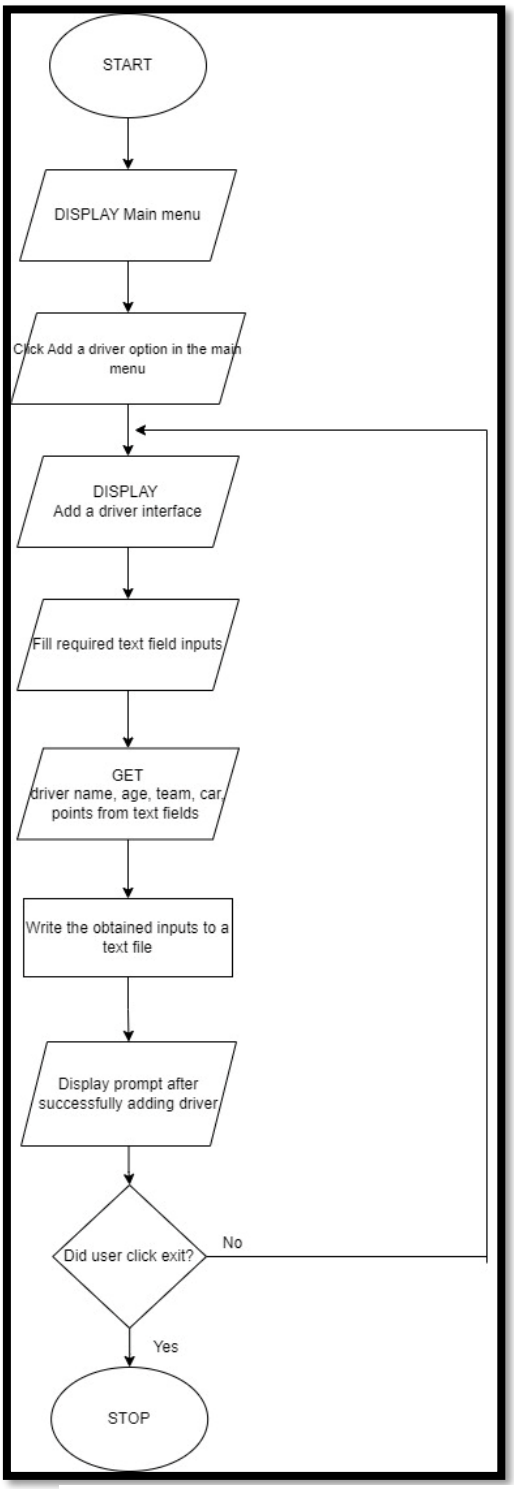

Figure 1: Flow chart for add functions.

1.2 Flow chart for delete function.

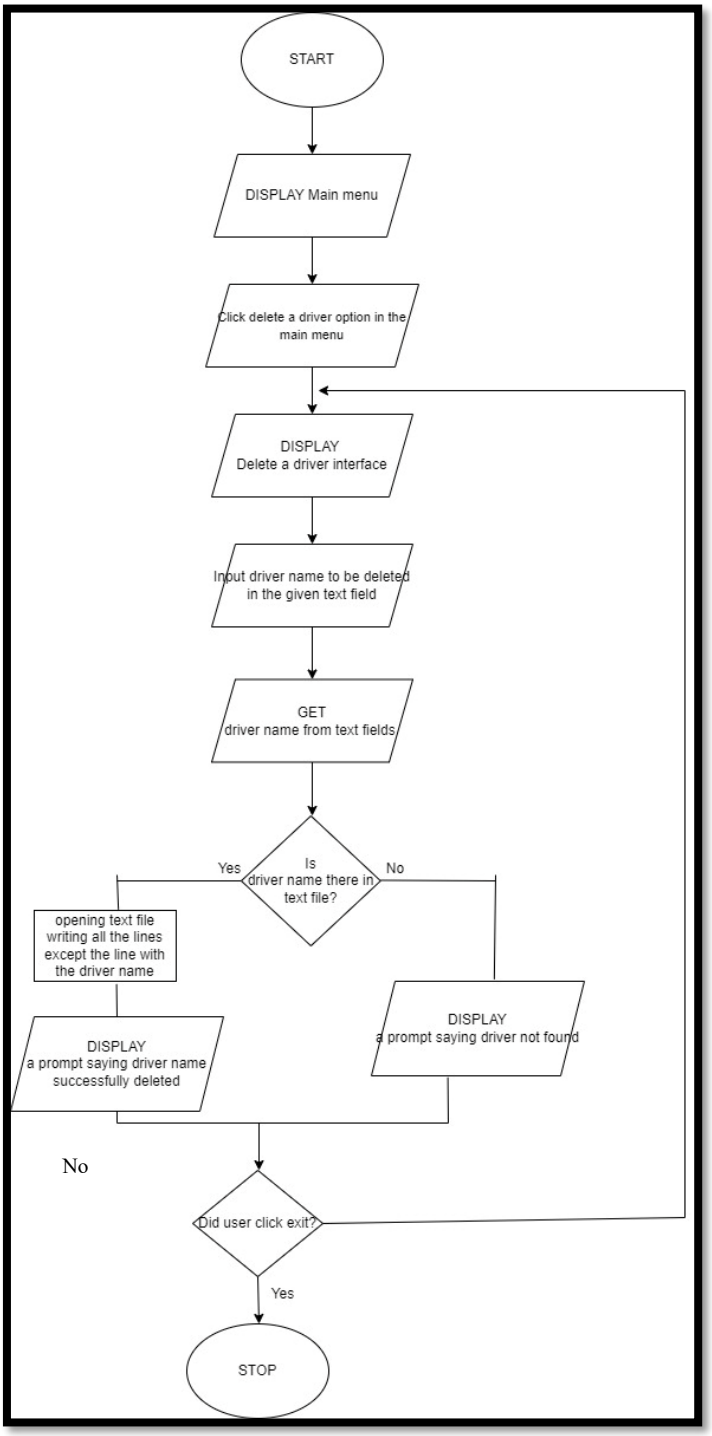

Figure 2: Flow chart for delete function.

2. Introduction to functions with code.

2.1 Add Function

This function provides user the ability to add a driver to the system. Text fields are provided in the GUI interface of this section where the users have to enter the driver details and then the inputs are obtained from the text fields and entered into a text file. If the record adding process is successful user is prompted if not user is prompted with the necessary reason for failure.

Code in functions class

A method is created where it has the drive details as parameters and those parameters are written to a text file when it is called. After that a prompt is displayed to the user.

```
void addDriver(String name, int age, String team, String car, int points){

    try {
        // reading file
        FileWriter driver = new
FileWriter("C:\\Users\\User\\IdeaProjects\\RCC
Management\\driver_details.txt", true);
        BufferedWriter driver_records = new BufferedWriter(driver);
        //writing records into text file
        driver_records.write(name + " " + age + " " + team + " " + car + "
" + points + "\n");
        driver_records.close();
        // prompt shown to user after successfully adding a driver
        Alert driveradded = new Alert(Alert.AlertType.INFORMATION);
        driveradded.setTitle("Add driver record");
        driveradded.setContentText("Driver record has been successfully
added");
        Optional<ButtonType> result = driveradded.showAndWait();
    }
    catch (Exception e) {return;}

}
```

Code in HelloController class

The inputs in text field are obtained by getText() method and validated to check if the age and points field contain integer values. If those contain any other value than integer a prompt is displayed to the user and until user closes and reopens the window user wont be able to re enter the records.

```
public void add_driver_function() {
    String d_name = name.getText();

    // validating age input
```

```
    String drv_age = age.getText();
    int d_age = 0;
    try {
        d_age = Integer.parseInt(drv_age);

    } catch (NumberFormatException e) {
        vlabel.setText("Invalid Entry");

    }

    String d_team = team.getText();
    String d_car = car.getText();

    //validating points input

    String drv_points = points.getText();
    int d_points = 0;
    try {
        d_points = Integer.parseInt(drv_points);

    } catch (NumberFormatException e) {
        vlabel.setText("Invalid Entry");
    }


    // if the label is displaying invalid entry user cant add data until
closing and reopening the window
    if (vlabel.getText().equals("Invalid Entry")){
        Alert wrongvalue = new Alert(Alert.AlertType.INFORMATION);
        wrongvalue.setTitle("Invalid Entry");
        wrongvalue.setHeaderText("Couldn't Add Record !");
        wrongvalue.setContentText("Please Enter only numeric values to Age
or Points"+"\n"+"\n"+"CLOSE AND REOPEN WINDOW BEFORE TRYING AGAIN");
        wrongvalue.showAndWait();
    }
    else
    {
        functions x = new functions();
        x.addDriver(d_name, d_age, d_team, d_car, d_points);
    }
}
```

2.2 Delete Function

This function allows user to delete a driver record by entering the driver's name.

Code in functions class.

This method get driver name as the parameter because that is what is used to find if the driver details exist in the file and then the whole text file is written once again without the line that contains the driver name. Then a prompt is displayed saying driver record successfully deleted.

```
void deleteDriver(String name){

    try {
        // opening the driver details file and reading lines
        File file = new File("C:\\Users\\User\\IdeaProjects\\RCC
Management\\driver_details.txt");
        BufferedReader reader = new BufferedReader(new FileReader(file));
        String line;
        StringBuilder content = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            String[] x = line.split(" ");
            if (!name.equals(x[0])) {
                content.append(line).append("\n");
            }
        }
        reader.close();

        // opening file in write mode and writing the records back except
the record of the user entered driver name
        FileWriter writer = new FileWriter(file);
        writer.write(content.toString());
        writer.close();
        // prompt shown to user after successful deletion
        Alert driverdelete = new Alert(Alert.AlertType.INFORMATION);
        driverdelete.setTitle("Delete drive record");
        driverdelete.setContentText("Driver record has been deleted
successfully");
        Optional<ButtonType> result = driverdelete.showAndWait();

    }
    catch (IOException e) {}

}
```

Code in HelloController class.

Here the program gets the driver name from the text field and then checks if the driver name is there in the text file and if it is found the deleteDriver method from the functions class is called and if it is not found a prompt is displayed to the user saying driver not found.

```
public void delete_driver_function() {
    String driverName = name.getText();
    boolean found = false;

    try {
        File driverFile = new File("driver_details.txt");
        Scanner findRecord = new Scanner(driverFile);
        while (findRecord.hasNextLine()) {
            String data = findRecord.nextLine();
            if (data.contains(driverName)) {
                found = true;
                break;
            }
        }
        findRecord.close();
    }
    catch (FileNotFoundException e) {}

    //if driver name is found in the text file the function works if not a
prompt is displayed to the user
    if (found) {
        functions x = new functions();
        x.deleteDriver(driverName);
    }
    else {
        Alert errorMessage = new Alert(Alert.AlertType.INFORMATION);
        errorMessage.setTitle("Deletion Error");
        errorMessage.setHeaderText("Couldn't Delete Record");
        errorMessage.setContentText("Driver not found please check and
enter again");
        errorMessage.showAndWait();
    }
}
```

2.3 Update Function

This function allows the user to update driver records of a driver by entering the driver's name. This code works in a way that first after entering the driver name the record is deleted from the text file and then all the details along with the ones needed to be updated should be reentered and that will be written in the text file.

Codes in functions class

This method is used to delete the driver record. The name is taken as a parameter for this method.

```
void updateGetDriver(String name){

    try {
        // opening the target file and reading lines
        File driverFile = new File("C:\\Users\\User\\IdeaProjects\\RCC
Management\\driver_details.txt");
        BufferedReader reader = new BufferedReader(new
FileReader(driverFile));
        String line;
        StringBuilder content = new StringBuilder();
        while ((line = reader.readLine()) != null) {
            String[] x = line.split(" ");
            if (!name.equals(x[0])) {
                content.append(line).append("\n");
            }
        }
        reader.close();

        // opening file in write mode and writing the records back except
the record of the user entered driver name
        FileWriter writer = new FileWriter(driverFile);
        writer.write(content.toString());
        writer.close();

    } catch (IOException e) {}
}
```

This method is used to enter all the driver details and write it to a text file. The driver details are taken as parameters. Upon successfully updating a prompt is displayed to the User.

```
void updateDriver(String updatedName, int updatedAge, String
updatedTeam,String updatedCar,int updatedPoints){
    try {
        //re entering all the driver details to update
        FileWriter driverFile = new
FileWriter("C:\\Users\\User\\IdeaProjects\\RCC
Management\\driver_details.txt", true);
        BufferedWriter driver_records = new BufferedWriter(driverFile);
        driver_records.write(updatedName + " " + updatedAge + " " +
updatedTeam + " " + updatedCar + " " + updatedPoints + "\n");
        driver_records.close();
```

```
        //prompt shown to user after successfully updating
        Alert driverupdated = new Alert(Alert.AlertType.INFORMATION);
        driverupdated.setTitle("Update driver record");
        driverupdated.setContentText("Driver record has been successfully
updated");
        Optional<ButtonType> result = driverupdated.showAndWait();
    } catch (Exception e) {
        return;
    }
}
```

Code in HelloController class.

The text file that contains the driver details is read and it checks if name is found inside the text file if yes the updated details obtained from the text field using the getText() method and required validations are done just like in the add a driver function then the 2 methods belonging to the updating part from the function classes are called. If the driver name is not found in the text file a prompt saying so will be displayed to the user.

```
public void update_driver_function() {

    // deleting the driver record
    String driverName = name.getText();
    boolean found = false;
    try {
        File driverFile = new File("driver_details.txt");
        Scanner driverCheck = new Scanner(driverFile);
        while (driverCheck.hasNextLine()) {
            String driverNameCheck = driverCheck.nextLine();
            if (driverNameCheck.contains(driverName)) {
                found = true;
                break;
            }
        }
        driverCheck.close();
    }
    catch (FileNotFoundException e) {}

    //if driver name is found in the text file the function works if not a
prompt is displayed to the user
    if (found) {
        String updated_d_name = updated_name.getText();
        String updated_d_team = updated_team.getText();
        String updated_d_car = updated_car.getText();

        String updated_d_age = updated_age.getText();
        int uDAge = 0;
        // validating age input
        try {
            uDAge = Integer.parseInt(updated_d_age);

        } catch (NumberFormatException e) {
            uplabel.setText("Invalid Entry");
        }


        String updated_d_points = updated_points.getText();
```

```
        int uDPoints = 0;

        try {
            uDPoints = Integer.parseInt(updated_d_points);

        } catch (NumberFormatException e) {
            uplabel.setText("Invalid Entry");
        }
        // if the label is displaying invalid entry user cant add data
until closing and reopening the window
        if (uplabel.getText().equals("Invalid Entry")){
            Alert wrongvalue = new Alert(Alert.AlertType.INFORMATION);
            wrongvalue.setTitle("Invalid Entry");
            wrongvalue.setHeaderText("Couldn't Add Record !");
            wrongvalue.setContentText("Please Enter only numeric values to
Age or Points"+"\n"+"\n"+"CLOSE AND REOPEN WINDOW BEFORE TRYING AGAIN");
            wrongvalue.showAndWait();
        }
        else
        {
            functions x = new functions();
            x.updateGetDriver(driverName);
            x.updateDriver(updated_d_name, uDAge, updated_d_team,
updated_d_car, uDPoints);
        }
    }

    else
    {
        Alert errorMessage = new Alert(Alert.AlertType.INFORMATION);
        errorMessage.setTitle("Update Error");
        errorMessage.setHeaderText("Couldn't Update Record");
        errorMessage.setContentText("Driver not found please check and
enter again");
        errorMessage.showAndWait();
    }
}
```

2.4 Viewing rally cross standing table function.

This function allows the user to view the championship table sorted according to the points of each driver.

Code of the class used for table view (vct class).

```
package com.example.rcc_management;

// class for VCT table view
public class vct {
    private String zname;
    private int zage;
    private String zteam;
    private String zcar;
    private int zpoints;

    public vct(String zname, int zage, String zteam, String zcar, int
zpoints) {
        this.zname = zname;
        this.zage = zage;
        this.zteam = zteam;
        this.zcar = zcar;
        this.zpoints = zpoints;
    }

    public String getZname() {
        return zname;
    }

    public int getZage() {
        return zage;
    }

    public String getZteam() {
        return zteam;
    }

    public String getZcar() {
        return zcar;
    }

    public int getZpoints() {
        return zpoints;
    }

}
```

Code of HelloController class.

The driver file is read and the details are added to various lists throughout the code to sort and then write the details in sorted manner to another text file called vctsortedfile and reading that file and filling the tale data in table view.

```
public void dis() throws IOException {
    // opening a file in read mode to read the lines in that file
    BufferedReader driverFile = new BufferedReader(new
FileReader("driver_details.txt"));
    List<String> lines = new ArrayList<>();
    String line = driverFile.readLine();
    while (line != null) {
        lines.add(line);
        line = driverFile.readLine();
    }
    driverFile.close();

    // two lists are implemented to append the score and driver details
    List<Integer> score = new ArrayList<>();
    List<String[]> driverList = new ArrayList<>();
    for (String x : lines) {
        String[] lined = x.split(" ");
        score.add(Integer.parseInt(lined[4]));
        driverList.add(lined);
    }

    // manually sorting the score which was appended for the score list
    for (int i = 0; i < score.size(); i++) {
        for (int j = i + 1; j < score.size(); j++) {
            if (score.get(i) < score.get(j)) {
                int tempScore = score.get(i);
                score.set(i, score.get(j));
                score.set(j, tempScore);

                String[] tempDriver = driverList.get(i);
                driverList.set(i, driverList.get(j));
                driverList.set(j, tempDriver);
            }
        }
    }

    // implemeting a new list to store the sorted data
    List<String[]> sortedDriverList = new ArrayList<>();

    // appending the above list to get sorted data in descending oreder
    for (int y : score) {
        for (String[] x : driverList) {
```

```
            if (Integer.parseInt(x[4]) == y) {
                sortedDriverList.add(x);
            }
        }
    }

    // opening a file and writing the sorted data list into it
    FileWriter vctfile = new FileWriter("vctsortedfile.txt");
    for (String[] k : sortedDriverList) {
        vctfile.write(String.join(" ", k) + "\n");
    }
    vctfile.close();

    ObservableList<vct> tableData = FXCollections.observableArrayList();


// opening a file in read mode to read the lines in the sorted file
    BufferedReader file2 = new BufferedReader(new
FileReader("vctsortedfile.txt"));
    String line2 = file2.readLine();
    while (line2 != null) {
        String[] sorteddata = line2.split(" ");
        // create a new User object and add it to the observable array list
        tableData.add(new vct(sorteddata[0],
Integer.parseInt(sorteddata[1]), sorteddata[2], sorteddata[3],
Integer.parseInt(sorteddata[4])));
        line2 = file2.readLine();
    }
    file2.close();

// setting up the columns in table view
    zname.setCellValueFactory(new PropertyValueFactory<vct,
String>("zname"));
    zage.setCellValueFactory(new PropertyValueFactory<vct,
Integer>("zage"));
    zteam.setCellValueFactory(new PropertyValueFactory<vct,
String>("zteam"));
    zcar.setCellValueFactory(new PropertyValueFactory<vct,
String>("zcar"));
    zpoints.setCellValueFactory(new PropertyValueFactory<vct,
Integer>("zpoints"));

// adding data into the table view
    ztable.setItems(tableData);


}
```

2.5 Stimulate random race function.

This function generates a random race and saves it to a text file.

Code of the class used for table view (randomrace class).

```
package com.example.rcc_management;

// class for random race generation function table view
public class randomrace {
    private int randomposition;
    private String randomname;
    private int randompoints;

    public randomrace(String randomposition, String randomname, String
randompoints) {
        this.randomposition = Integer.parseInt(randomposition);
        this.randomname = randomname;
        this.randompoints = Integer.parseInt(randompoints);
    }

    public int getRandomposition() {
        return randomposition;
    }

    public String getRandomname() {
        return randomname;
    }

    public int getRandompoints() {
        return randompoints;
    }
}
```

Code of HelloController class.

Date and locations are randomly generated and stored. The driver names in the list of drivers taken from the driver file are shuffled for changing the driver's name pattern and positions and points are assigned to each of them. The details generated at that time are displayed to user using a table view. Then all these data are saved to another text file.

```
public void randomdis() throws IOException {
    // generating a random date
    Random randomGeneration = new Random();
```

```
    int day = randomGeneration.nextInt(25) + 1;
    int month = randomGeneration.nextInt(12) + 1;
    int year = randomGeneration.nextInt(2) + 2023;
    LocalDate date = LocalDate.of(year, month, day);
    String raceDate = date.toString();

    // generating a random location out of given locations
    List<String> locations = Arrays.asList("Nyirad", "Holjes",
"Montalegre", "Barcelona", "Riga", "Norway");
    String raceLocation =
locations.get(randomGeneration.nextInt(locations.size()));

    // opening the file which contains driver details to read driver names
    List<String> driverFile =
Files.readAllLines(Paths.get("driver_details.txt"));
    FileWriter raceFile = new FileWriter("racefile.txt", true);

    // name list implemented to store the driver names
    List<String> nameList = new ArrayList<>();
    // reading and retrieving driver name from champion standing order file
to be passed to a list
    for (String line : driverFile) {
        String[] getName = line.split(" ");
        String driverName = getName[0];
        nameList.add(driverName);
    }

    // position list and points list implemented to store the position and
points in a list
    List<Integer> positionList = new ArrayList<>();
    List<Integer> pointsList = new ArrayList<>();
    // using while loop to generate position for the user and points
depending on the position
    int count = 1;
    while (count <= driverFile.size()) {
        positionList.add(count);
        if (count == 1) {
            pointsList.add(10);
        } else if (count == 2) {
            pointsList.add(7);
        } else if (count == 3) {
            pointsList.add(5);
        } else {
            pointsList.add(0);
        }
        count++;
    }

    // shuffling the driver names
    Collections.shuffle(nameList);

    // implementing a new list to store race data
    List<List<Object>> raceDataList = new ArrayList<>();
    // using z variable to increase the index values of the elements inside
the new list
    int z = 0;
    while (z < driverFile.size()) {
        List<Object> j = new ArrayList<>();
        j.add(positionList.get(z));
        j.add(nameList.get(z));
        j.add(pointsList.get(z));
```

```
        raceDataList.add(j);
        z++;
    }

    // writing data into observable list to add data to table view
    ObservableList<randomrace> raceData =
FXCollections.observableArrayList();
    for (List<Object> list : raceDataList) {
        String position = list.get(0).toString();
        String name = list.get(1).toString();
        String points = list.get(2).toString();
        raceFile.write(raceLocation+" "+raceDate+" "+position+" "+name+"
"+points+"\n");
        raceData.add(new randomrace(position,name,points));
    }

    raceFile.close();


    //table view column setup
    randomname.setCellValueFactory(new
PropertyValueFactory<randomrace,String>("randomname"));
    randomposition.setCellValueFactory(new
PropertyValueFactory<randomrace,Integer>("randomposition"));
    randompoints.setCellValueFactory(new
PropertyValueFactory<randomrace,Integer>("randompoints"));

    randomracetable.setItems(raceData);
    datelabel.setText(raceDate);
    locationlabel.setText(raceLocation);

}
```

2.6 Display all races function.

This function displays all the races saved in the text file to the user.

Code of class used for table view(displayrace class).

```
package com.example.rcc_management;

// class for displaying all races function table view
public class displayrrace {
    private String racedate;
    private String  racelocation;
    private int  raceposition;
    private String  racedriver;

    private int racepoints;

    public displayrrace(String racedate, String racelocation, String
raceposition, String racedriver, String racepoints) {
        this.racedate = racedate;
        this.racelocation = racelocation;
        this.raceposition = Integer.parseInt(raceposition);
        this.racedriver = racedriver;
        this.racepoints = Integer.parseInt(racepoints);
    }

  public String getRacedate() {
        return racedate;
    }

    public String getRacelocation() {
        return racelocation;
    }

    public int getRaceposition() {
        return raceposition;
    }

    public String getRacedriver() {
        return racedriver;
    }

    public int getRacepoints() {
        return racepoints;
    }
}
```

Code of HelloController class.

The race file which was saved in the random race generating function is read and then the details off all the races are shown to the user.

```
public void allrandom() throws IOException  {
```

```
    ObservableList<displayrrace> data =
FXCollections.observableArrayList();

    //reading race file
    BufferedReader raceFile = new BufferedReader(new
FileReader("racefile.txt."));
    String line2 = raceFile.readLine();
    while (line2 != null) {
        String[] j = line2.split(" ");
        // create a new User object and add it to the observable array list
        data.add(new displayrrace(j[0],j[1],j[2],j[3],j[4]));
        line2 = raceFile.readLine();
    }
    raceFile.close();

    //table view column setup
    racedate.setCellValueFactory(new
PropertyValueFactory<displayrrace,String>("racedate"));
    racelocation.setCellValueFactory(new
PropertyValueFactory<displayrrace,String>("racelocation"));
    raceposition.setCellValueFactory(new
PropertyValueFactory<displayrrace,Integer>("raceposition"));
    racedriver.setCellValueFactory(new
PropertyValueFactory<displayrrace,String>("racedriver"));
    racepoints.setCellValueFactory(new
PropertyValueFactory<displayrrace,Integer>("racepoints"));

    allraces.setItems(data);

}
```

2.7 Save function.

This function is used to save details.

Code for this function

The driver details file is read and the exact same is copied to a new file named save file so that user gets to experience a save function in an occasion like updating a record and user should save it before loading to view the updated one.

```
public void save_function() throws IOException {
    FileInputStream loadfile = new FileInputStream("driver_details.txt");
    FileOutputStream savefile = new FileOutputStream("savefile.txt");

    // reading the files in drive details file and writing it to= save file
    int i;
    while((i=loadfile.read())!=-1){
        savefile.write((char)i);
    }
    loadfile.close();
    savefile.close();
    // prompt shown to the user
    Alert savedtotextfile = new Alert(Alert.AlertType.INFORMATION);
    savedtotextfile.setContentText("Data has been saved to a text file");
    savedtotextfile.showAndWait();

}
```

2.8 Load function

This function is used to display the saved details.

Code of the class used for table view (load class).

```
package com.example.rcc_management;

//class for load function table view
public class load {
    private String loadname;
    private int loadage;
    private String loadteam;
    private String loadcar;
    private int loadpoints;

    public load(String loadname, String loadage, String loadteam, String
loadcar, String loadpoints) {
        this.loadname = loadname;
        this.loadage = Integer.parseInt(loadage);
        this.loadteam = loadteam;
        this.loadcar = loadcar;
        this.loadpoints = Integer.parseInt(loadpoints);
    }

    public String getLoadname() {
        return loadname;
    }

    public int getLoadage() {
        return loadage;
    }

    public String getLoadteam() {
        return loadteam;
    }

    public String getLoadcar() {
        return loadcar;
    }

    public int getLoadpoints() {
        return loadpoints;
    }
}
```

Code of HelloController class

The save file is read and displayed to the user using a table view.

```
public void load_function() throws IOException {
    ObservableList<load> data = FXCollections.observableArrayList();

    // reading and displaying data from save file
    BufferedReader saveFile = new BufferedReader(new
FileReader("savefile.txt"));
    String saveFileLines = saveFile.readLine();
    while (saveFileLines != null) {
        String[] x = saveFileLines.split(" ");
        data.add(new load(x[0],x[1],x[2],x[3],x[4]));
        saveFileLines = saveFile.readLine();
    }
    saveFile.close();

    //table view column setup
    loadname.setCellValueFactory(new
PropertyValueFactory<load,String>("loadname"));
    loadage.setCellValueFactory(new
PropertyValueFactory<load,Integer>("loadage"));
    loadteam.setCellValueFactory(new
PropertyValueFactory<load,String>("loadteam"));
    loadcar.setCellValueFactory(new
PropertyValueFactory<load,String>("loadcar"));
    loadpoints.setCellValueFactory(new
PropertyValueFactory<load,Integer>("loadpoints"));

    loadingtable.setItems(data);
    String j = "Use the main menu to edit the above details";
    loadlabel.setText(j);

}
```

2.9 Exit Function.

The program is quitted after asking for confirmation using a prompt and upon stopping the program a statement is printed in the console.

```
public void exit_function() {
    //confirmation prompt from user asking whether to quit the program or
not
    Alert exit = new Alert(Alert.AlertType.CONFIRMATION);
    exit.setTitle("Quit Program");
    exit.setContentText("Are you sure you want to quit");

    if (exit.showAndWait().get() == ButtonType.OK) {
        System.out.println("SYSTEM CLOSED");
        System.exit(0);
    }

}
```

3. Test plans and Test cases

3.1 Test case and plan for Add function.

Test case number	Test Plan	Inputs					Expected Output	Actual Output
		Name	Age	Team	Car	Points		
1	Entering driver record	Walter	25	Godzilla	Mercedes	80	Prompt saying added successfully.	Prompt saying added successfully.
2	Entering driver record	Jessy	28	Risers	Subaru	70	Prompt saying added successfully.	Prompt saying added successfully.
3	Entering driver record	Mason	30	Flyers	Renault	65	Prompt saying added successfully.	Prompt saying added successfully.
4	Entering a string for the age of driver record	Skylar	ggg	Groves	Supra	95	Prompt saying couldn't add record.	Prompt saying couldn't add record.
5	Entering a string for the points of driver record	Mogan	22	Vagos	Dodge	jjj	Prompt saying couldn't add record.	Prompt saying couldn't add record.

Screenshot of text file before running add function test cases.

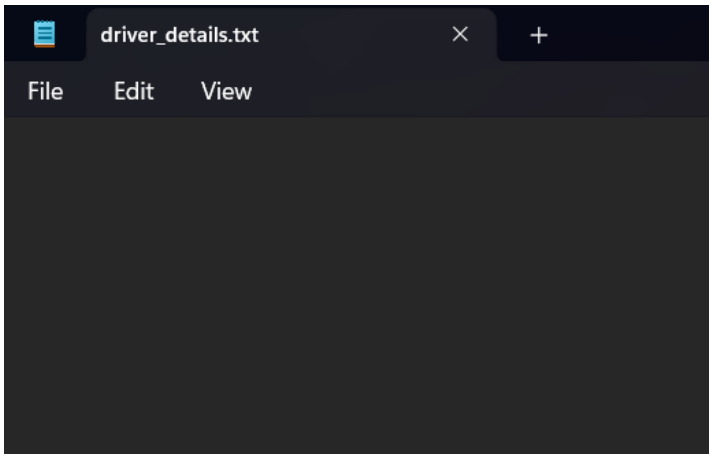

Figure 3: Screenshot of text file before running add function.

Test case number 1 screenshot.

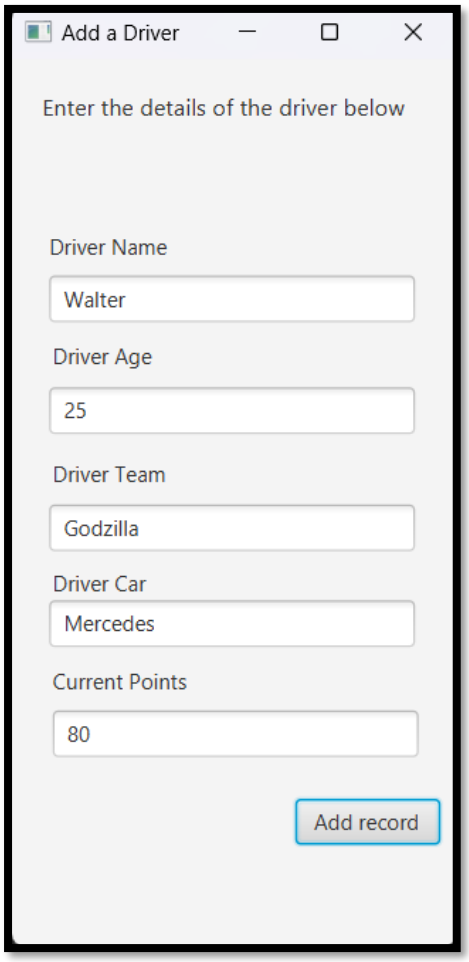

Figure 4: Screenshot of add interface test case 1.

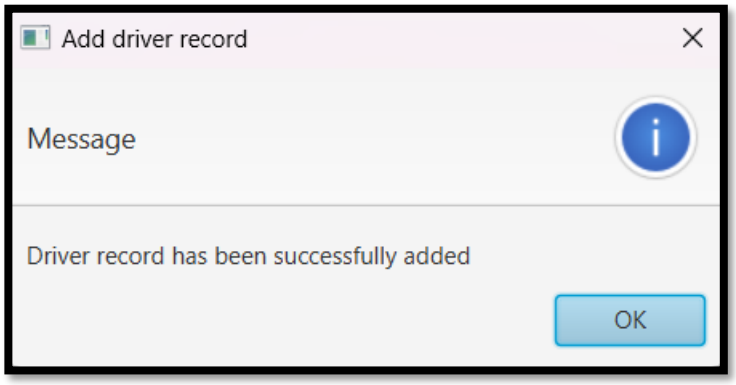

Figure 5: Screenshot of prompt after running test case 1.

Test case number 2 screenshot.

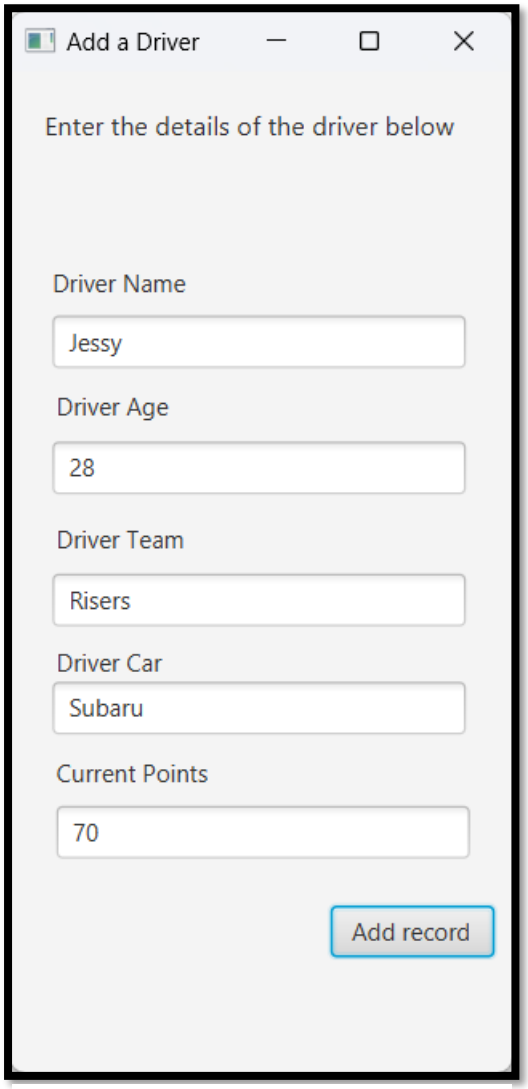

Figure 6: Screenshot of add interface test case 2.

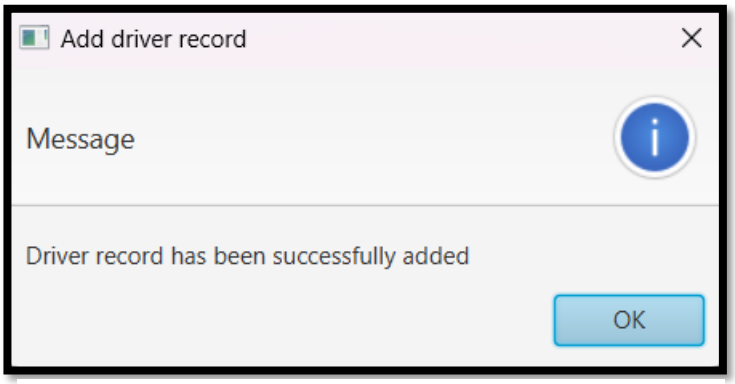

Figure 7: Screenshot of prompt after running test case 2.

Test case number 3 screenshot.

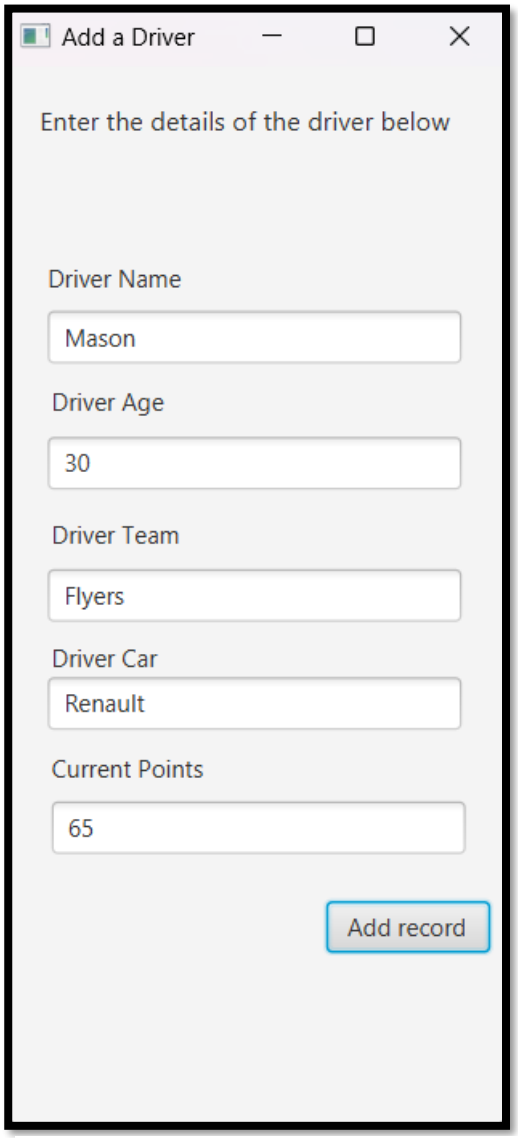

Figure 9: Screenshot of add interface test case 3.

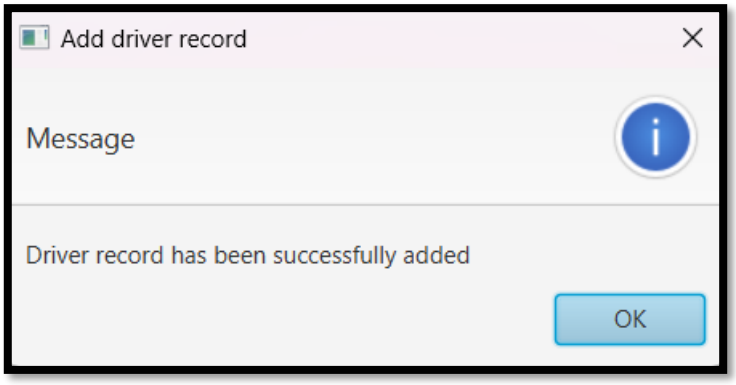

Figure 8: Screenshot of prompt after running test case 3.

Test case number 4 screenshot.

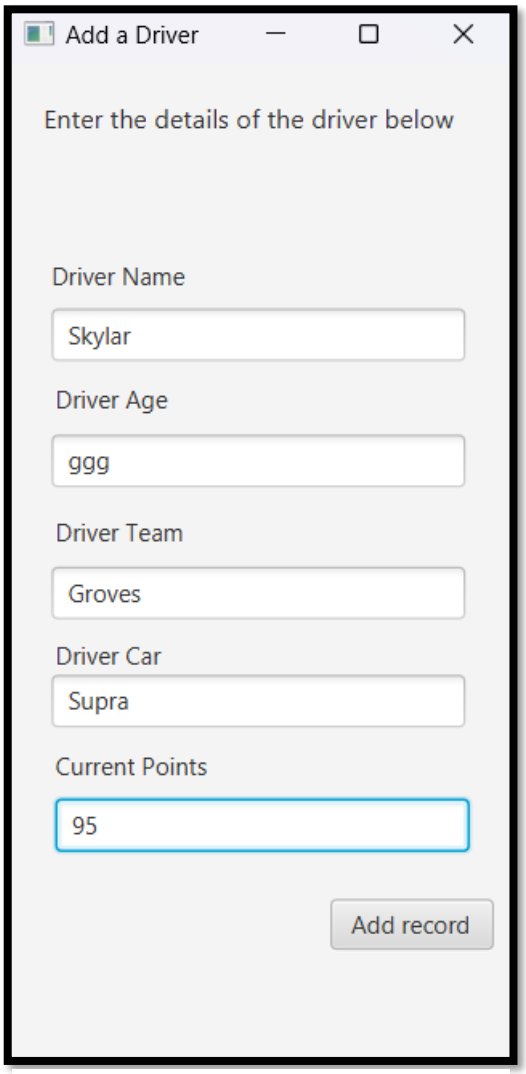

Figure 10: Screenshot of add interface test case 4.

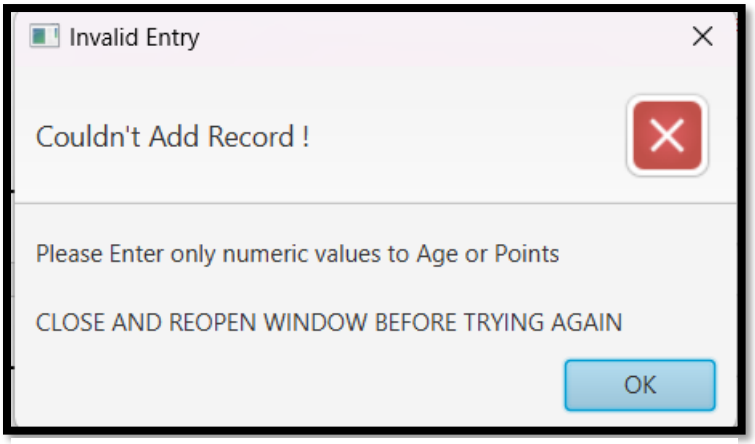

Figure 11:Screenshot of prompt after running test case 4.

Test number 5 screenshot.

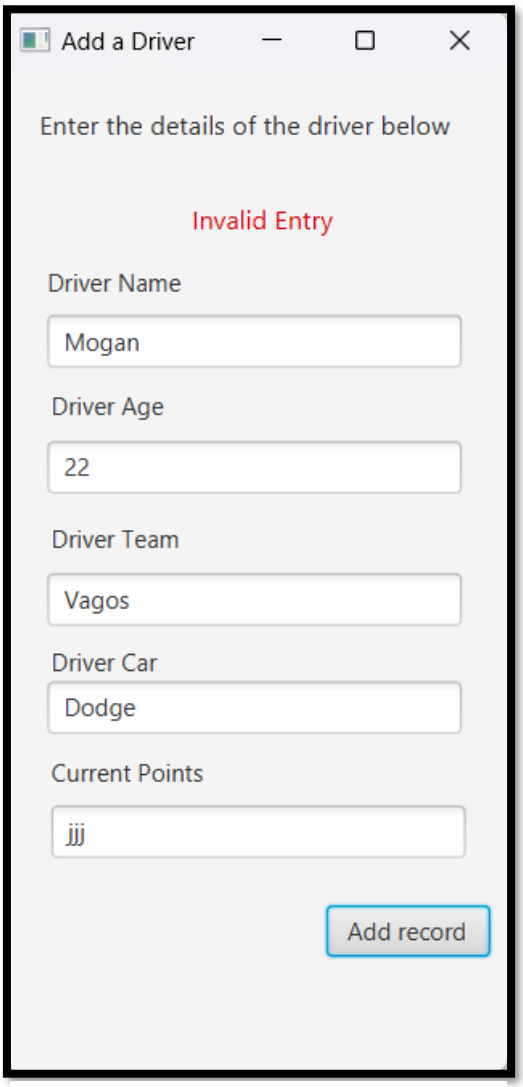

Figure 12:Screenshot of add interface test case 5.

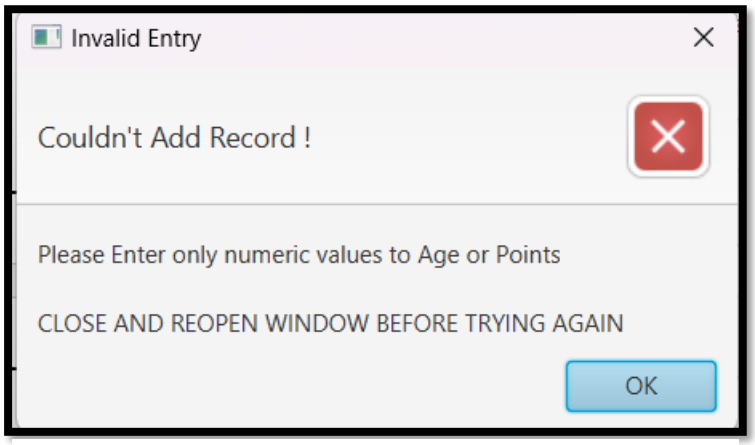

Figure 13:Screenshot of prompt after running test case 5.

Screenshot of text file after running add function test cases.

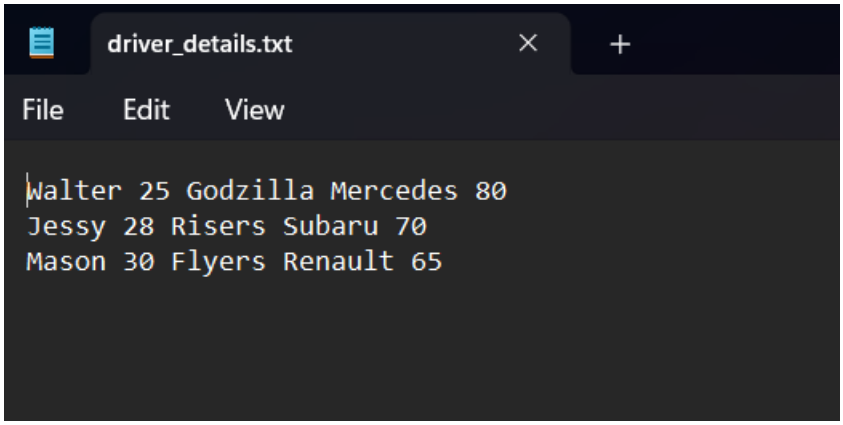

Figure 14: Screenshot of text file after adding function

3.2 Test cases and plan for delete function.

Test case number	Plan	Name	Expected output	Actual output
1	Deleting a driver	Mason	Prompt saying driver deleted successfully	Prompt saying driver deleted successfully
2	Deleting a driver that doesn't exist	Jack	Prompt saying driver not found.	Prompt saying driver not found.

Text file before running test case.

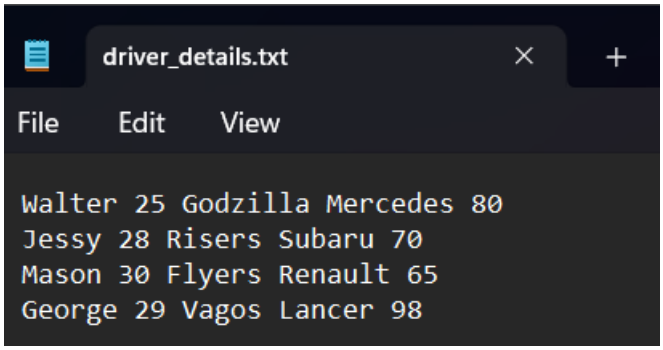

Figure 15: Screenshot of text file before delete function.

Test case number 1 screenshots.

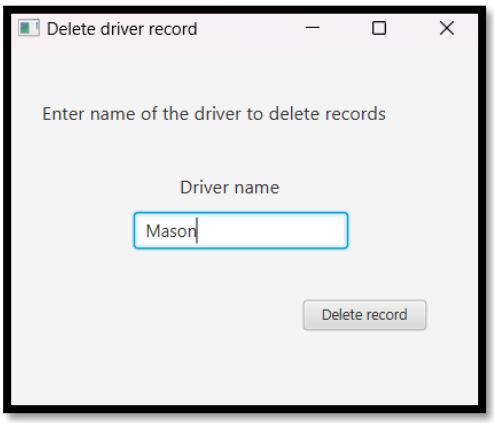

Figure 17: Screenshot of delete interface test case 1.

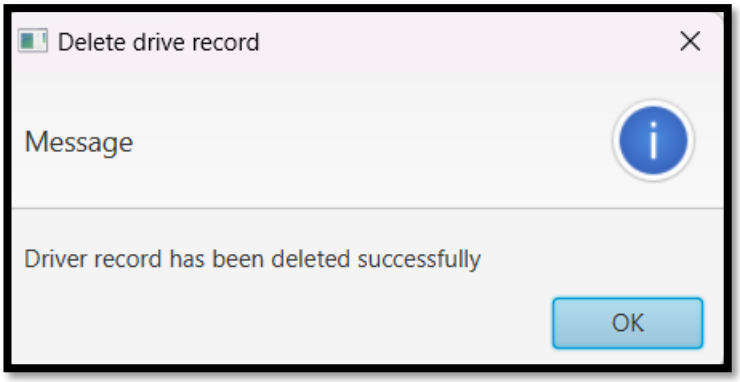

Figure 16:. Screenshot of prompt after running test case 1.

Test case number 2 screenshots.

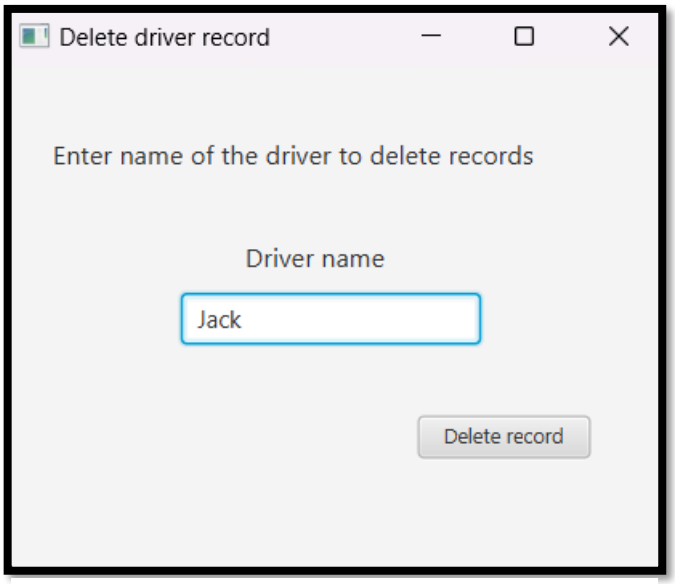

Figure 18: Screenshot of delete interface test case 2.

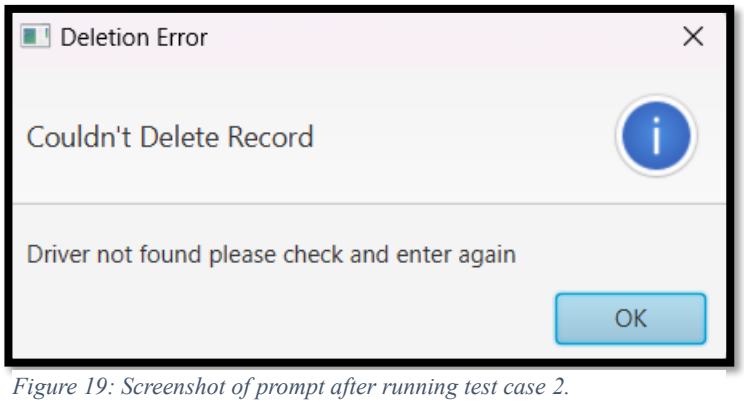

Figure 19: Screenshot of prompt after running test case 2.

Text file after running test cases.

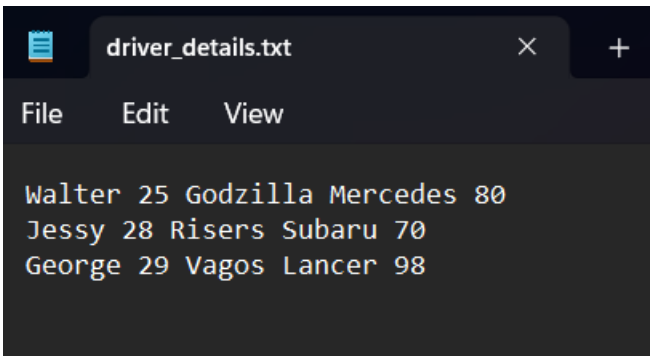

Figure 20: Screenshot of test case after running delete function.

3.3 Test cases for update function.

Test case number	Plan	Driver name	Inputs					Expected output	Actual Output
			Name	Age	Team	Car	Points		
1	Updating a driver	Jessy	Michael	28	Risers	Subaru	70	Driver record successfully updated	Driver record successfully updated
2	Entering incorrect data type for age.	Walter	John	jjj	Godzilla	Mercedes	80	Couldn't add record prompt	Couldn't add record prompt
3	Entering incorrect data type for points.	Walter	John	25	Godzilla	Mercedes	ggg	Couldn't add record prompt	Couldn't add record prompt
4	Entering driver name that does not exist.	Ivan	John	25	Godzilla	Mercedes	85	Driver doesn't exist prompt	Driver doesn't exist prompt

Text file before running test cases.

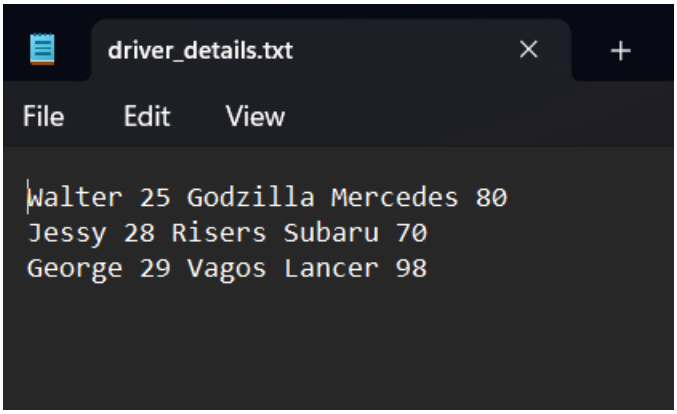

Figure 21: Screenshot of text file before update function.

Test case number 1 screenshots.

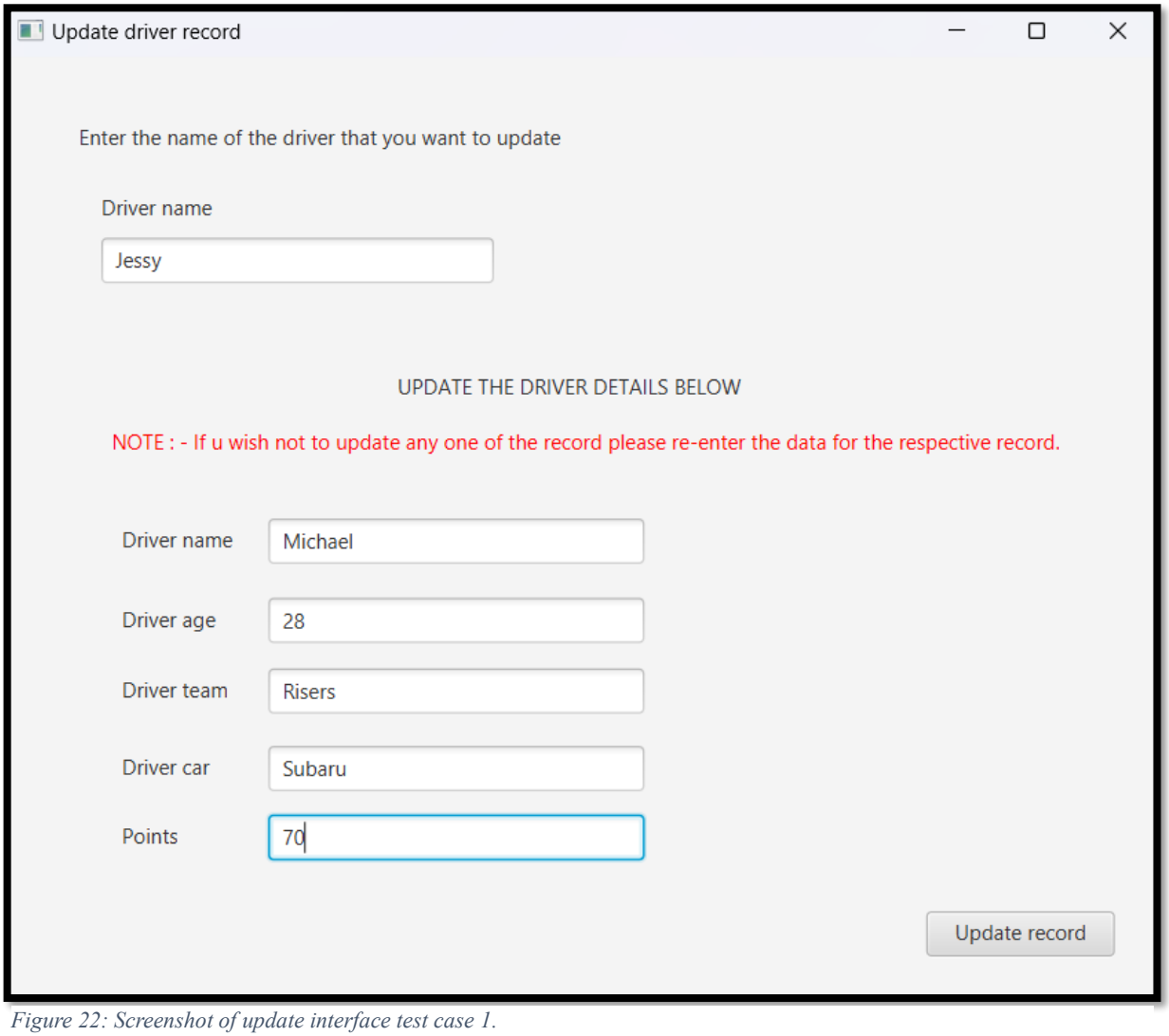

Figure 22: Screenshot of update interface test case 1.

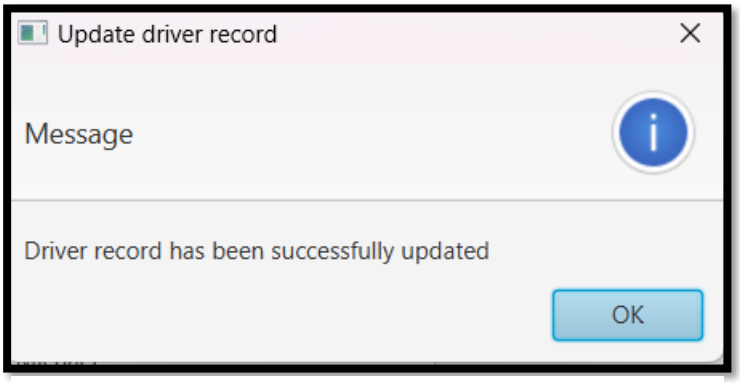

Figure 23:Screenshot of prompt after running test case 1

Test case number 2 screenshots.

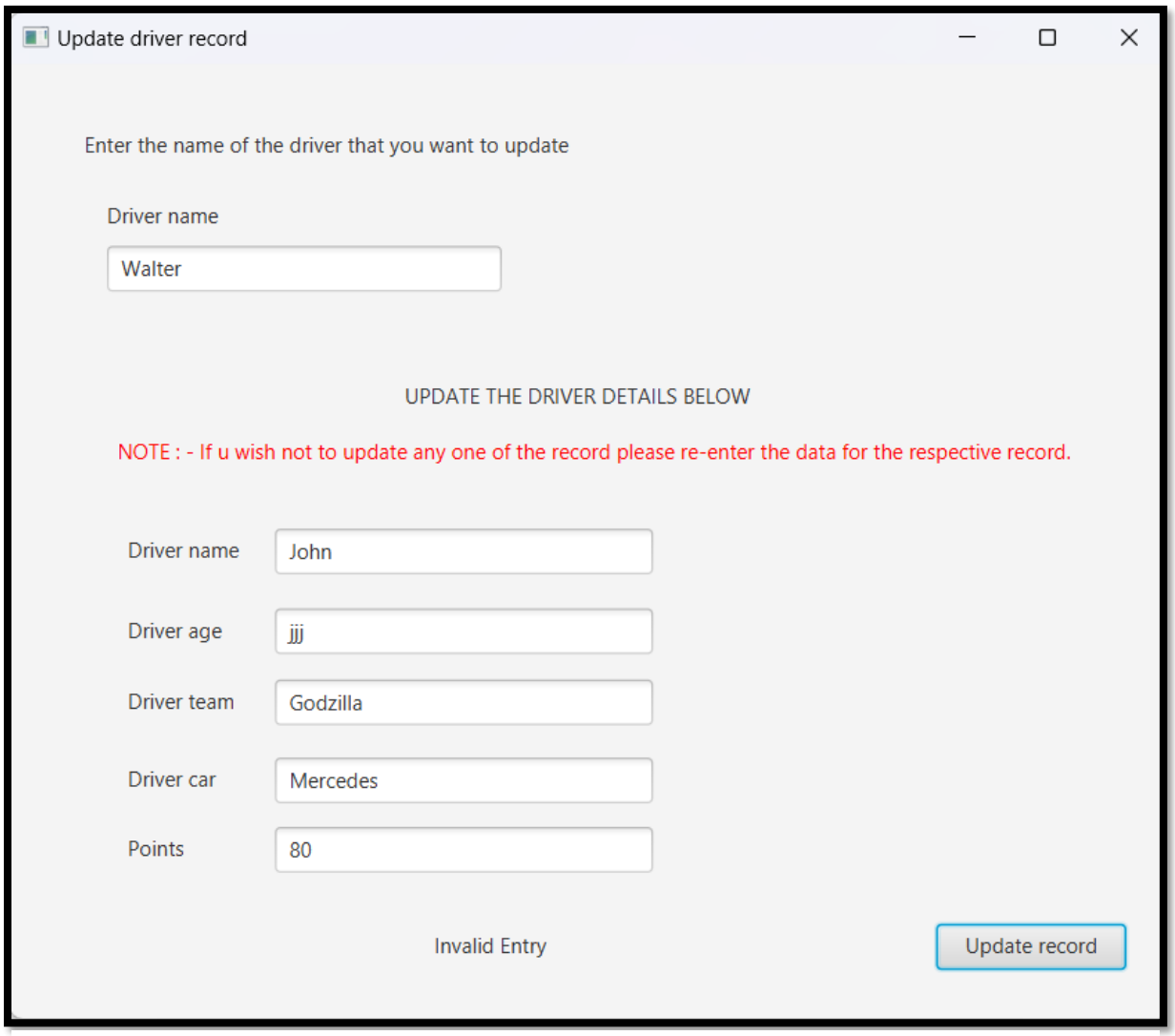

Figure 24: Screenshot of update interface test case 2.

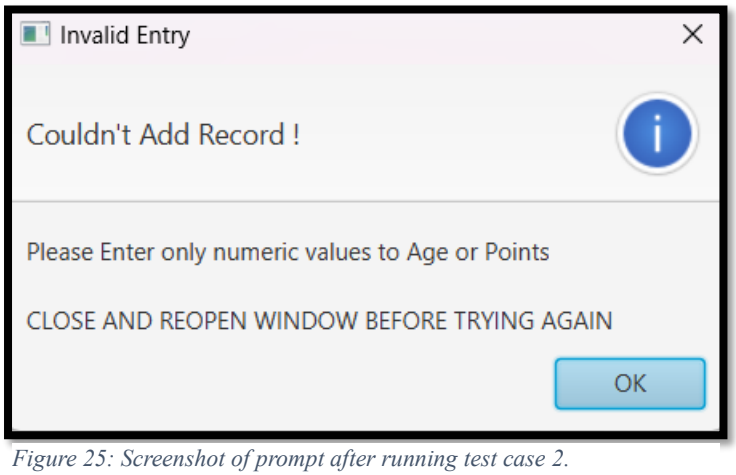

Figure 25: Screenshot of prompt after running test case 2.

Test case number 3 screenshots.

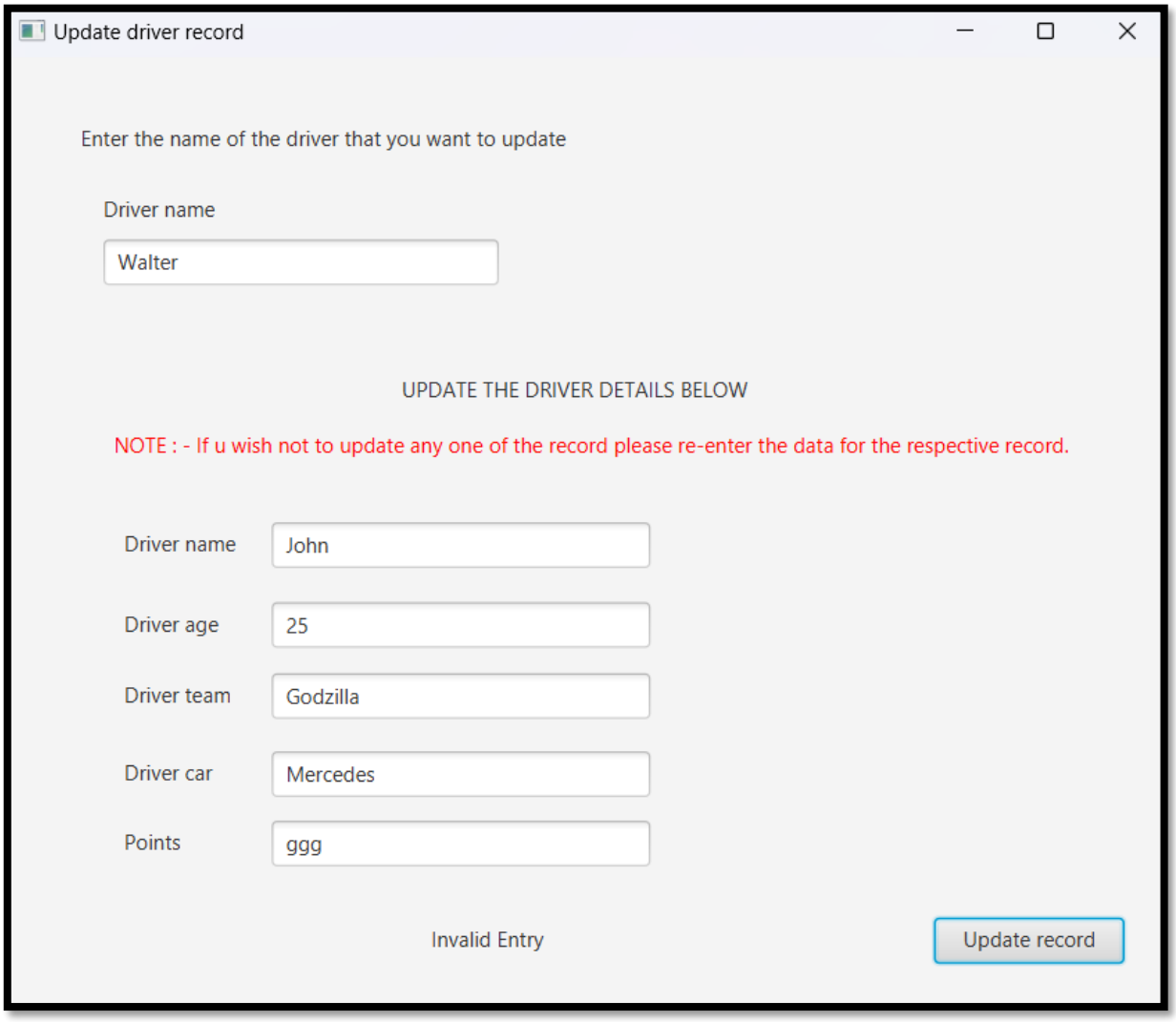

Figure 26: Screenshot of update interface test case 3.

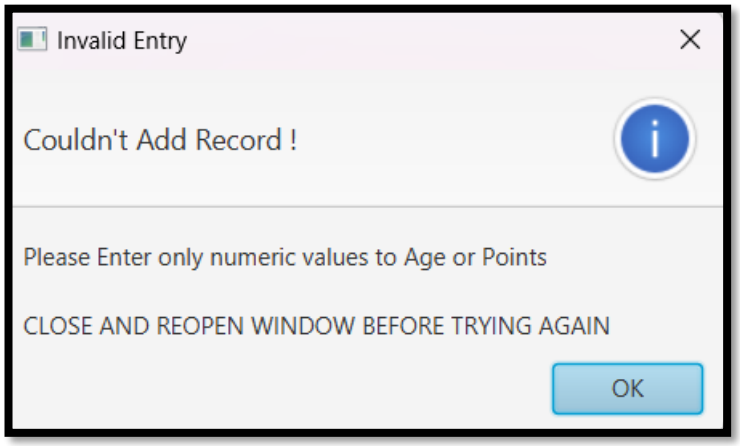

Figure 27: Screenshot of prompt after running test case 3.

Test case number 4 screenshots.

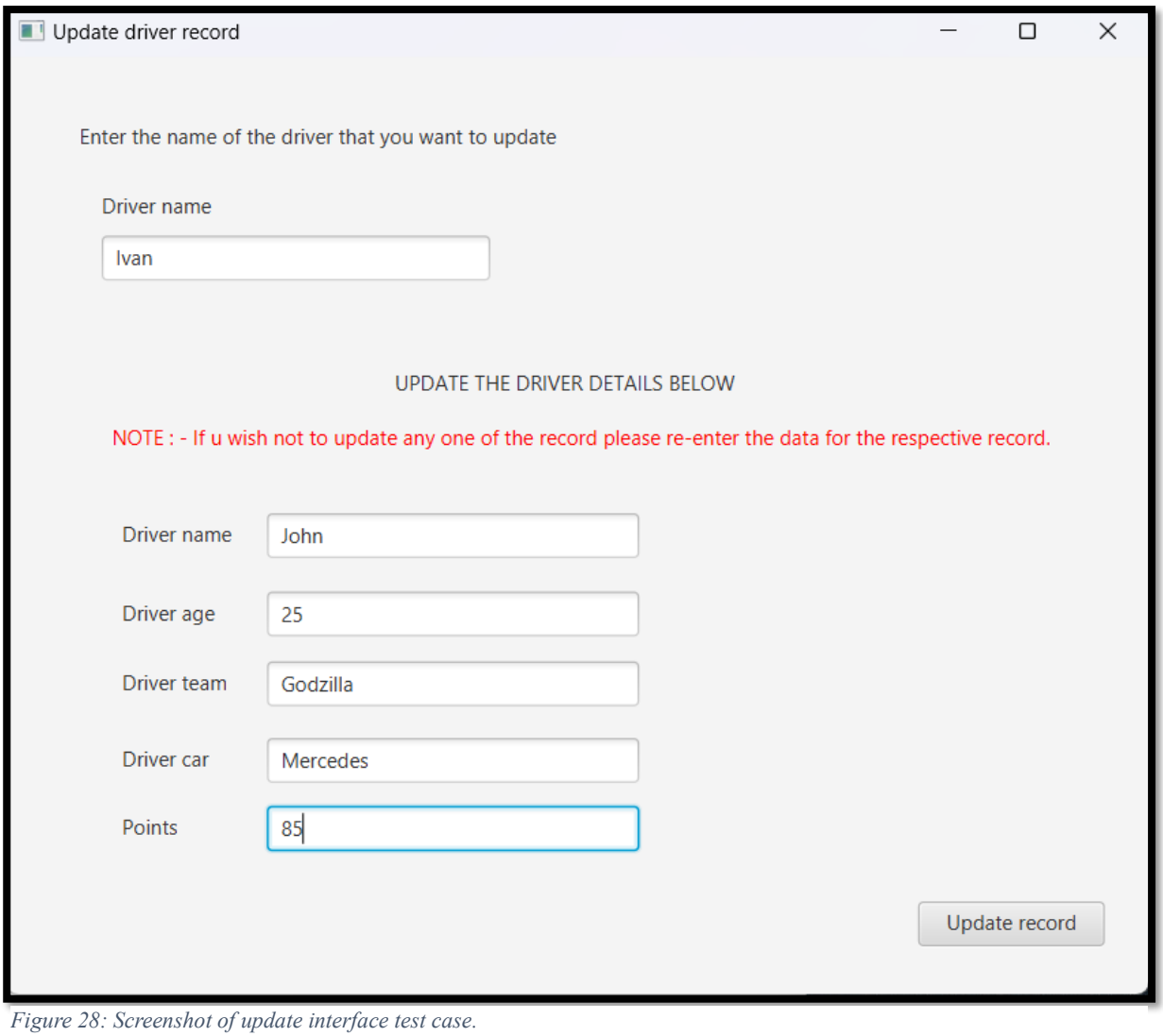

Figure 28: Screenshot of update interface test case.

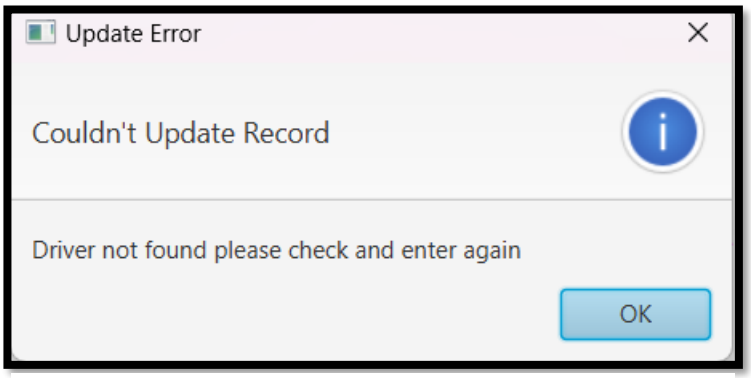

Figure 29: Screenshot of prompt after running test case.

4. Robustness and the maintainability

The add update and delete functions are maintained in another class so that adjustments for those code can be done in that function. Methods are separated in the hello controller class for each function for better maintainability. Navigation to different section is also done inside methods for future adjustments such as window sizes if required and code commenting is done wherever necessary for future reference or better understanding.

5. Conclusion and Assumptions.

1. The points after stimulating the races should be added manually.

2. As there is no primary key given, I assumed it as the name of the driver.

6. References

www.roseindia.net. (n.d.). *How to Write to a File in Java without overwriting*. [online] Available at: https://www.roseindia.net/java/javafile/How-to-Write-to-a-File-in-Java-without-overwriting.shtml.

www.youtube.com. (n.d.). *java program to copy content from one file to another file | Learn Coding*. [online] Available at: https://youtu.be/Z2CaeP4T9O4.

www.javatpoint.com. (n.d.). *Java String join() method - javatpoint*. [online] Available at: https://www.javatpoint.com/java-string-join.

W3Schools (2019). *Java ArrayList*. [online] W3schools.com. Available at: https://www.w3schools.com/java/java_arraylist.asp.

www.youtube.com. (n.d.). *JavaFX TextField*. [online] Available at: https://youtu.be/gN29Y600k5g.

www.javatpoint.com. (n.d.). *Java List - javatpoint*. [online] Available at: https://www.javatpoint.com/java-list.

www.tutorialspoint.com. (n.d.). *How to create an alert in JavaFX?* [online] Available at: https://www.tutorialspoint.com/how-to-create-an-alert-in-javafx.

Villan, M.A. (2022). *JavaFX ObservableList Tutorial*. [online] Genuine Coder. Available at: https://genuinecoder.com/javafx-observable-list-tutorial/.

www.youtube.com. (n.d.). *Encapsulation | Object Oriented Programming (OOP) Sinhala Tutorial | Part 18*. [online] Available at: https://youtu.be/gBfs4Ai6VOw.

www.youtube.com. (n.d.). *Classes and Objects With Codings | Object Oriented Programming (OOP) Sinhala Tutorial | Part 04*. [online] Available at: https://youtu.be/K2ouSyPwxgY.

www.youtube.com. (n.d.). *Classes and Objects | Object Oriented Programming (OOP) Sinhala Tutorial | Part 03*. [online] Available at: https://youtu.be/8HMEc5KRQ1A.

www.youtube.com. (n.d.). *JavaFX + Scene Builder - Navigation Between Pages*. [online] Available at: https://youtu.be/nmpRP8mT2nU.

www.youtube.com. (n.d.). *Java File Input/Output - It's Way Easier Than You Think*. [online] Available at: https://youtu.be/ScUJx4aWRi0.

www.youtube.com. (n.d.). *JavaFX and Scene Builder Course - IntelliJ #30: TableView and TableColumn*. [online] Available at: https://youtu.be/fnU1AlyuguE.

7.Other GUI parts.

The Main menu Page.

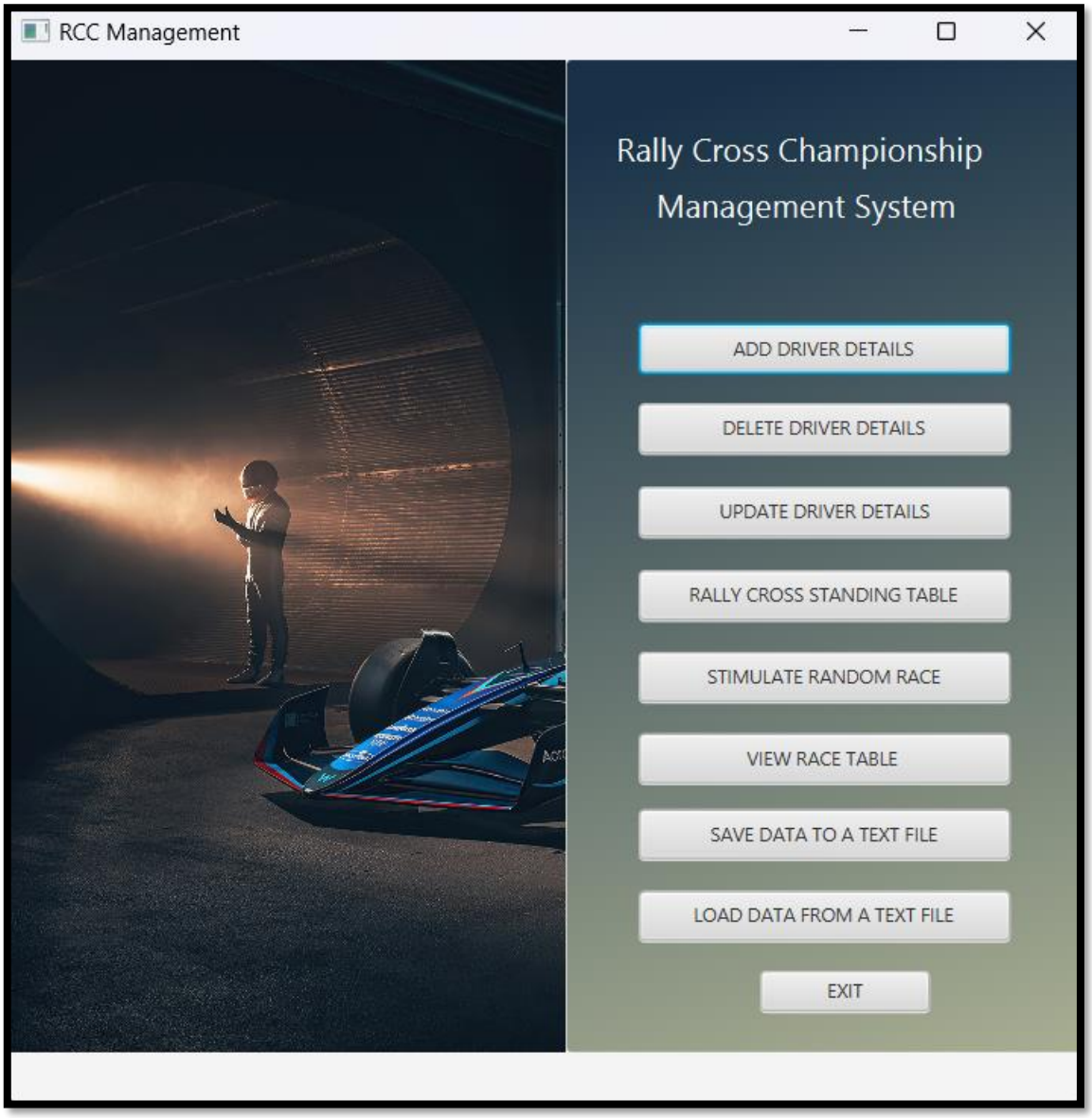

Figure 30: Main menu page of the program.

Rally cross standing table interface.

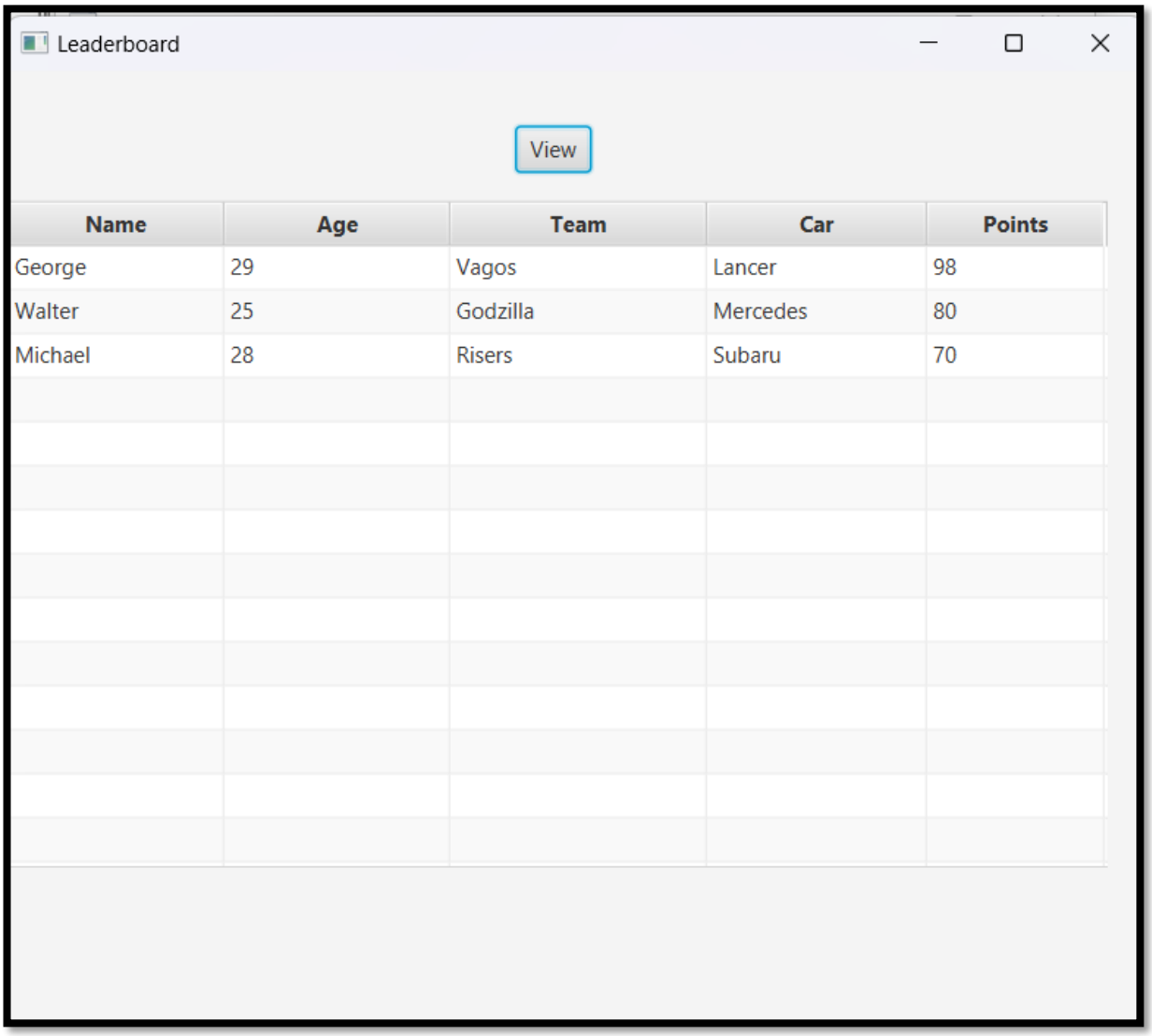

Figure 31: Table interface to view drivers.

Stimulating random race interface.

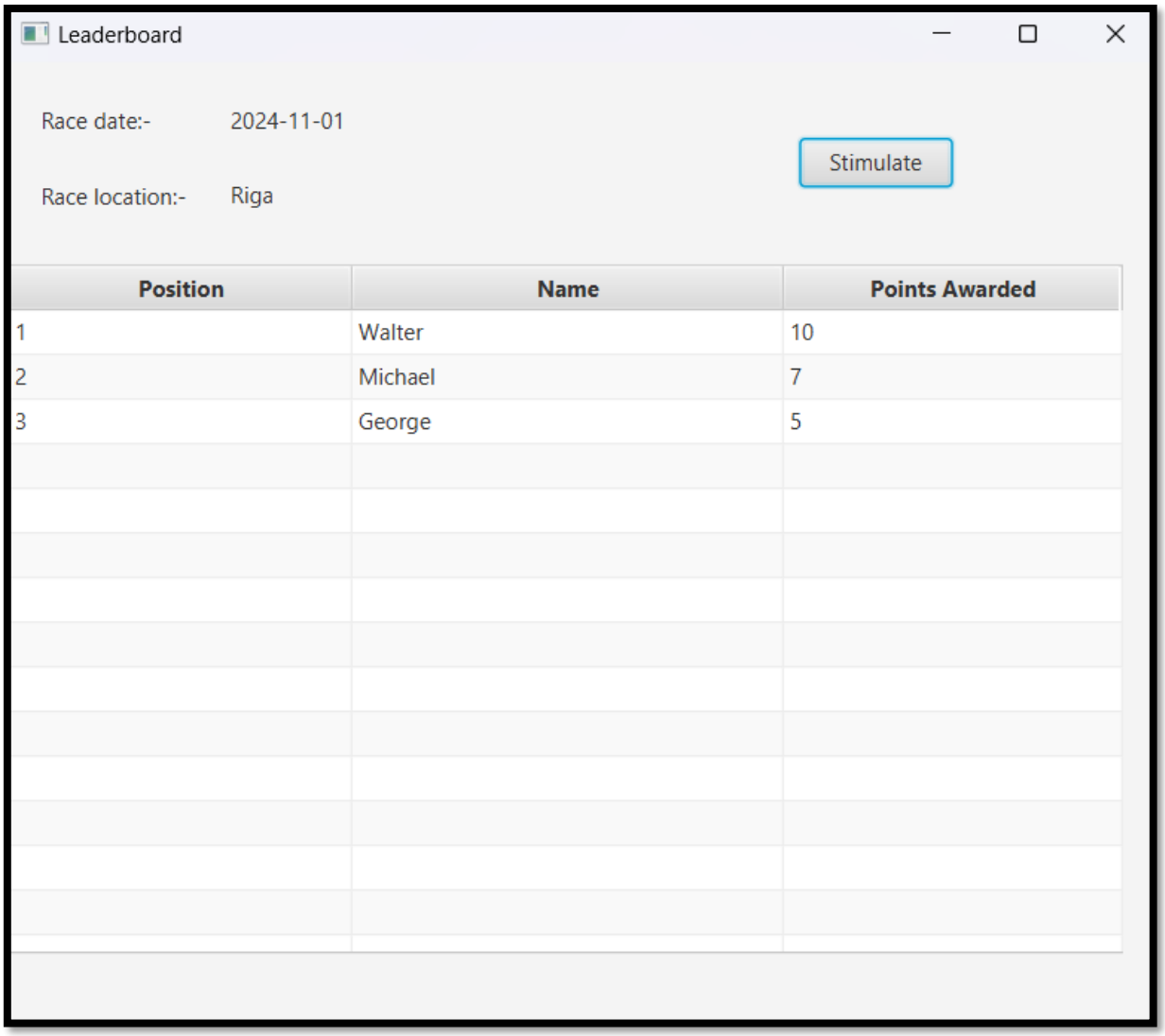

Figure 32: Random race function.

All race view function.

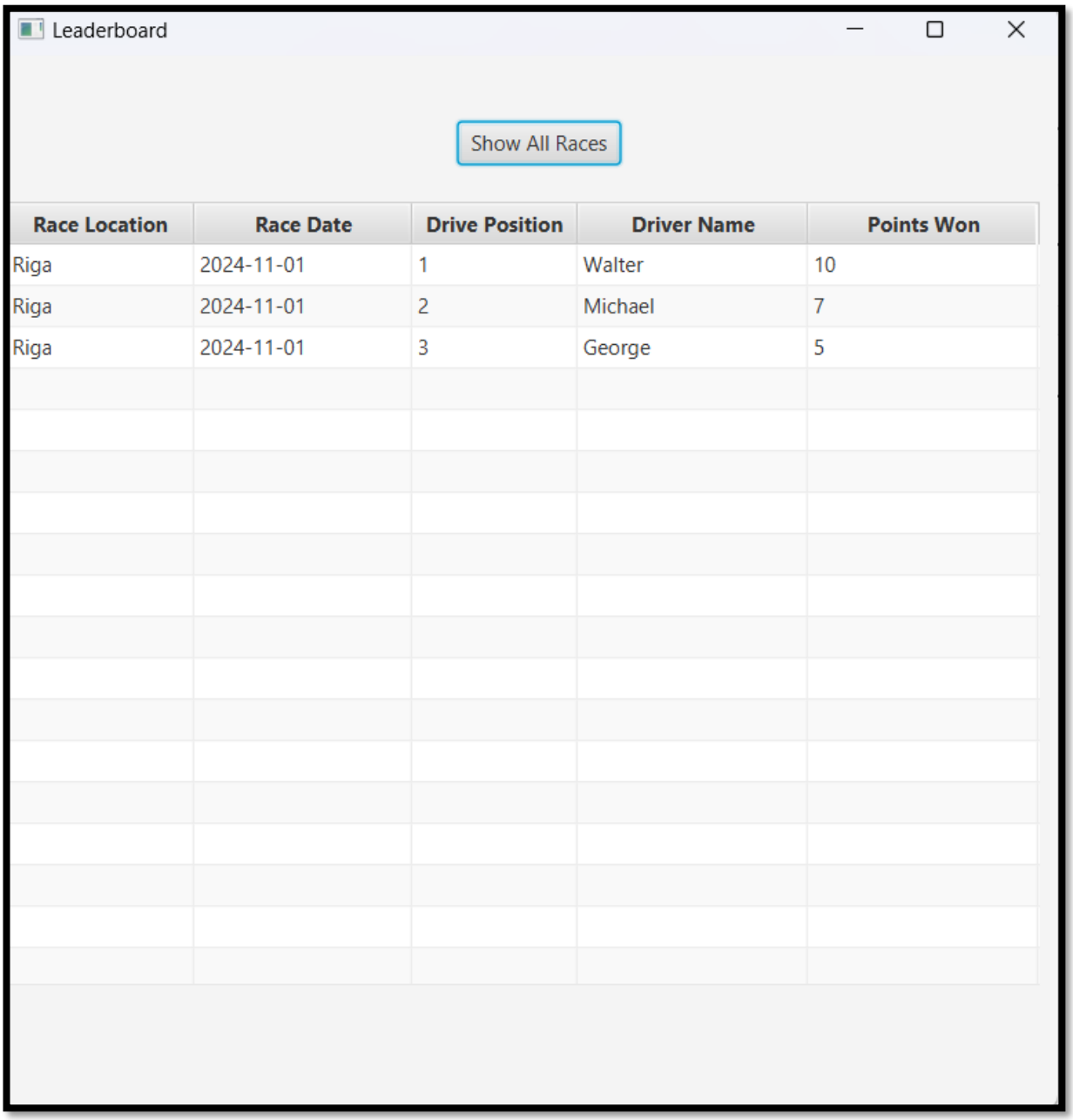

Figure 33: All race views table interface.

Save prompt after saving a file.

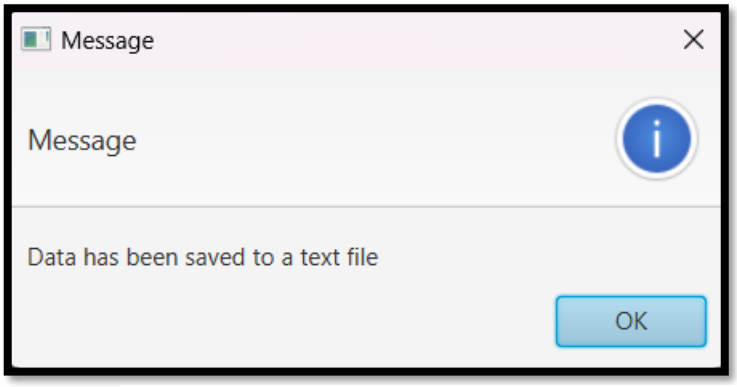

Figure 34: Prompt after saving the text file.

Interface displayed when loading a text file.

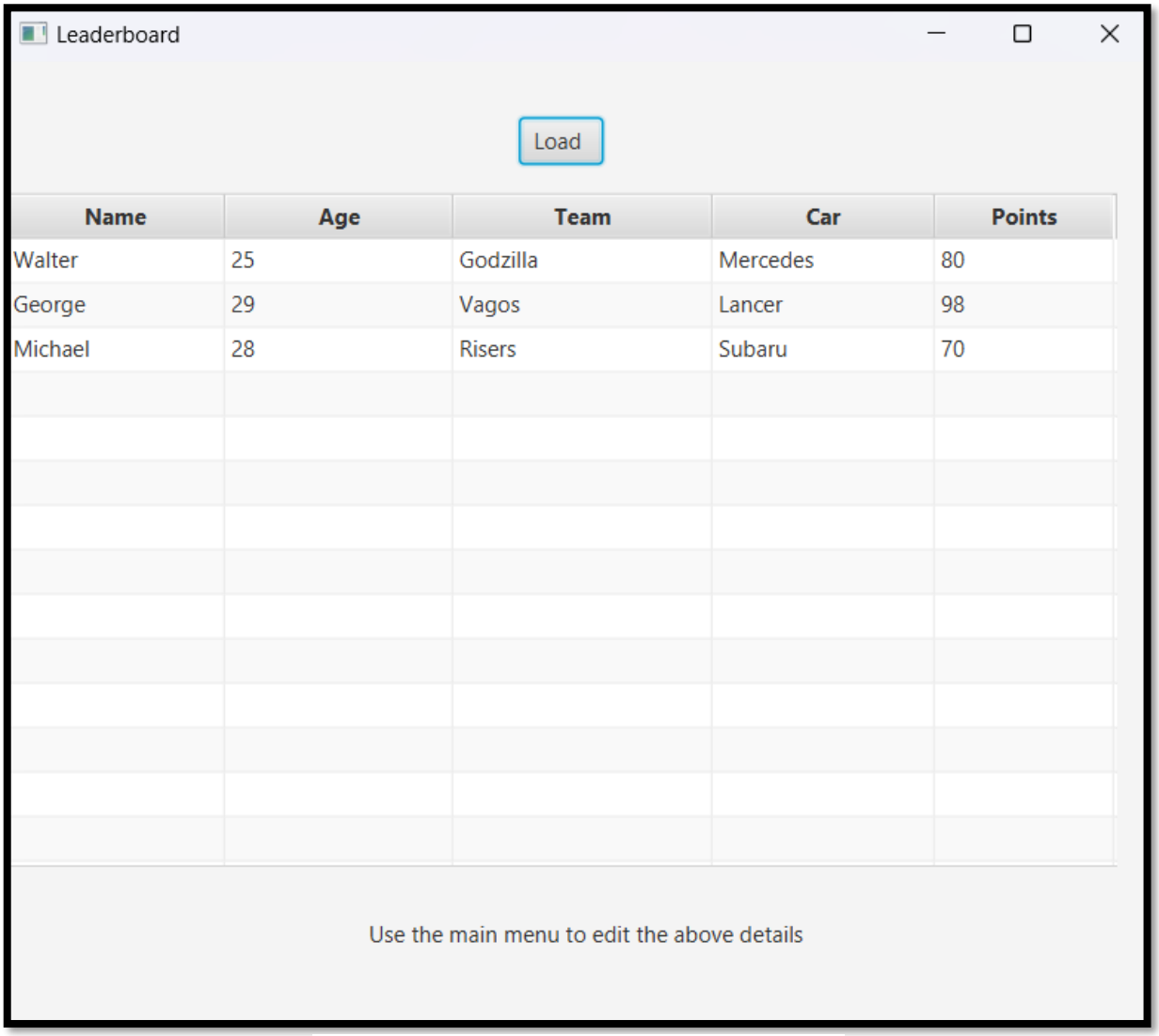

Figure 35: Load function table view.

Confirmation prompt when quitting the program.

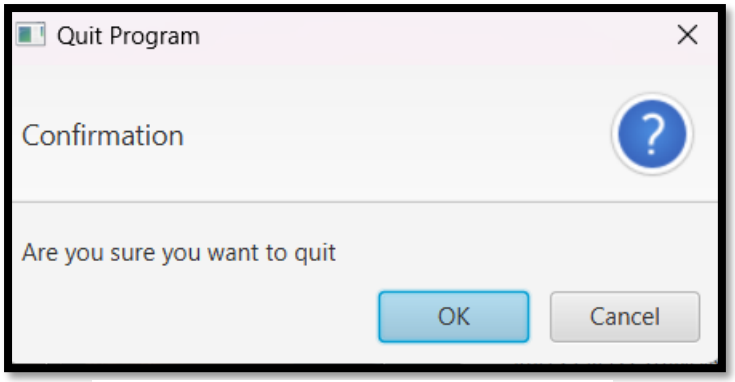

Figure 36: Confirmation prompt when quitting.