

## **BSc (Hons) Artificial Intelligence and Data Science**

Module: CM 1601 Programming Fundamentals 2

Individual Coursework Report

Module Leader: Ms. Sachinthani Perera

RGU Student ID : 2330912

IIT Student ID : 20230437

Student Name : Gihanga Sandothmi

## **Acknowledgment**

I want to express my sincere gratitude to everyone who helped me finish this coursework. First and foremost, I would like to express my sincere gratitude to my family for their consistent support, comprehension, and encouragement during this academic path. In addition, I owe my friends for their unwavering support and companionship during the difficulties encountered with this schoolwork. In addition, I would like to thank my mentors and educators for their advice and inspiration. Their knowledge, helpful criticism, and support have been crucial in forming my perspective on the topic and how I approach it. Finally, I would want to express my gratitude to all the writers, scholars, and experts whose contributions and wisdom have shaped and enhanced the information in this study.

## **Executive Summary**

This report includes a thorough examination of a JavaFX program that was developed using Scenebuilder and written in Java. The program was constructed to fulfill the requirements of this class. Parts of the code needed to implement this software are included in this report. The mentioned code snippet is then described in detail. In addition, the codes are supported by pictures showing the user interface. The report includes a Summary, an Acknowledgment, and References as extra material.

## **Table of Contents**

Acknowledgment .....	2
Executive Summary .....	3
Table of Contents .....	4
Table of Figures .....	6
Flowcharts .....	7
Add horse details.....	7
Update horse details .....	8
Delete horse details .....	9
View horse details .....	9
Save horse details.....	10
Simulate for major round .....	10
Winning horse details.....	11
Visualizing horse details .....	11
Introduction to functions with code. ....	12
Add horse details function .....	12
Code .....	12
Code Explanation.....	14
Update horse details function.....	15
Code .....	15
Code Explanation.....	18
Delete horse details function.....	19
The code.....	19
Code explanation .....	19
View horse details method .....	20
The code.....	20
Code explanation .....	21
Save horse details function .....	22
The code.....	22
Code explanation .....	23
Simulating data for the major round function.....	24

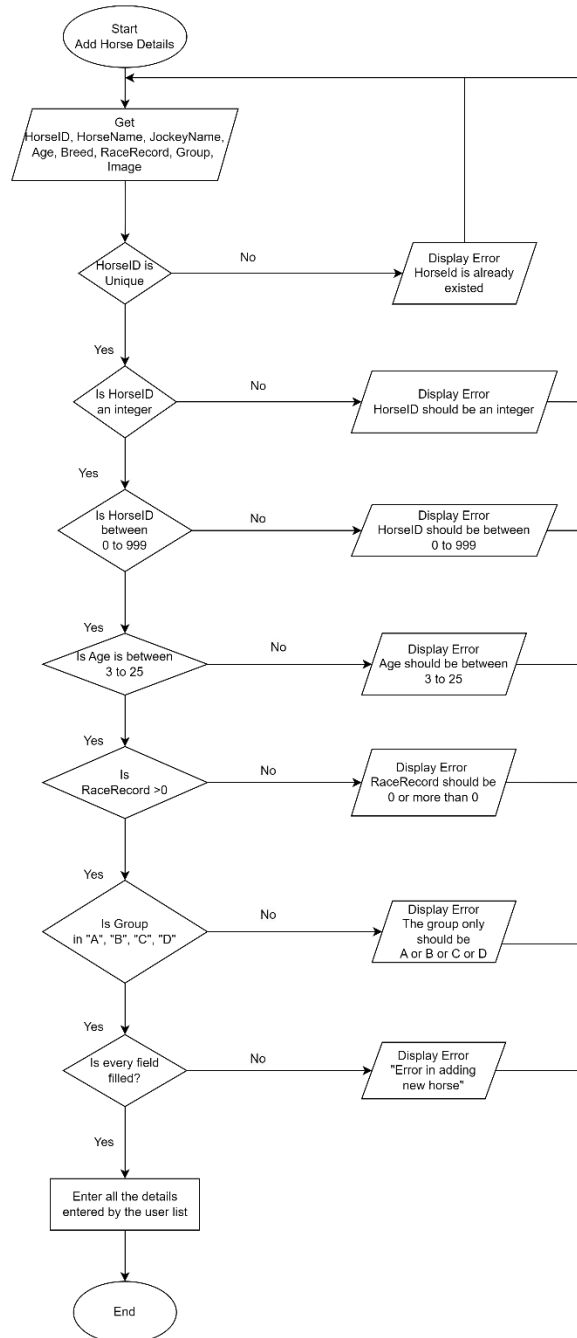
The code.....	24
Code explanation .....	26
Winning horse details function .....	27
The code.....	27
Code explanation .....	28
Visualizing winning horse details function.....	29
The code.....	29
Code explanation .....	29
Horse image method .....	30
The code.....	30
Code explanation .....	30
Test plans and Test cases.....	31
Test Cases.....	31
Robustness & Maintainability.....	39
Conclusion & Assumptions.....	40
References .....	41

## **Table of Figures**

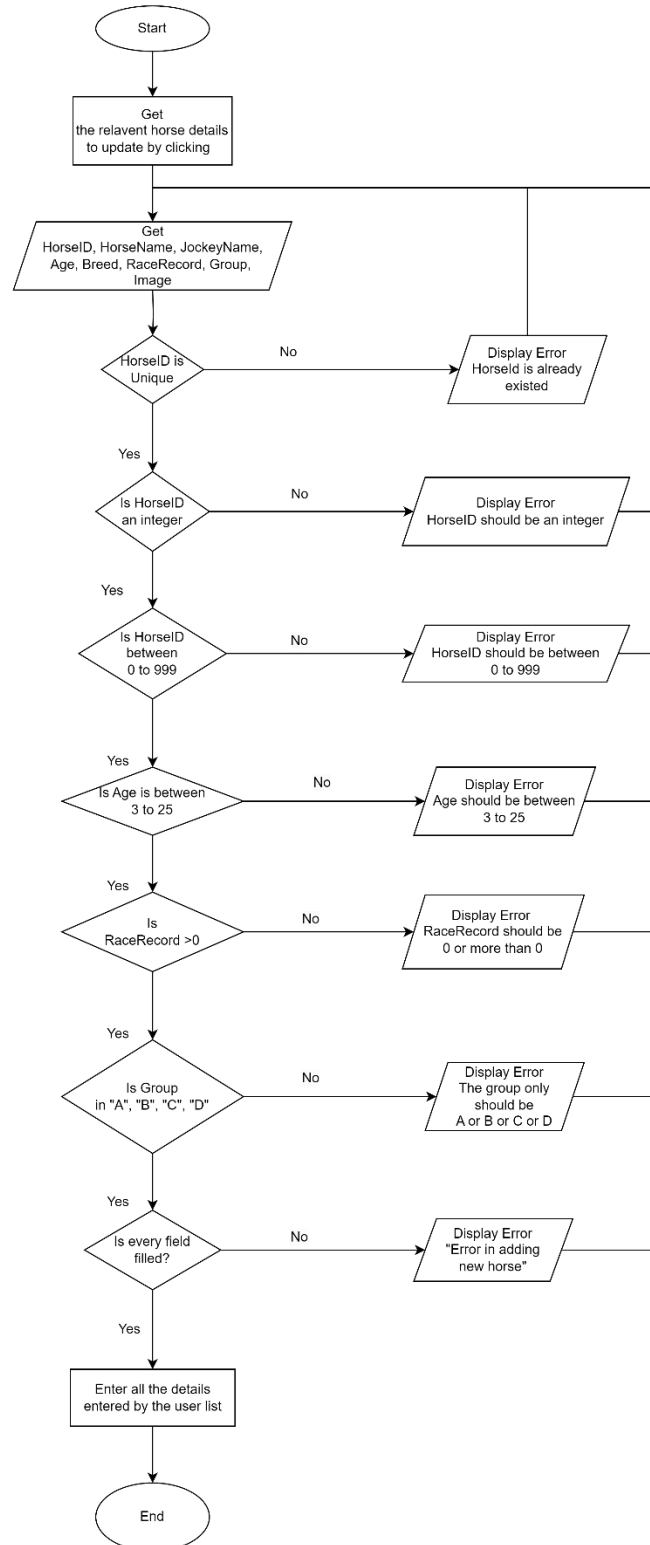
Figure 1 test case 1.....	32
Figure 2 test case 2.....	32
Figure 3 test case 3.....	33
Figure 4 test case 4.....	33
Figure 5 test case 5.....	34
Figure 6 test case 6.....	34
Figure 7 test case 7.....	35
Figure 8 test case 8.....	35
Figure 9 test case 9.....	36
Figure 10 test case 10.....	36
Figure 11 test case 11 .....	37
Figure 12 test case 12.....	37
Figure 13 test case 13.....	38
Figure 14 home page.....	38

# Flowcharts

## Add horse details

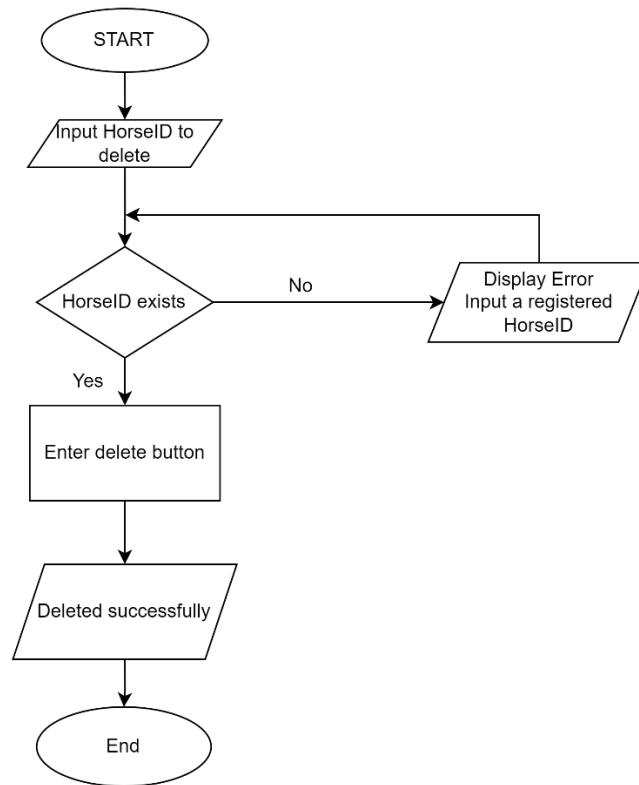


## Update horse details

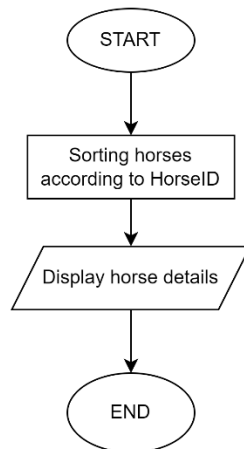




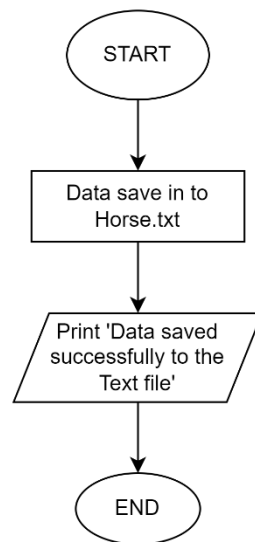
## Delete horse details



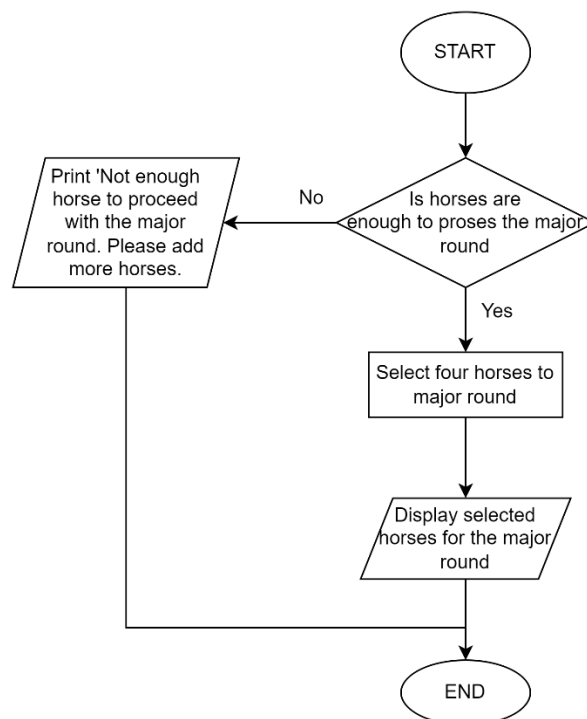
## View horse details



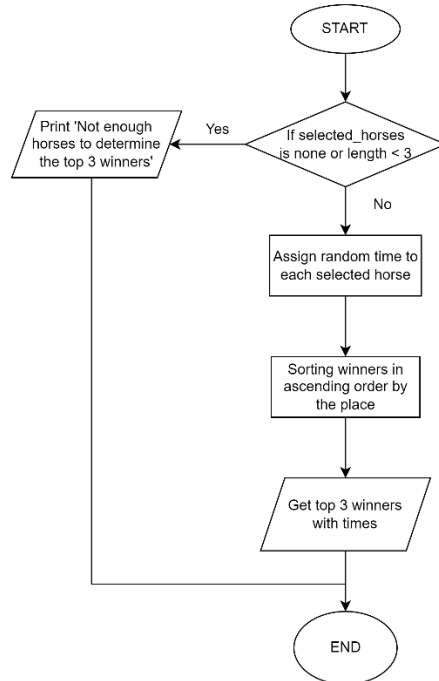
## Save horse details



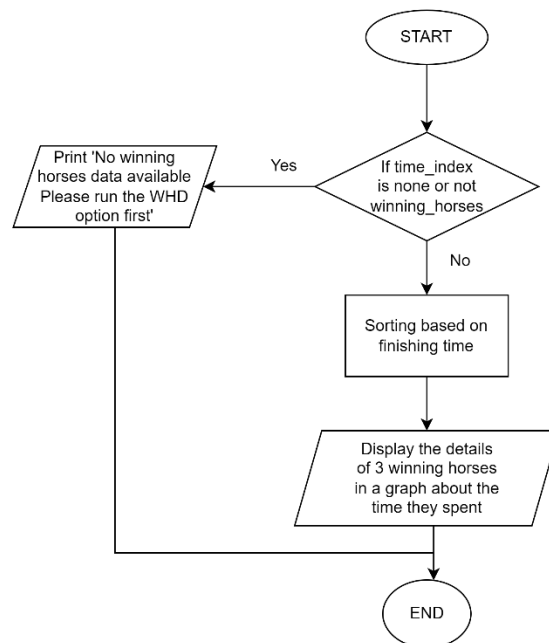
## Simulate for major round



## Winning horse details



## Visualizing horse details



# Introduction to functions with code.

## Add horse details function

### Code

```
@FXML
void AddHorse(ActionEvent actionEvent) {
    HorseData Horse = null;
    try {

        // Get horse information from user input

        int horseID;
        try {
            horseID = Integer.parseInt(addHId.getText());
        } catch (NumberFormatException ex) {
            showMsgAdd("Horse ID must be an integer", false);
            return;
        }

        // Check if the horse ID is in the correct range
        if (horseID < 1 || horseID > 999) {
            showMsgAdd("Horse ID must be between 001 and 999", false);
            return;
        }

        String horseName = addHName.getText();
        // Check if the name is empty or contains non-alphabetic characters
        if (horseName.isEmpty()) {
            showMsgAdd("Please Enter horse's Name", false);
            return;
        } else if (!horseName.matches("[a-zA-Z ]+")) {
            showMsgAdd("The name field can only contain alphabetic
characters", false);
            return;
        }

        String jockeyName = addJName.getText();
        // Check if the name is empty or contains non-alphabetic characters
        if (jockeyName.isEmpty()) {
            showMsgAdd("Please Enter jockey's Name", false);
            return;
        } else if (!jockeyName.matches("[a-zA-Z ]+")) {
            showMsgAdd("The name field can only contain alphabetic
characters", false);
            return;
        }

        int horseAge = Integer.parseInt(addJAge.getText());
```

```

// Check if the horse age is within a valid range
if (horseAge < 3 || horseAge > 25){
    showMsgAdd("Horses must be between 3 and 25 years old", false);
    return;
}

String horseBreed = addBreed.getText();
// Check if the horse's breed is empty
if (horseBreed.isEmpty()) {
    showMsgAdd("Please Enter horse's breed", false);
    return;
}

int raceRecord = Integer.parseInt(addRecord.getText());
// Check if the horse age is within a valid range
if (raceRecord < 0){
    showMsgAdd("Race record cannot be less than 0.", false);
    return;
}

String group = addGroup.getText().toUpperCase();
// Check if the group name is empty
if (group.isEmpty()) {
    showMsgAdd("Please Enter a group name", false);
    return;
}
// Check if the team name is one of A, B, C, or D
if (!group.matches("[ABCD]")) {
    showMsgAdd("Group name must be one of: A, B, C, or D", false);
    return;
}

Image horsePicture = horseImage;
// Check if the horse picture URL is empty
if (horsePicture == null) {
    showMsgAdd("Please insert the horse picture", false);
    return;
}

// Create a new horse data object with the user input
Horse = new HorseData(horseID, horseName, jockeyName, horseAge,
horseBreed, raceRecord, group, horsePicture);

horseImage = null;
} catch (NumberFormatException t) {
    // Catch any NumberFormatException that may occur
    showMsgAdd("Invalid Horse", false);
}

if (Horse != null) {
    for (int i = 0; i < myHorse.size(); i++) {
        // Check if the horse ID already exists in the list
        if (myHorse.get(i).getHorseID() == Horse.getHorseID()) {
            showMsgAdd("Horse with ID " + Horse.getHorseID() + " Already
Exists", false);
            return;

```

```

    }
}
// Add the new horse to the list of horses
myHorse.add(new HorseData(Horse.getHorseID(), Horse.getHorseName(),
Horse.getJockeyName(), Horse.getHorseAge(), Horse.getHorseBreed(),
Horse.getRaceRecord(), Horse.getGroup(), Horse.getHorsePicture()));
showMsgAdd("Horse with ID " + Horse.getHorseID() + " Added
Successfully", true);
// Clear the input fields
addHId.setText("");
addHName.setText("");
addJName.setText("");
addJAge.setText("");
addBreed.setText("");
addRecord.setText("");
addGroup.setText("");
} else {
    showMsgAdd("Error in adding new horse", false);
}
}
}

```

### **Code Explanation**

To add a new horse to a list of horses, use the JavaFX event handler `AddHorse` function. Obtaining the horse data from the user input areas is where the function begins. After that, it verifies every input field to make sure the data is in the right range and format. It verifies that the horse ID, name, jockey's name, age, breed, race history, and group all match the necessary requirements. Once the data has been verified, the verified user input is used to construct a new {HorseData} object. Once the `HorseData` object has been successfully constructed, it verifies that the horse ID is already present in the list. It displays an error message if it does. In the event if not, a success message is shown and the new horse data is added to the list. The input fields are finally cleared. In summary, the function creates and adds a new HorseData object to the list only after verifying that the user input satisfies the necessary requirements.

## Update horse details function

### Code

```
@FXML
void updateHorse(ActionEvent actionEvent) {
    HorseData Hor = null;
    try {

        // Get horse information from user input

        int horseID;
        try {
            horseID = Integer.parseInt(updateHId.getText());
        } catch (NumberFormatException ex) {
            showMsgAdd("Horse ID must be an integer", false);
            return;
        }

        // Check if the horse ID is in the correct range
        if (horseID < 1 || horseID > 999) {
            showMsgAdd("Horse ID must be between 001 and 999", false);
            return;
        }

        String horseName = updateHName.getText();
        // Check if the name is empty or contains non-alphabetic characters
        if (horseName.isEmpty()) {
            showMsgAdd("Please Enter horse's Name", false);
            return;
        } else if (!horseName.matches("[a-zA-Z ]+")) {
            showMsgUpdate("The name field can only contain alphabetic
characters", false);
            return;
        }

        String jockeyName = updateJName.getText();
        // Check if the name is empty or contains non-alphabetic characters
        if (jockeyName.isEmpty()) {
            showMsgAdd("Please Enter jockey's Name", false);
            return;
        } else if (!jockeyName.matches("[a-zA-Z ]+")) {
            showMsgUpdate("The name field can only contain alphabetic
characters", false);
            return;
        }

        int horseAge = Integer.parseInt(updateJAge.getText());
        // Check if the horse age is within a valid range
        if (horseAge < 3 || horseAge > 25){
            showMsgUpdate("Horses must be between 3 and 25 years old",
false);
            return;
        }
    }
}
```

```

String horseBreed = updateBreed.getText();
// Check if the horse's breed is empty
if (horseBreed.isEmpty()) {
    showMsgUpdate("Please Enter horse's breed", false);
    return;
}

int raceRecord = Integer.parseInt(updateRecord.getText());
// Check if the horse age is within a valid range
if (raceRecord < 0) {
    showMsgUpdate("Race record cannot be less than 0.", false);
    return;
}

String group = updateGroup.getText().toUpperCase();
// Check if the group name is empty
if (group.isEmpty()) {
    showMsgUpdate("Please Enter a group name", false);
    return;
}
// Check if the group name is one of A, B, C, or D
if (!group.matches("[ABCD]")) {
    showMsgUpdate("Group name must be one of: A, B, C, or D", false);
    return;
}

Image horsePicture = updatehorse.getHorsePicture();

Hor = new HorseData(horseID, horseName, jockeyName, horseAge,
horseBreed, raceRecord, group, horsePicture);

horseImage = null;

} catch (NumberFormatException t) {
    // Catch any NumberFormatException that may occur
    showMsgAdd("Please enter valid input", false);
}

if (Hor != null) {
    // Check for duplicate horse ID
    int selectedHorseIndex =
updateTable.getSelectionModel().getFocusedIndex();
    for (int i = 0; i < updateTable.getItems().size(); i++) {
        if (i != selectedHorseIndex) {
            HorseData horseDetails = updateTable.getItems().get(i);
            if (horseDetails.getHorseID() == Hor.getHorseID()) {
                showMsgUpdate("Horse with ID " + Hor.getHorseID() + "
already exists", false);
                return;
            }
        }
    }
}
updateHId.clear();
updateHName.clear();
updateJName.clear();
updateJAge.clear();

```



```

        updateBreed.clear();
        updateRecord.clear();
        updateGroup.clear();

        // Update the horse data
        if (selectedHorseIndex >= 0 && selectedHorseIndex < myHorse.size()) {
            myHorse.set(selectedHorseIndex, Hor);
            showMsgUpdate("Horse with ID " + Hor.getHorseID() + " updated
successfully", true);
            // Clear the input fields
            updateHId.setText("");
            updateHName.setText("");
            updateJName.setText("");
            updateJAge.setText("");
            updateBreed.setText("");
            updateRecord.setText("");
            updateGroup.setText("");
        } else {
            showMsgUpdate("Invalid index for updating horse", false);
        }
    } else {
        showMsgUpdate("Error in updating horse", false);
    }
}

@FXML
public void fillFields(MouseEvent event) {
    //Get the selection from table and fill that data to text fields.
    HorseData selected = updateTable.getSelectionModel().getSelectedItem();
    if (selected != null) {
        updateHId.setText(selected.getHorseID().toString());
        updateHName.setText(selected.getHorseName());
        updateJAge.setText(selected.getHorseAge().toString());
        updateJName.setText(selected.getJockeyName());
        updateBreed.setText(selected.getHorseBreed());
        updateRecord.setText(selected.getRaceRecord().toString());
        updateGroup.setText(selected.getGroup());
    } else {
        updateHId.clear();
        updateHName.clear();
        updateJAge.clear();
        updateJName.clear();
        updateBreed.clear();
        updateRecord.clear();
        updateGroup.clear();
    }
}

```

## **Code Explanation**

An event handler called ``updateHorse`` is in charge of changing a horse's details within a JavaFX application. The horse's ID, name, jockey name, age, breed, racing record, and group are all retrieved via user input. After that, the function checks each input field to make sure the data is in the correct range and format. If validation is unsuccessful, error messages are displayed. Following validation of the data, the validated user input is used to generate a new `{HorseData}` object. The function changes the horse data in the `{myHorse}` list and shows a success message after checking for duplicate horse IDs. If no duplicates are discovered. In addition, the function handles error messaging appropriately if there are problems with the input or updating process. A The ``fillFields`` method is probably connected to mouse events like clicking, hovering, or dragging in the UI. It gets the chosen item from a table called `{updateTable}` when it is invoked. Selecting an item allows the user to interact and change the selected ``HorseData`` object by extracting its properties and filling the associated text fields with the data. To ensure a clean interface for fresh data entering or no selection, all text fields are cleared if no item is selected. This feature makes it easier for users to examine and edit specific horse data within the program, which improves user experience.

## Delete horse details function

### The code

```
@FXML
void deleteHorse(ActionEvent actionEvent){
    if (myHorse.isEmpty()) {
        showMsgDelete("Driver list is empty", false);
        return;
    }
    HorseData selected = deleteTable.getSelectionModel().getSelectedItem();
    myHorse.remove(selected);
    deleteTable.refresh();
    showMsgDelete("Deleted Successfully", true);
}

@FXML
void onDltSearch(ActionEvent actionEvent)
{
    if(!dltID.getText().isEmpty())
    {
        int id = Integer.parseInt(dltID.getText());
        for (HorseData horse : myHorse) {
            if (horse.getHorseID().equals(id)) {
                deleteHorse.add(horse);
            }
        }
        deleteTable.setItems(deleteHorse);
    }
}
```

### Code explanation

A JavaFX event handler called `deleteHorse` removes a particular horse from the `myHorse` list. It first determines whether the list is empty, and if it is, it shows an error message stating such. The selected horse is retrieved from the `deleteTable`, removed from the `myHorse` list, the `deleteTable` is refreshed, and a success message confirming the successful deletion is displayed if the list is not empty. In response to an event, most often a search action, the `onDltSearch` function gets an ID from the `dltID` input field. Iterating through the {myHorse} list, it looks for horses with a matching ID if the input is not empty. The `deleteTable` is updated to show the matching horses when a match is detected, and the matched horses are added to the `deleteHorse` list. Users can use this feature to look for and view horses based on their ID in case they need to delete them.

## View horse details method

### The code

```
@FXML
void vrl (ActionEvent actionEvent){
    if (myHorse.size() < 2) {
        showMsgVrl("Please enter atleast 2 Race details", false);
    }else {
        ArrayList<HorseData> horses = new ArrayList<>();
        ArrayList<Integer> id = new ArrayList<>();
        for(HorseData horse:myHorse)
        {
            horses.add(horse);
            id.add(horse.getHorseID());
        }
        ArrayList<HorseData> horses_new = new ArrayList<>();
        ArrayList<Integer> id_new = new ArrayList<>();
        int size = id.size();

        try {
            for (int j = 0; j < size; j++) {
                int min = Collections.min(id);
                int min_index = id.indexOf(min);
                horses_new.add(horses.get(min_index));
                id_new.add(id.get(min_index));
                horses.remove(min_index);
                id.remove(min_index);
            }
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }

        ObservableList<HorseData> horses1 =
FXCollections.observableArrayList(horses_new);

        BreedSortTable.setItems(horses1);
    }
}
```

### **Code explanation**

In a JavaFX application, the `vrl` method acts as an event handler, reacting to an action event (such as a button click). It initially determines whether there are fewer than two elements in the `myHorse` list. If so, the user is prompted with a notice asking for at least two race information; if not, the horse data is sorted. Two lists, `horses` and `id`, are initialized by the method to store the horse data and the corresponding IDs that are taken from the `myHorse` list. The sorted horse data and IDs are then stored in two new lists, `horses_new` and `id_new`. It finds the minimal ID by iterating over the IDs, extracts the matching horse data, and adds the horses to the new lists while deleting the old ones using a selection sort technique. Lastly, it uses the sorted horse data to update the items in the `BreedSortTable` JavaFX table, improving the user experience by presenting the data in a logical order. Although the table is titled `breedSortTable`, the `horseID` is used to sort it.

## Save horse details function

### The code

```
@FXML
void saveDataToFile(ActionEvent actionEvent) {
    File file = new File("D:\\C\\Desktop\\Uni Module\\Year 1\\Programming
Fundamentals\\Java\\Java
CW\\203092_GihangaSandothmi\\javafx2\\javafx2\\src\\main\\resources\\com\\
\\example\\javafx2\\horse.txt");
    if (myHorse.isEmpty()) {
        showMsgSave("Horse list is empty", false);
    } else {
        try {
            // Create a PrintWriter object to write the data to the chosen
file
            PrintWriter save = new PrintWriter(file);
            // Iterate over the list of drivers and write each driver's data
to the file
            for (HorseData horse : myHorse) {
                save.write(horse.getHorseID() + "," + horse.getHorseName() +
",," + horse.getHorseAge() + "," + horse.getJockeyName() + "," +
horse.getGroup()+",,"+horse.getRaceRecord()+",,"+horse.getHorseBreed()+",,"+hors
e.getHorsePicture() + "\n");
            }
            // Close the PrintWriter object and show a success message if
writing to the file is successful. otherwise, show an error message
            save.close();
            showMsgSave("Successfully wrote to the file.", true);
        } catch (IOException e) {
            showMsgSave("An error occurred.", false);
            e.printStackTrace();
        }
    }
}
```

### **Code explanation**

The JavaFX event handler `saveDataToFile` is in charge of storing horse data in a text file. The `myHorse` list is checked to see if it is empty, after which a `File` object representing the destination file is initialized. An error message stating that the horse list is empty is displayed if the list is empty. The method iterates through the `myHorse` list, writing each horse's data in a comma-separated manner to the file, if the list is not empty. It does this by creating a `PrintWriter` object. It shows a success message when writing is successful, and an error message and stack trace are printed when writing is unsuccessful. All in all, the function lets the user save the horse data to a designated file place and gives relevant feedback depending on how things work out.

## Simulating data for the major round function

### The code

```
@FXML
public void simulateRandomRace(ActionEvent actionEvent) {
    // Check if there are atleast 3 racers entered
    int limit=1;

    if (myHorse.size() < limit*4) {
        showMsgRandomRace("Please enter atleast 4 Horse details", false);
    } else {

        addBtn.setDisable(true);
        updateBtn.setDisable(true);
        deleteBtn.setDisable(true);

        // Create a PrintWriter to write the random race details to a file
        try {

            majorRound.clear();
            majorRoundTable.refresh();
            ArrayList<HorseData> groupA= new ArrayList<>();
            ArrayList<HorseData> groupB= new ArrayList<>();
            ArrayList<HorseData> groupC= new ArrayList<>();
            ArrayList<HorseData> groupD= new ArrayList<>();

            for (int i = 0; i < myHorse.size(); i++)
            {
                HorseData horse = myHorse.get(i);
                switch (horse.getGroup()) {
                    case "A" -> groupA.add(horse);
                    case "B" -> groupB.add(horse);
                    case "C" -> groupC.add(horse);
                    case "D" -> groupD.add(horse);
                }
            }
            Random rand = new Random();

            for(int i=0;i<limit;i++) {
                int a = rand.nextInt(0, groupA.size());
                int b = rand.nextInt(0, groupB.size());
                int c = rand.nextInt(0, groupC.size());
                int d = rand.nextInt(0, groupD.size());

                HorseData horseA = groupA.get(a);
                HorseData horseB = groupB.get(b);
                HorseData horseC = groupC.get(c);
                HorseData horseD = groupD.get(d);

                raceHorses.clear();

                raceHorses.add(horseA);
```



```

        raceHorses.add(horseB);
        raceHorses.add(horseC);
        raceHorses.add(horseD);
    }

    ArrayList<LocalTime> raceTimes = new ArrayList<>();

    for (int i=0;i<limit*4;i++)
    {

        int time = rand.nextInt(10,90);

        int remainder = time%60;
        int min = time/60;
        int sec = remainder;

        String timeString = "0"+min+": "+sec;
        LocalTime t = LocalTime.of(0,min,sec);
        raceTimes.add(t);
    }
    ArrayList<HorseData> horses = new ArrayList<>();
    ArrayList<LocalTime> times = new ArrayList<>();

    for(int i=0;i<limit*4;i++)
    {

        horseDataList.add(new
Object[] {raceHorses.get(i),raceTimes.get(i)});
    }
    try

    {
        for (int i = 0; i < limit * 4; i++) {

            for (int j = 0; j < limit * 4; j++) {
                int max_time_index =
raceTimes.indexOf(max_time(raceTimes));
                horses.add(raceHorses.get(max_time_index));
                times.add(raceTimes.get(max_time_index));
                raceHorses.remove(max_time_index);
                raceTimes.remove(max_time_index);
            }
        }
    }
    catch (IndexOutOfBoundsException e)
    {
        ;
    }

    majorRound.addAll(horses);
    majorRoundTable.setItems(majorRound);

```

```
    }  
    catch (Exception e) {  
        throw new RuntimeException(e);  
    }  
}  
}
```

### **Code explanation**

In a JavaFX application, the `simulateRandomRace` method is an event handler that is called upon by an action event, such as a button click. It first verifies that at least four horse racers have been entered; if not, it shows a message asking the user to provide further information. It disables some buttons to stop users from entering the race simulation after it meets the requirements. Then, in order to classify horses according to their designated groupings, it initializes four groups (`groupA`, `groupB`, `groupC`, and `groupD`). It chooses one horse to compete in the race from each group using a random number generator. Each horse within a given range has a random race time created for them, and the horses and their times are added to a list. The horses are then rearranged according to their race times using a sorting algorithm, and the user can see the sorted list in a table. During this process, any exceptions are detected and dealt with accordingly.

## Winning horse details function

### The code

```
@FXML
void WinningHorseTable(ActionEvent actionEvent){

    if(!horseDataList.isEmpty())

    {
        ArrayList<HorseData> horses = new ArrayList<>();
        ArrayList<LocalTime> times = new ArrayList<>();
        for(Object[] obj:horseDataList)
        {
            horses.add((HorseData)obj[0]);
            times.add((LocalTime)obj[1]);
        }
        ArrayList<HorseData> horses_new = new ArrayList<>();
        ArrayList<LocalTime> times_new = new ArrayList<>();
        while (!horses.isEmpty())
        {
            for (int i = 0; i < horses.size(); i++)
            {
                LocalTime min_time = max_time(times);
                int min_time_index = times.indexOf(min_time);
                horses_new.add(horses.get(min_time_index));
                times_new.add(times.get(min_time_index));
                horses.remove(min_time_index);
                times.remove(min_time_index);
            }
        }
        for(int i=0;i<3;i++)
        {
            HorseData horse = horses_new.get(i);
            HorsePlace place = new
HorsePlace(i+1,horse.getHorseID(),horse.getHorseName(),horse.getGroup(),horse
.getHorsePicture(),times_new.get(i));
            placement.add(place);
        }
        standingTable.setItems(placement);
    }
}

private LocalTime max_time(ArrayList<LocalTime> time_list)
{
    LocalTime max=time_list.get(0);
    for (int i=0;i<time_list.size();i++)
    {
        if (time_list.get(i).isBefore(max))
        {
            max=time_list.get(i);
        }
    }
}
```

```
}  
    return max;  
}
```

### **Code explanation**

Another event handler in a JavaFX application is the `WinningHorseTable` method, which is probably connected to a user action like clicking a button. The first thing it does is see if the `horseDataList` list is empty. It pulls each horse's data and race time from the list into distinct ArrayLists if there are any horse data entries. The horses are then arranged according to their racing times using a sorting algorithm, guaranteeing that the fastest horse finishes first. The top three winning horses are then represented by instances of `HorsePlace`, which are created using the sorted horse data and added to the {placement} list. Ultimately, the `placement` list appears in a table, most likely displaying the winning horses' standings. The `max\_time` function aids in determining the longest time among a collection of race times. During this process, any exceptions that arise are not explicitly addressed.

## Visualizing winning horse details function

### The code

```
void chart()
{
    xAxis.setLabel("Horse");
    xAxis.setStyle("-fx-text-fill: white; -fx-font-size: 28px;");
    xAxis.setTickLabelRotation(90);
    yAxis.setLabel("Time");
    yAxis.setStyle("-fx-text-fill: white; -fx-font-size: 28px;");

    XYChart.Series<String,Number> series1 = new XYChart.Series<>();
    int count=1;

    for (HorsePlace place:placement)
    {
        LocalTime time = place.getTime();
        int min = time.getMinute();
        int sec = time.getSecond();
        int total = min * 60 + sec;
        series1.getData().add(new XYChart.Data<>(count + " place", total));
        count+=1;
    }
    visualizeChart.getData().add(series1);
}
```

### Code explanation

The `chart` method, probably in a JavaFX application, is in charge of creating and filling a chart. The horse names and race times are represented by configuring the X- and Y-axis labels and styles, respectively. After that, the chart data's empty series is initialized. It iterates over the `placement` list, which is assumed to include information regarding the positioning of horses in a race, using a loop. It extracts the race time, converts it to total seconds, and adds it as a data point to the series along with the matching place number for each `HorsePlace` object in the list. By dynamically creating a graphic that shows the race results in terms of time and horse placement, the approach improves user visualization.

## Horse image method

### The code

```
private Image horseImage;

@FXML
void selectImage(ActionEvent event) {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Select Horse Image");
    fileChooser.getExtensionFilters().addAll(
        new FileChooser.ExtensionFilter("Image Files", "*.png", "*.jpg",
        "*.gif", "*.jpeg")
    );
    File selectedFile = fileChooser.showOpenDialog(new Stage());
    if (selectedFile != null) {
        horseImage = new Image(selectedFile.toURI().toString());
    }
}

@FXML
void selectImage2(ActionEvent event) {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Select Horse Image");
    fileChooser.getExtensionFilters().addAll(
        new FileChooser.ExtensionFilter("Image Files", "*.png", "*.jpg",
        "*.gif")
    );
    File selectedFile = fileChooser.showOpenDialog(new Stage());
    if (selectedFile != null) {
        horseImage = new Image(selectedFile.toURI().toString());
    }
}
```


### Code explanation

The purpose of these two JavaFX event handlers is to make it easier for users to choose horse photos from an application interface. The user can browse and pick an image file using the file chooser box that appears when the `selectImage` or `selectImage2` actions are initiated, usually by clicking a button. The dialog box restricts the files that are shown to only those that have particular image extensions, like PNG, JPG, GIF, and optionally JPEG. Once a file has been chosen and confirmed, its URI is transformed into a `Image` object and set to the `horseImage` variable. This feature improves the application's usability and aesthetic appeal by allowing users to dynamically import and correlate horse photos with appropriate data.

## Test plans and Test cases

### Test Cases

Test Case	Input	Expected output	Actual output	Pass/ fail
1	1, Penny, John, 20, Mustang, 20, A, image	Horse with ID 1 added successfully	Horse with ID 1 added successfully	Pass
2	Sd10, Tim, Sam, 16, Breton, 45, B, image	HorseID should be an integer	HorseID should be an integer	Pass
3	-1, Monny, Tom, 5, Mustang, 15, C, image	HorseID should be between 0 to 999	HorseID should be between 0 to 999	Pass
4	1, Nony, Jemmy, 10, Mustang, 25, C, image	HorseID is already existed	HorseID is already existed	Pass
5	123, Kin, John, 35, Ravan, 45, C, image	HorseID is already existed	Horse age should be between 3 to 25	Pass
6	123, Kin, John, 20, Ravan, 45, F, image	Group name should be A or B or C or D	Group name should be A or B or C or D	Pass
7	12, Penny, John, 20, Mustang, 20, A, image	HorseID 12 successfully updated	HorseID 12 successfully updated	Pass
8	34	Deleted successfully	Deleted successfully	Pass
9	Press Sort by HorseID button	Sorted successfully	Sorted successfully	Pass
10	Press save current data to text file	Data saved in text file successfully	Data saved in text file successfully	Pass
11	Press select for major round	Horses selected successfully	Horses selected successfully	Pass
12	Press sort by place	Horses sorted by place successfully	Horses sorted by place successfully	Pass
13	Press visualizing winning horses	Display winning horses time taken with chart successfully	Display winning horses time taken with chart successfully	Pass



**RAPID RUN**  
"GALLOP TO GLORY!"

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord


JockeyName

Group

Age

Picture


Horse with ID 1 Added Successfully

HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Menu ☰

Exit ➡

Figure 1 test case 1



**RAPID RUN**  
"GALLOP TO GLORY!"

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord


JockeyName

Group

Age

Picture

Horse ID must be an integer


HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Menu ☰

Exit ➡

Figure 2 test case 2





**RAPID RUN**  
"GALLOP TO GLORY!"

Menu ☰

Exit 🚪

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord

JockeyName

Group

Age

Picture

Horse ID must be between 001 and 999


HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Figure 3 test case 3



**RAPID RUN**  
"GALLOP TO GLORY!"

Menu ☰

Exit 🚪

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord

JockeyName

Group

Age

Picture

Horse with ID 1 Already Exists



HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Figure 4 test case 4



**RAPID RUN**  
"GALLOP TO GLORY!"

Menu ☰

Exit 🚪

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord

JockeyName

Group


Age

Picture

Horses must be between 3 and 25 years old

HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Figure 5 test case 5



**RAPID RUN**  
"GALLOP TO GLORY!"

Menu ☰

Exit 🚪

← **Add Horse Details**

HorseID

Breed

HorseName

RaceRecord

JockeyName

Group

Age

Picture

Group name must be one of: A, B, C, or D



HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
1	Penny	John	20	Mustang	20	A	

Figure 6 test case 6



**RAPID RUN**  
"GALLOP TO GLORY!"

← **Update Horse Details**

HorseID

Breed

HorseName

RaceRecord

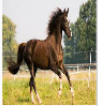
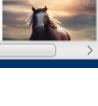
JockeyName

Group

Age

Picture

Horse with ID 12 updated successfully

HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
12	Penny	John	20	Mustang	20	A	
123	Kin	John	20	Ravan	45	B	
3	Ben	Ten	12	Mustang	122	D	
246	Hebi	Boobi	23	Ravan	100	C	

Menu ☰

Exit ➡

Figure 7 test case 7



**RAPID RUN**  
"GALLOP TO GLORY!"

← **Delete Horse Details**

Horse ID

Deleted Successfully

HorseID	HorseName	JockeyName	Age	Breed	RaceRecord	Group	HorsePicture
34	Booooo	Tommy	20	Ravan	12	A	

Menu ☰

Exit ➡

Figure 8 test case 8



Figure 9 test case 9

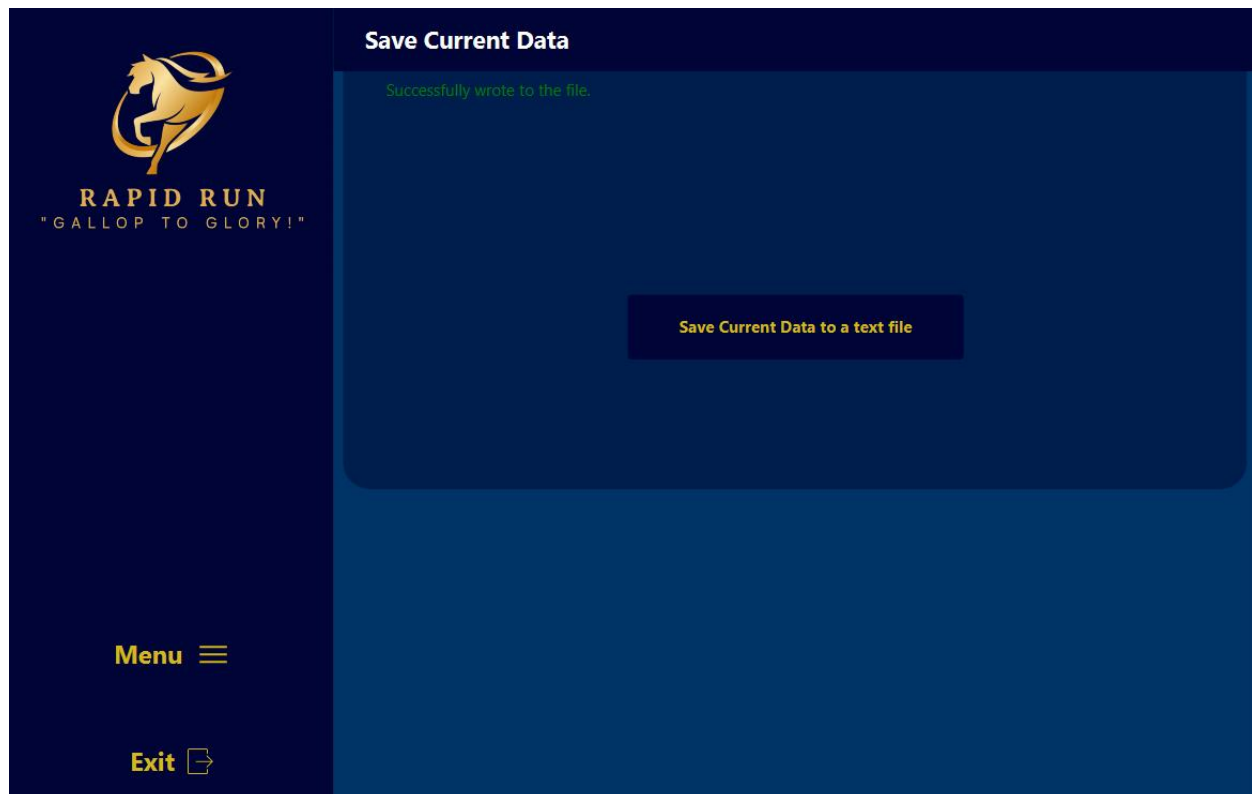


Figure 10 test case 10

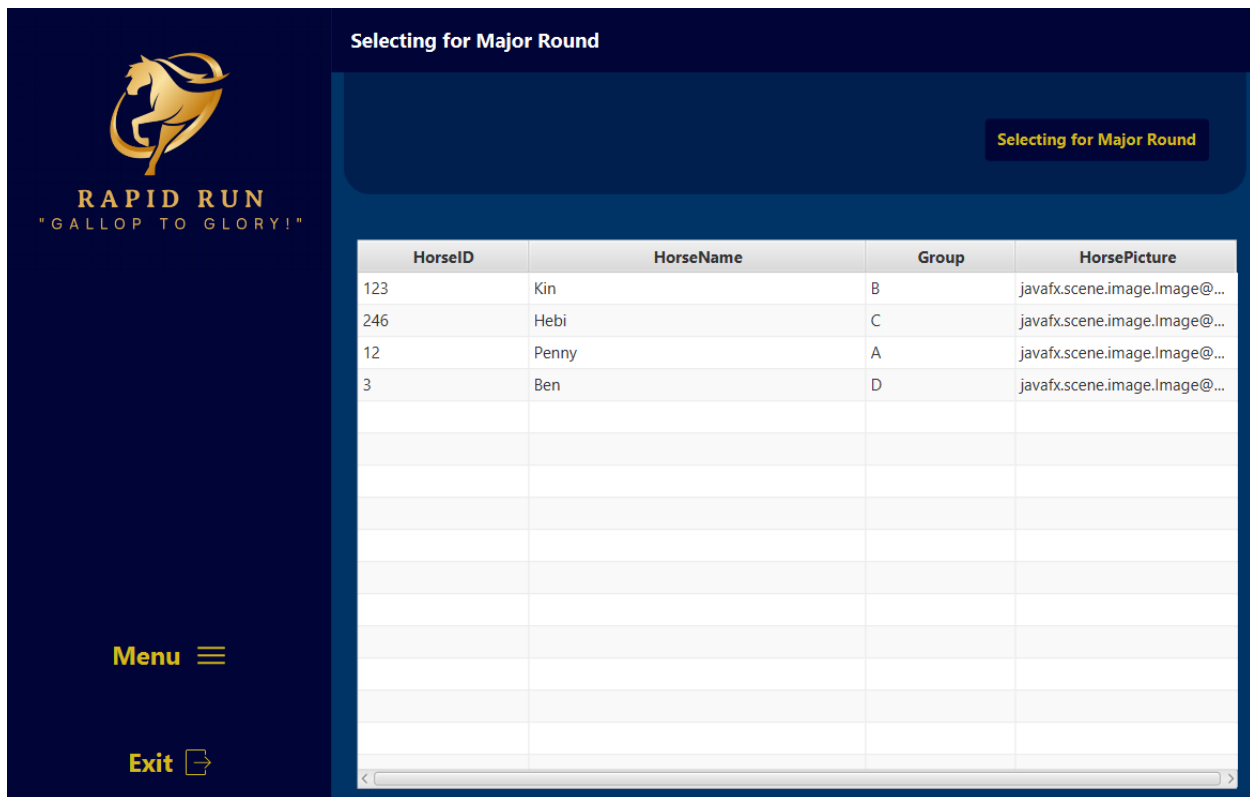


Figure 11 test case 11



Figure 12 test case 12

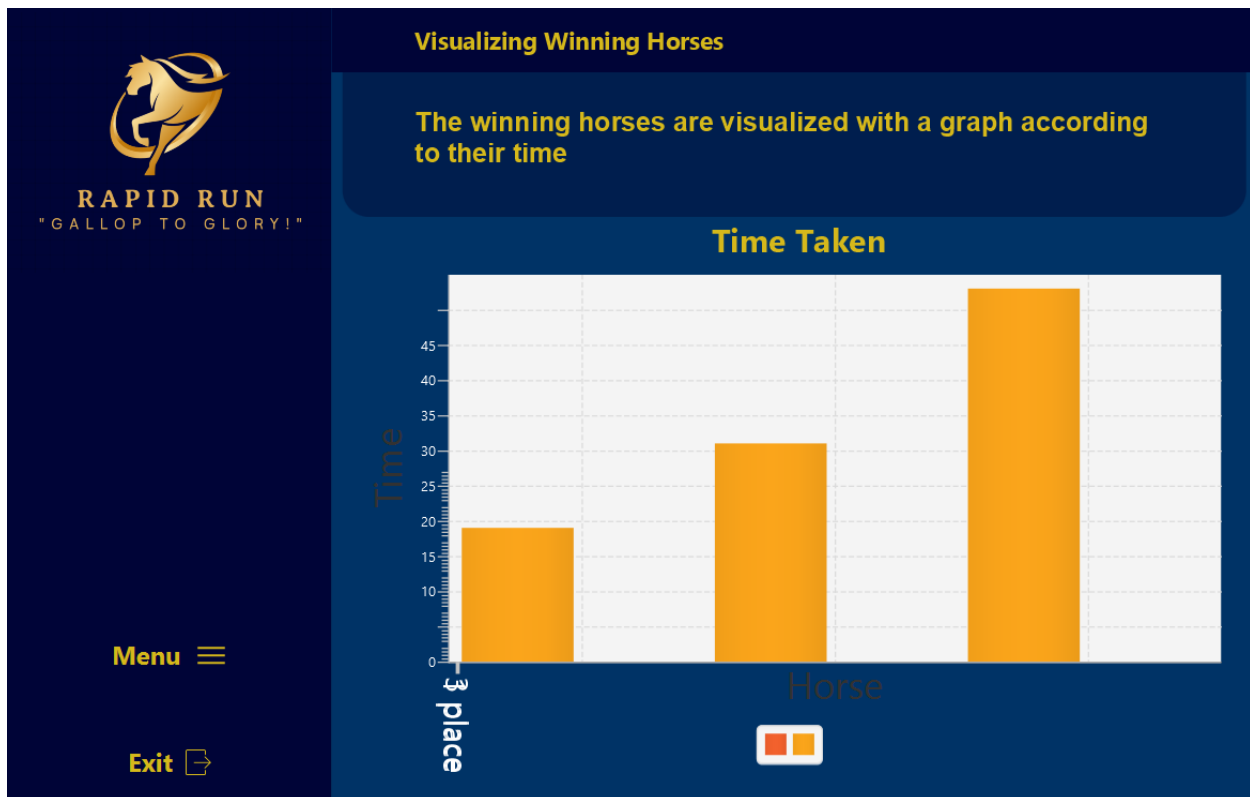


Figure 13 test case 13

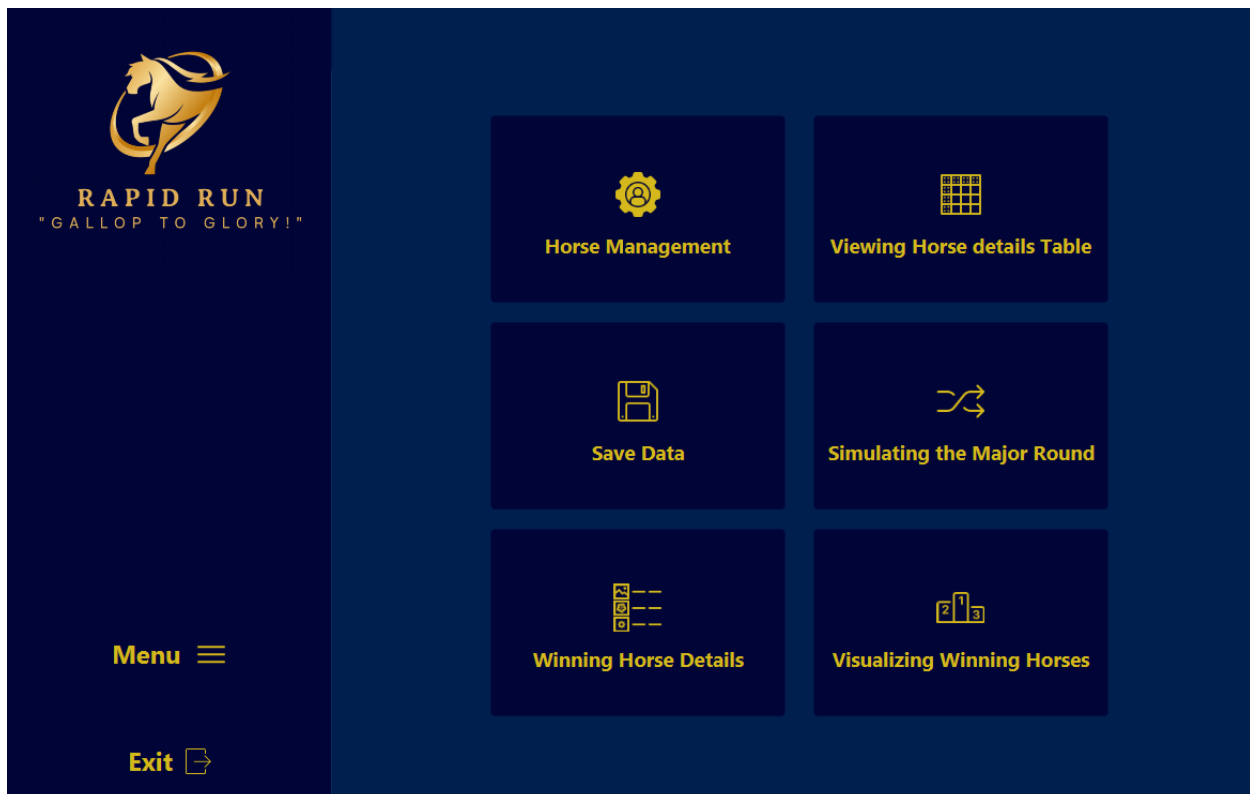


Figure 14 home page

## **Robustness & Maintainability**

Add input validations: Before getting the horse details entered by the user, input validation can be added to ensure that the user has entered valid data. For example, the age field should only accept integer values between 3 and 25. This will prevent invalid data from being entered into the system.

Extract duplicate check: To make a method more maintainable, the logic for the duplication check can be taken out and placed into a different one. This function can determine whether the horseID object already exists in the list by accepting it as an argument.

Enhance error handling: At the moment, when an exception happens, the code shows an error message. On the other hand, handling particular exceptions independently and giving the user more detailed error messages would be preferable.

## **Conclusion & Assumptions**

Through this coursework, we were given the difficult chance to investigate and test several facets and functionalities of languages and programs including Java, JavaFX, and SceneBuilder. With the use of a variety of functions, strategies, and practices, this training enabled us to fully comprehend scenarios and actual problems in the world and coming up with fixes for them. As a result, we will have a greater understanding of the subject and be better equipped to handle any issues that may arise in the real world in the future.

- HorseID must be an integer.
- Horse age must be between 3 to 25.
- When it comes to update the horse image cannot be changed.
- The time that I used for the race is 10s to 90s.



## **References**

- Stack Overflow. (n.d.). java - How do I add objects to an observable array backing list? [online] Available at: <https://stackoverflow.com/questions/29709608/how-do-i-add-objects-to-an-observable-array-backing-list>[Accessed 10 April. 2024].
- www.w3schools.com. (n.d.). Java Read Files. [online] Available at: [https://www.w3schools.com/java/java\\_files\\_read.asp](https://www.w3schools.com/java/java_files_read.asp)
- www.youtube.com. (n.d.). 1. How to create Modern GUI Using IntelliJ JavaFX 16 and SceneBuilder. [online] Available at: <https://www.youtube.com/watch?v=VOiFmZyGAds&t=12s>[Accessed 8 April. 2024].
- Stack Overflow. (n.d.). How to use PrintWriter and File classes in Java? [online] Available at: <https://stackoverflow.com/questions/11496700/how-to-use-printwriter-and-file-classes-in-java>[Accessed 12 April. 2024].
- www.javatpoint.com. (n.d.). Insertion Sort in Java - Javatpoint. [online] Available at: <https://www.javatpoint.com/insertion-sort-in-java>[Accessed 10 April. 2024].