Rapid Race

Documentation

Stephan Dabare

3/30/24

AI and DS

Executive Summary

This report describes the design and development of a Java application called ‘Rapid Race’ for managing horse racing data. The program is made up of numerous components, such as user interfaces for data entry and display, backend logic for data processing, and unit tests to ensure reliability.

The application's key features are as follows:

- A user-friendly interface for entering, updating, and viewing horse details.
- Ability to select random horses for a race, simulate race times, and display race results in a graph structure.
- JavaFX is used to create interactive graphical interfaces.
- Sorting functionalities are done using custom-created algorithms.

The programming process concentrated on creating code that was simple for everyone to read and understand. This greatly simplifies future codebase maintenance. Throughout the process, the code base followed the SOLID principles, a set of recommendations that promotes well-structured, adaptable, and expandable code.

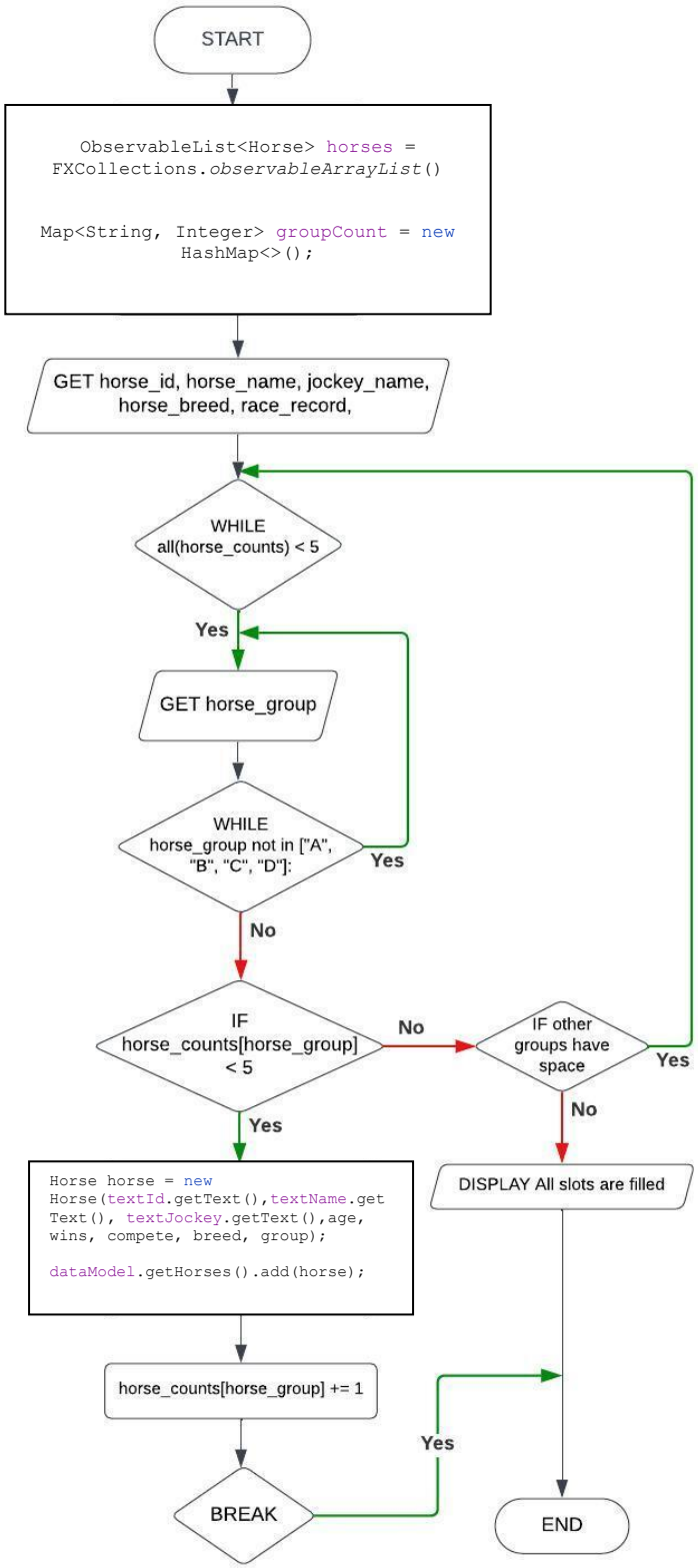
In conclusion, the Java application created for managing horse racing data is a reliable, maintainable, and efficient solution that fits the project's criteria.

1. Contents

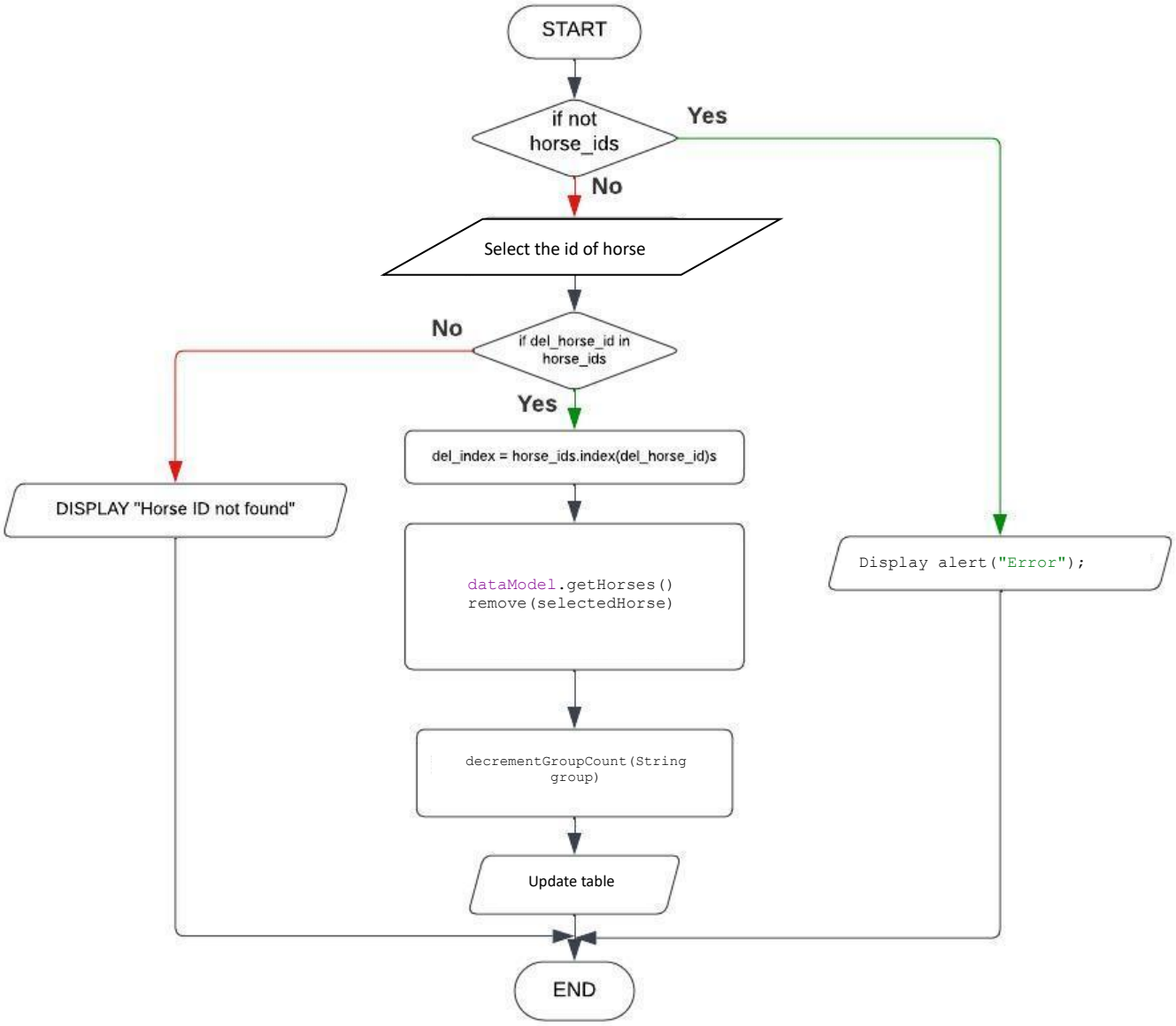
2. Flow Charts 3
3. Introduction to functions with code 13
   I. Adding Horse Details 13
   II. Deleting Horse Details 15
   III. Update Horse Details 16
   IV. View the Registered Horses' Details Table 17
   V. Save the Horse Details into the Text File 19
   VI. Selecting Four Horses Randomly 20
   VII. Select Winning Horses 21
   VIII. Visualize Winning Horses 22
   IX. GUI fxml codes 24
4. J-units, Test plan, and test cases 28
   I. J-units 28
   II. Test Plan and Test Cases 31
5. Code Quality, Robustness and Maintainability 38
   I. Readability 38
   II. Reliability and correctness (Robustness) 38
   III. Efficiency and performance 38
   IV. Maintainability 38
   V. Usage of OOP Concepts 39
   VI. SOLID Principles 40
6. Conclusion 41
7. References 42
8. Appendices 43

2. Flow Charts

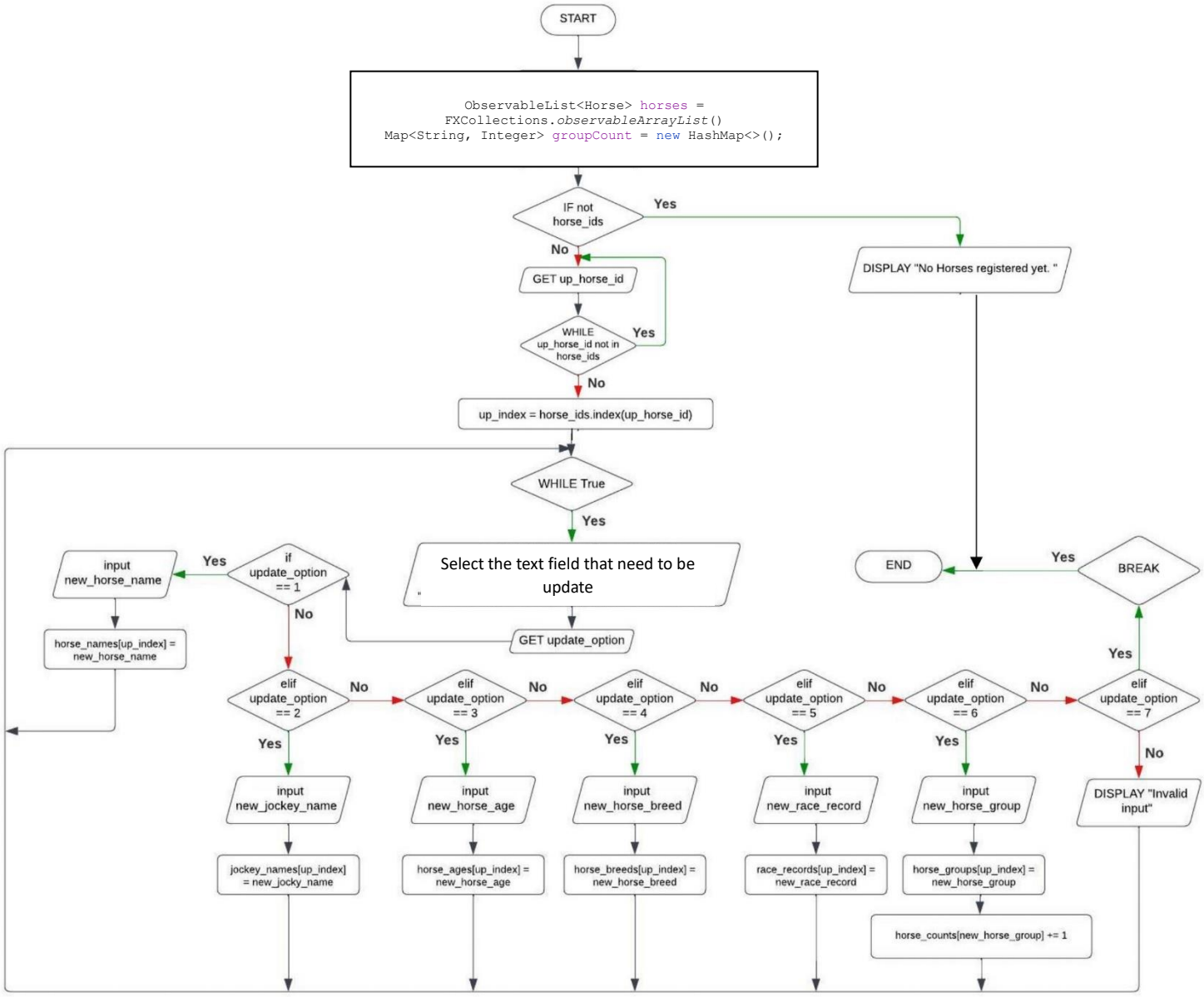
Adding Horse Details



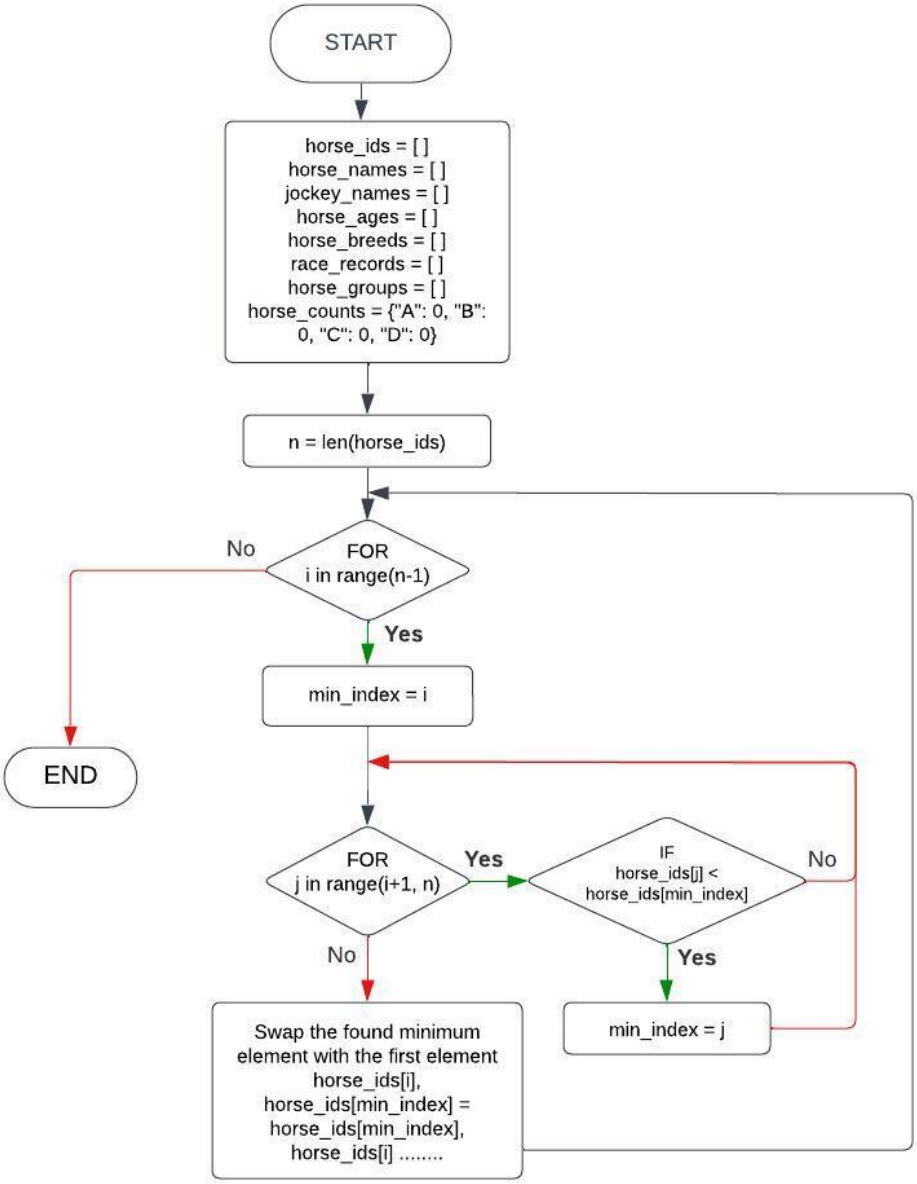
Deleting Horse Details



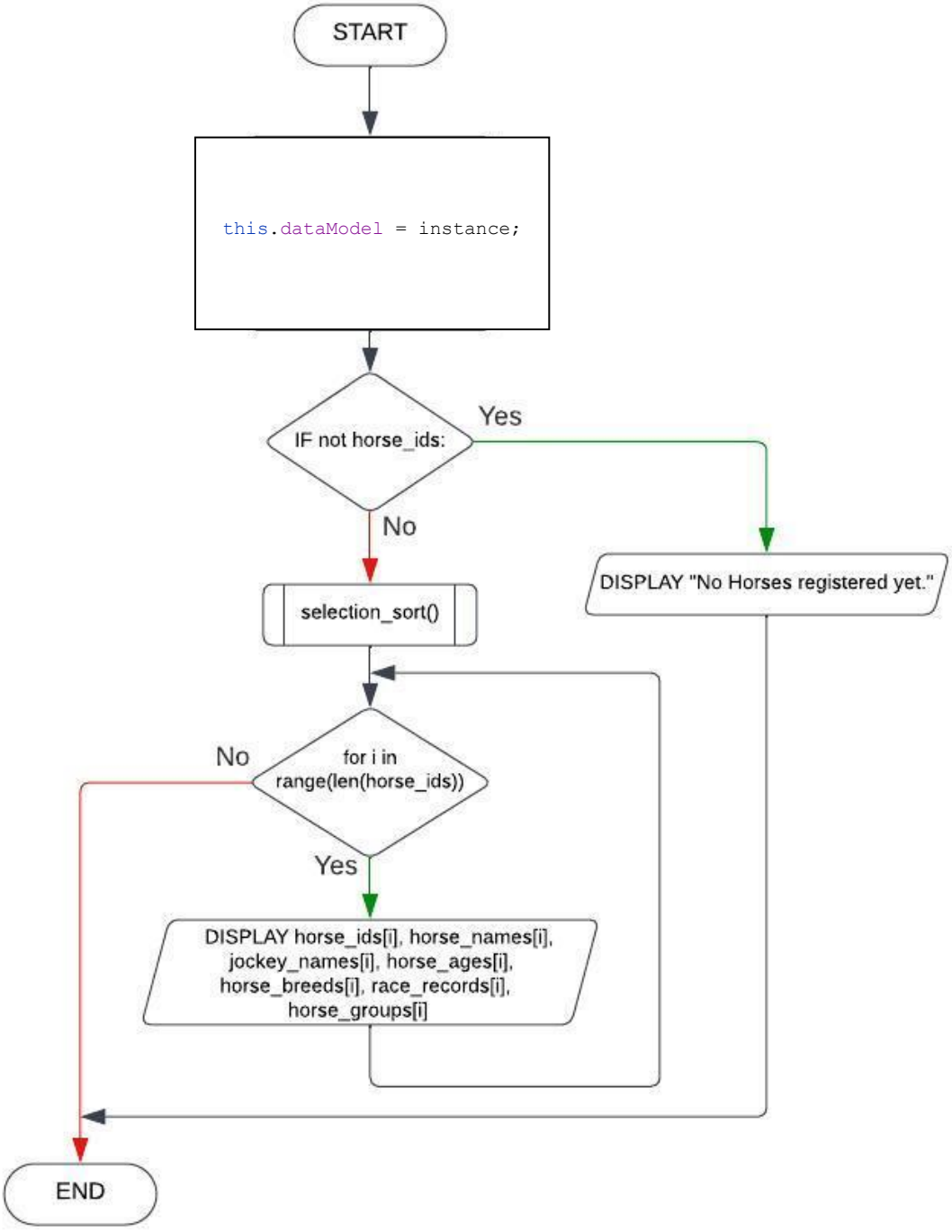
Update Horse Details



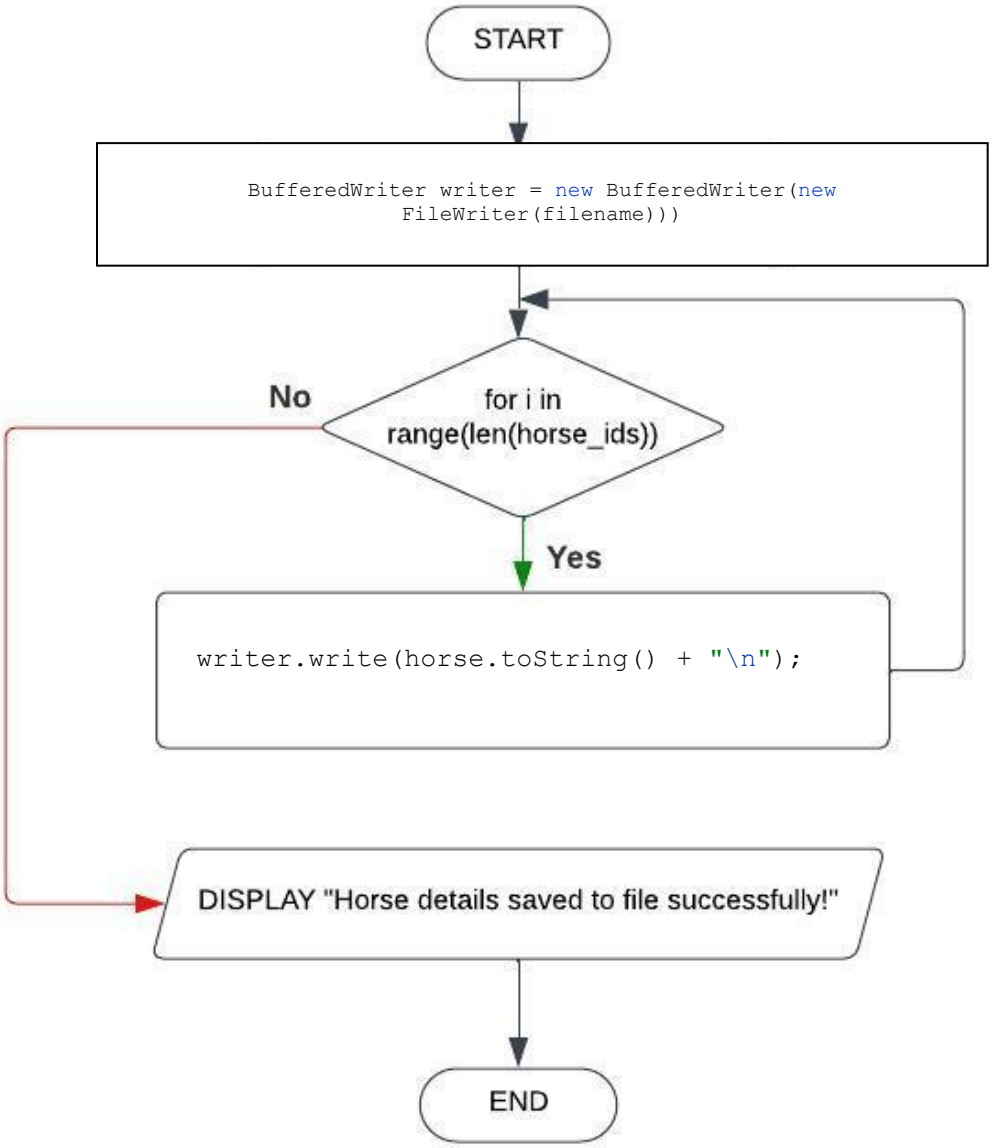
Sorting Algorithm Flowchart



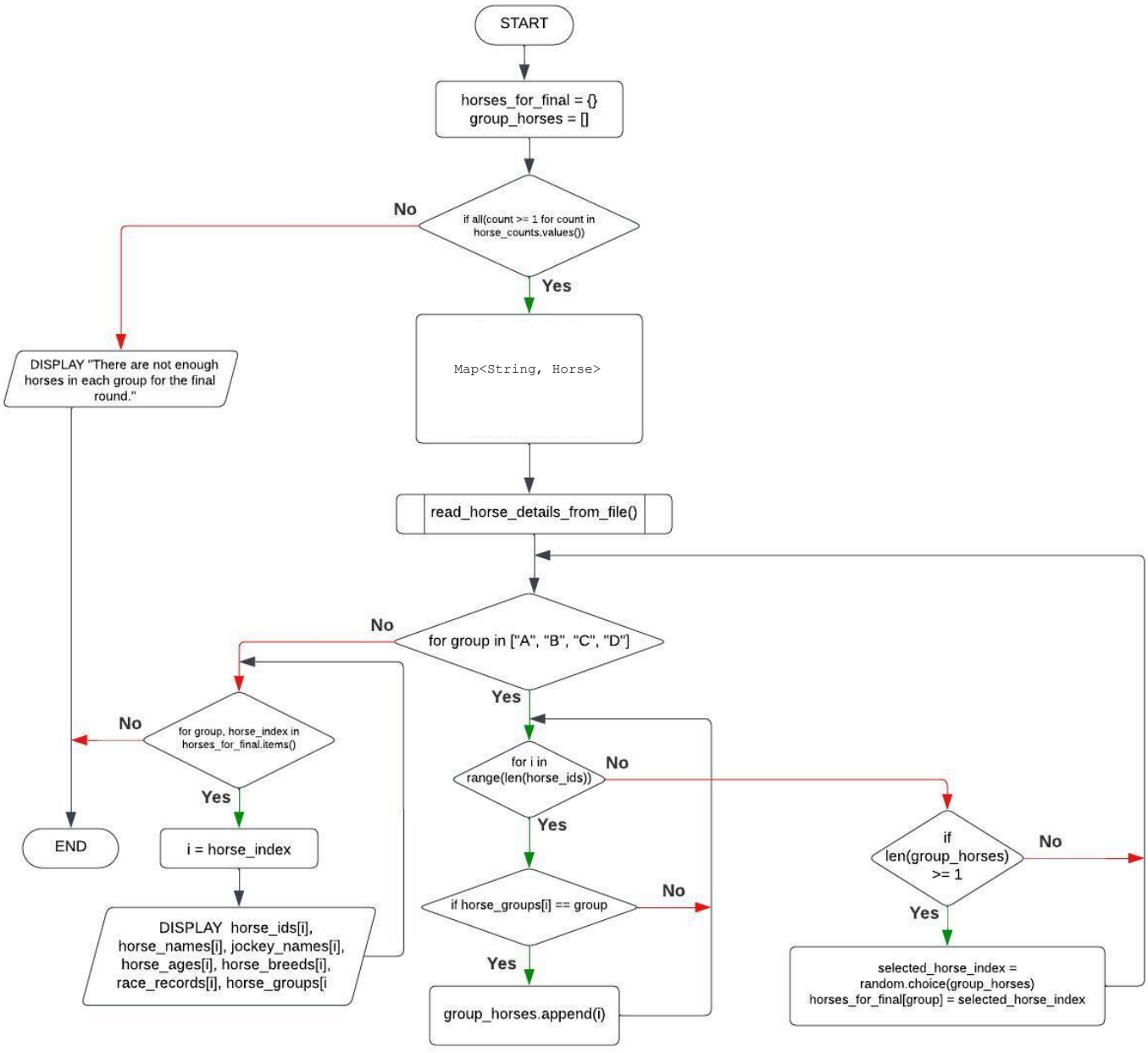
View the Registered Horses' Details Table



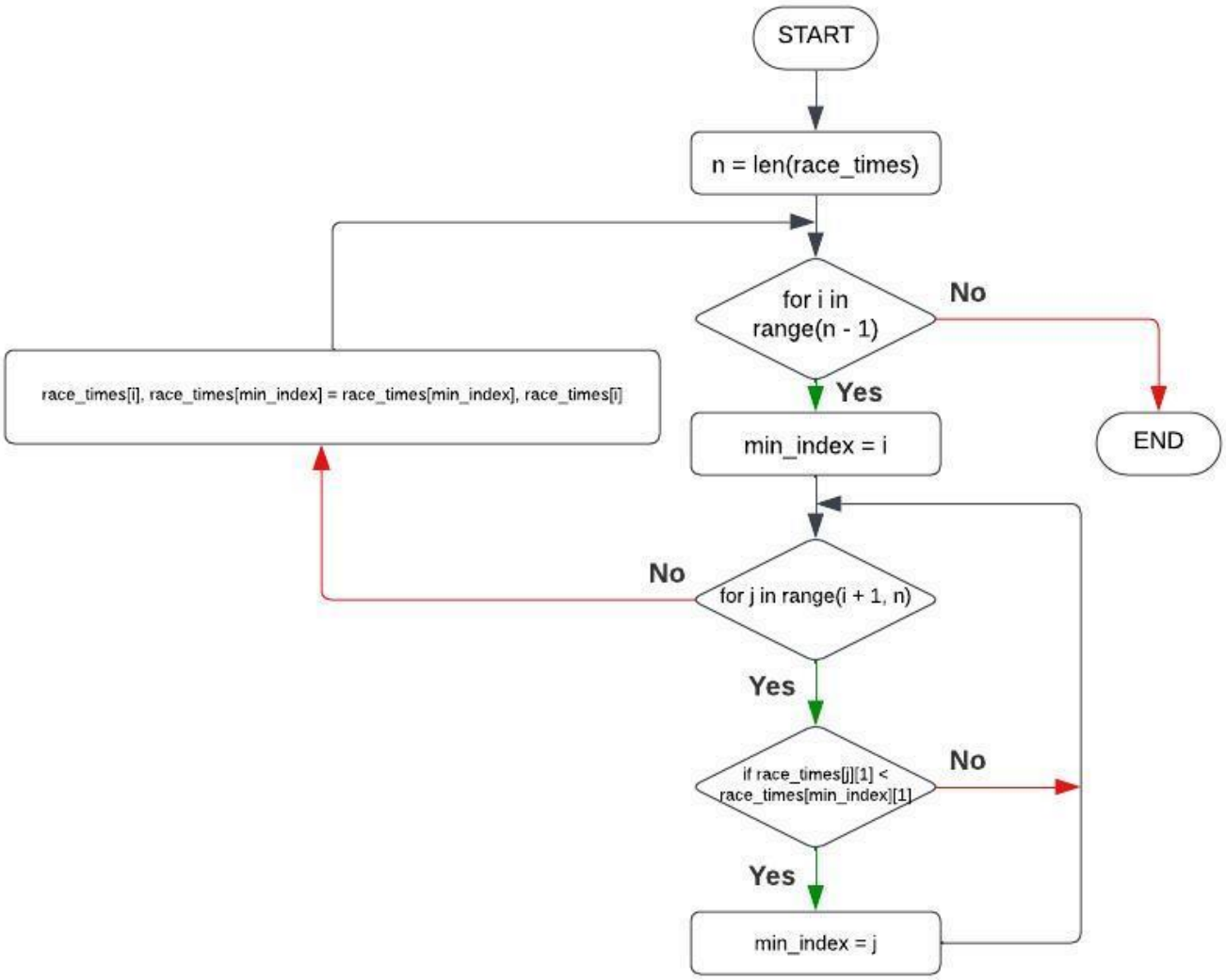
Save the Horse Details



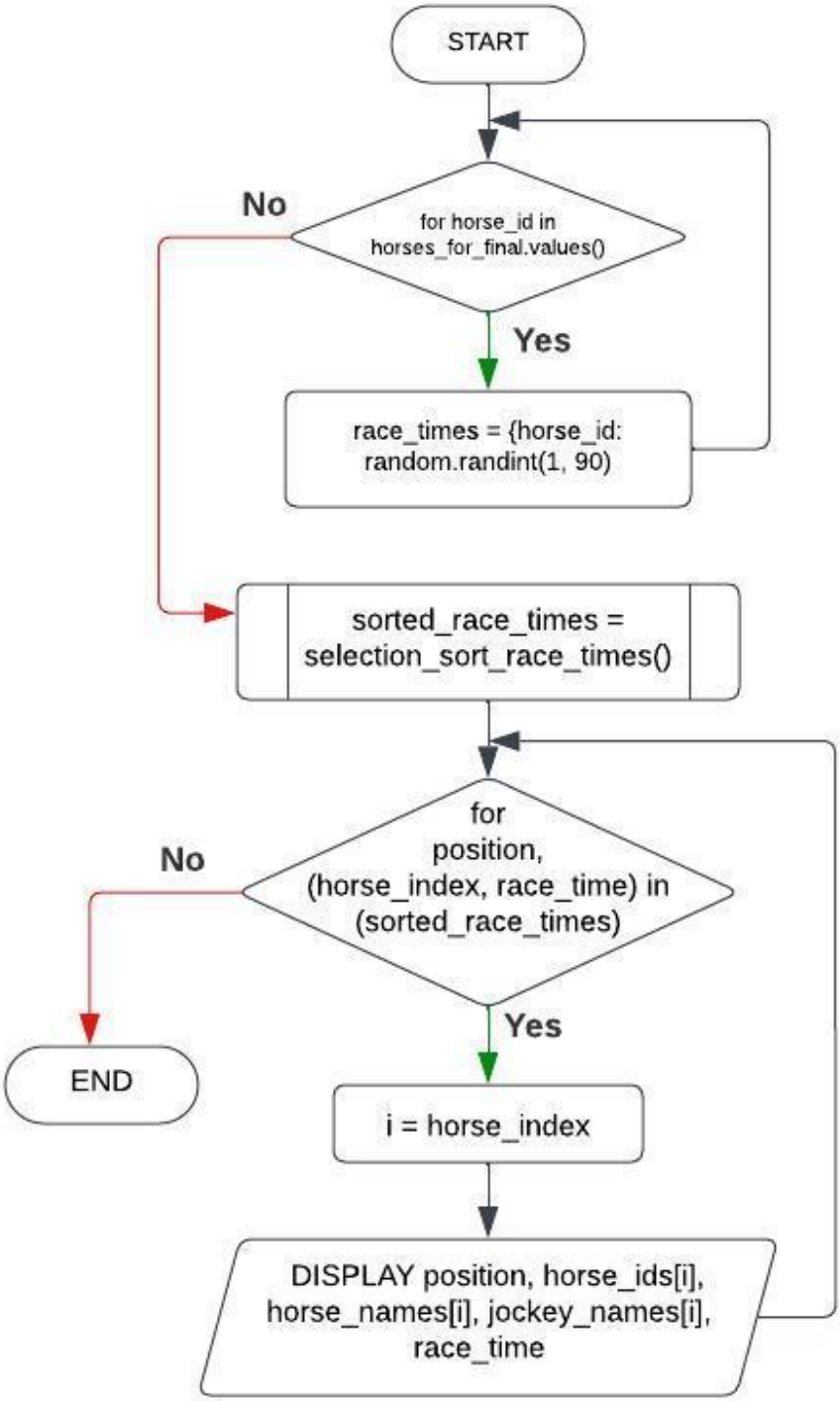
Selecting Four Horses Randomly



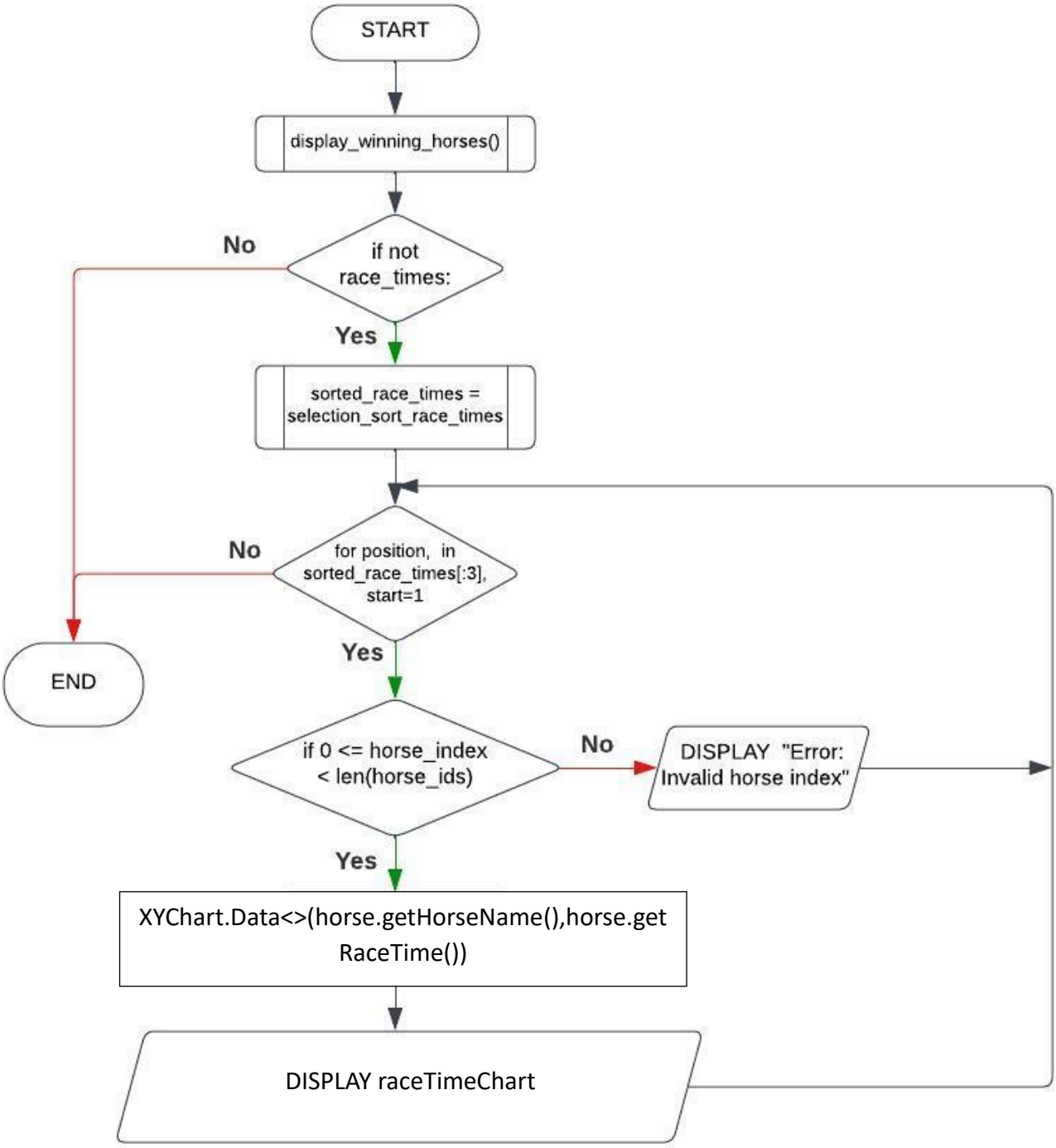
Time Sorting Algorithm Flowchart



Select Winning Horses



Visualize Winning Horses



3. Introduction to functions with code

I. Adding Horse Details

This function allows users to enter information about horses into the system. It determines the entered horse ID is taken before and valid. (Positive integers only) If the ID is valid, the user can enter information about the horses. Users can use text fields to enter information about horses.

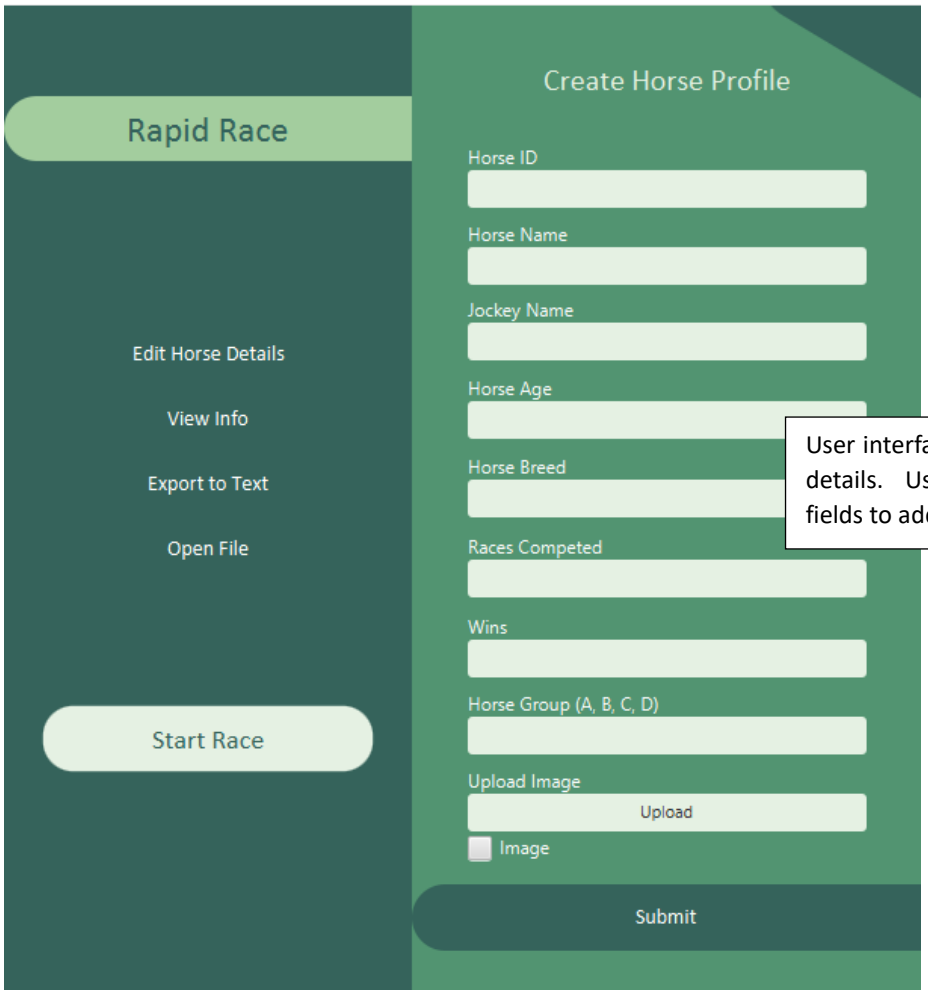
The method assigns for Submit button.

```
// Submit button method.
@FXML
void submit(ActionEvent event) {
    String enteredID = textId.getText().trim();

    // Check if the ID is empty.
    if (enteredID.isEmpty()) {
        showAlert("Horse ID cannot be empty.");
        return;
    }

    // Check if the entered ID is a positive integer.
    try {
        int id = Integer.parseInt(enteredID);
        if (id <= 0) {
            throw new NumberFormatException();
        }
    } catch (NumberFormatException e) {
        showAlert("Horse ID must be a positive integer.");
        return;
    }

    // Check if the ID is already taken.
    if (dataModel.isIDTaken(enteredID)) {
        // If the update operation is in progress, proceed with updating horse
details.
        if (updateLabel.isVisible()) {
            updateHorse1(event, enteredID);
        } else {
            showAlert("Horse ID is already taken.");
        }
        return;
    }

    // If the update button is not pressed and ID is not taken, add a new horse.
    if (!updateLabel.isVisible()) {
        addNewHorse();
    }
}
```

Method created to add new horse information.

```
// Method to add new horses.
private void addNewHorse() {
    // Checks for group availability.
    String group = textGroup.getText().toUpperCase();
    if (!dataModel.registerHorseForGroup(group)) {
        showAlert("All slots filled for group " + group);
```

```
            return;
        }
        try {
            // Get age from text field.
            int age = Integer.parseInt(textAge.getText());
            // Get wins from text field.
            int wins = Integer.parseInt(textWin.getText());
            // Get compete from text field.
            int compete = Integer.parseInt(textCompete.getText());
            // Get breed from text field.
            String breed = textBreed.getText();

            // Create a new Horse object with all arguments.
            Horse horse = new Horse(textId.getText(), textName.getText(),
textJockey.getText(),
                    age, wins, compete, breed, group);

            dataModel.getHorses().add(horse);
            // If submission is successful, clear the fields.
            clearFields();
        } catch (NumberFormatException e) {
            // Check for input validation.
            showAlert("Please enter valid numbers for age, wins, and compete.");
        } catch (IllegalArgumentException e) {
            showAlert(e.getMessage());
        }                                                                    }
```

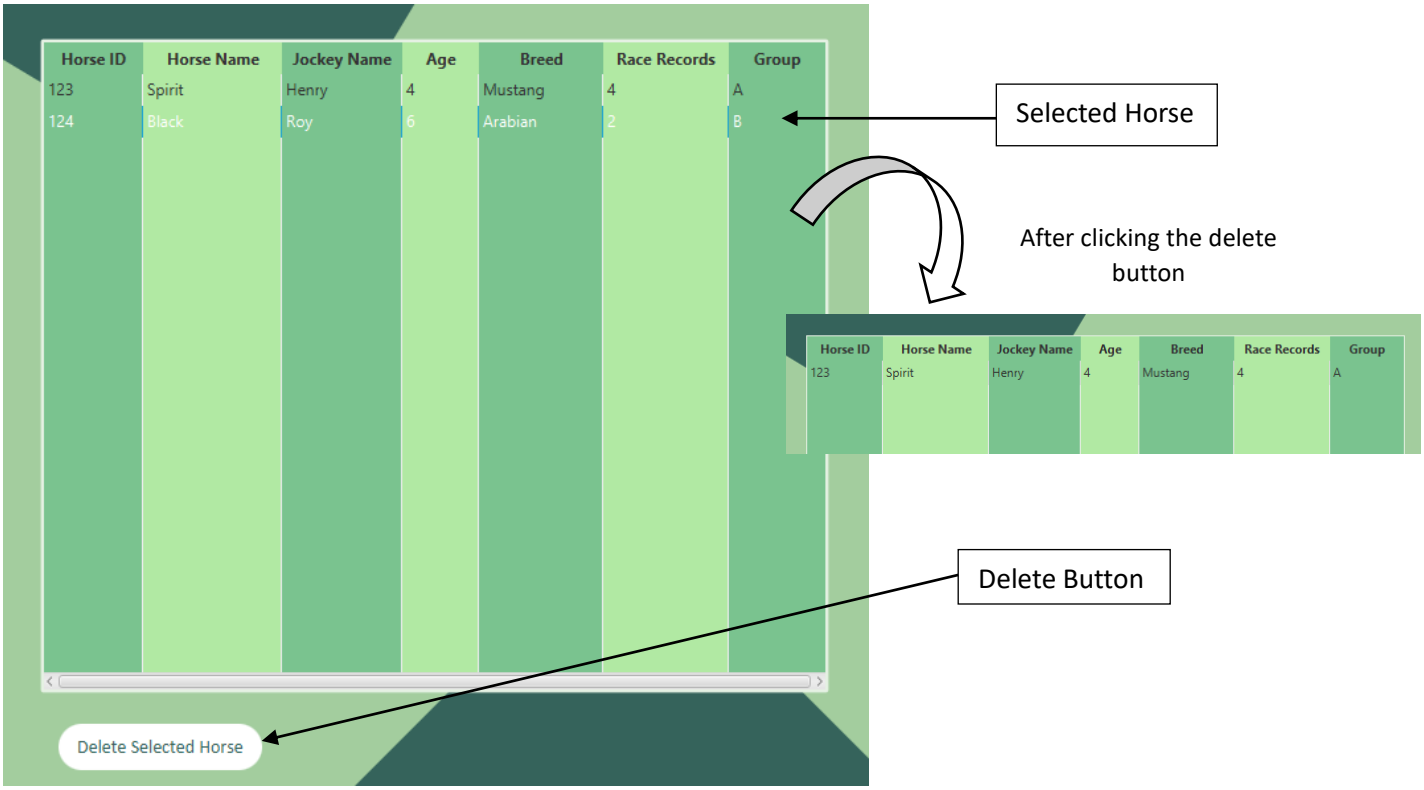


User interface for adding horse details. Users can use text fields to add details.

II. Deleting Horse Details

This function manages the deletion of specific horses from the collection. When the horse is chosen from the list, and clicking on the “Delete Selected Horse” button user can easily delete the relevant horse from the system. This function ensures the horse group count is also changed accordingly.

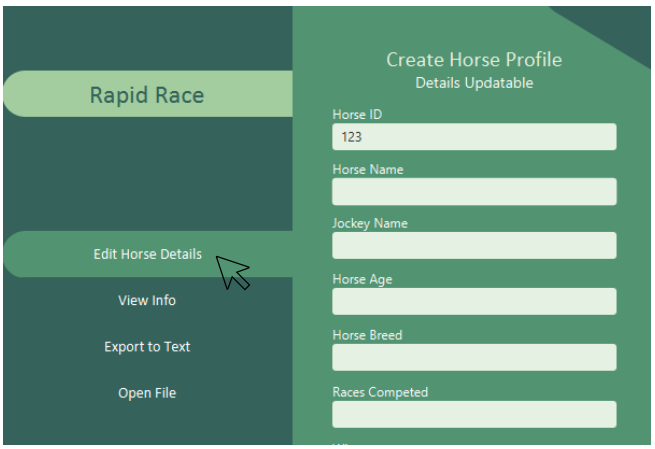
```
// Method that assign to delete horse details button.
@FXML
void removeHorse(ActionEvent event) {
    Horse selectedHorse = tableHorseDetails.getSelectionModel().getSelectedItem();
    if (selectedHorse != null) {
        // Decrement the count of the group to which the horse belongs.
        String group = selectedHorse.getGroup();
        if (dataModel.decrementGroupCount(group)) {
            // Remove the horse from the list.
            dataModel.getHorses().remove(selectedHorse);
            tableHorseDetails.getItems().remove(selectedHorse);
        } else {
            // If the group count couldn't be decremented, show an error message
            showAlert("Failed to decrement group count. Please try again.");
        }
    }
}
```

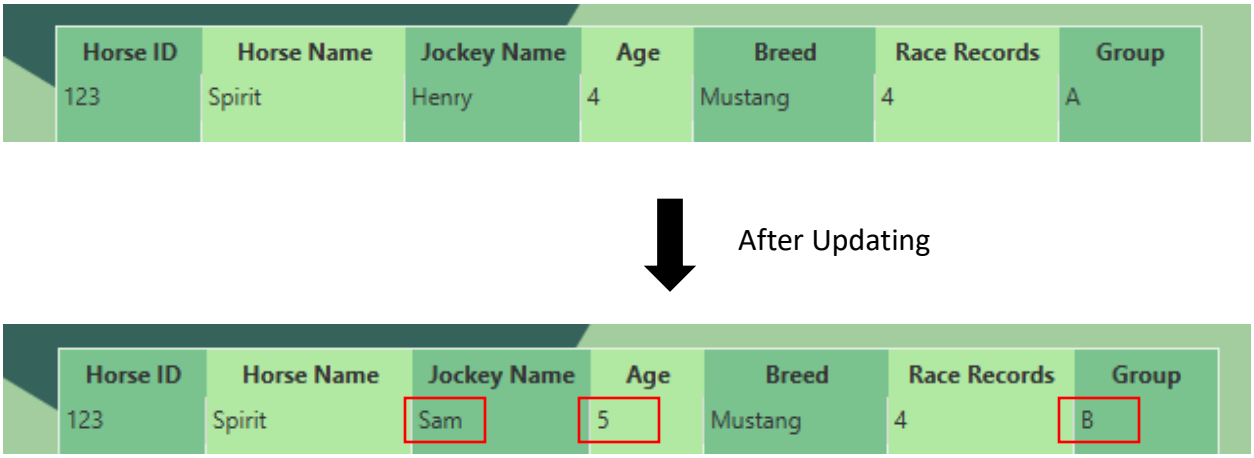


III. Update Horse Details

This function updates the information on an existing horse in the system. First, it retrieves the horse information using the entered ID. If the horse ID exists, it updates information. After clicking the Update horse details button user can update relevant information using the same text fields that are used for adding horse details. First user needs to provide an existing horse ID and then click the update button. Then all the text fields fill with old data. After that user can make changes and press the same submit button to input updated data into the system.

```
// Method to update horse details
private void updateHorse1(ActionEvent event, String enteredID) {
    Horse existingHorse = dataModel.getHorseByID(enteredID);
    if (existingHorse != null) {
        // Decrement the count of the old group and update the group if needed
        String oldGroup = existingHorse.getGroup();
        String newGroup = textGroup.getText().toUpperCase();
        if (!oldGroup.equals(newGroup)) {
            if (!dataModel.updateHorseGroup(enteredID, newGroup)) {
                showAlert("Failed to update horse group. Please try again.");
                return;
            }
        }
        try {
            // Update horse details
            existingHorse.setHorseName(textName.getText());
            existingHorse.setJockeyName(textJockey.getText());
            existingHorse.setAge(Integer.parseInt(textAge.getText()));
            existingHorse.setBreed(textBreed.getText());
            existingHorse.setCompete(Integer.parseInt(textCompete.getText()));
            existingHorse.setWins(Integer.parseInt(textWin.getText()));
            existingHorse.setGroup(newGroup);
            tableHorseDetails.refresh(); // Refresh the table view to reflect changes
            // If submission is successful, clear the fields
            clearFields();
            // Hide the update notification label
            updateLabel.setVisible(false);
        } catch (NumberFormatException e) {
            showAlert("Please enter valid numbers for age, wins, and compete.");
        } catch (IllegalArgumentException e) {
            showAlert(e.getMessage());
        }
    }
}
```





IV. View the Registered Horses' Details Table

This function is implemented in a new scene in the JavaFX application. The goToViewDetails method handles the transition and passes the required data to the controller. The table view is customized to display horse information respectively.

The sorting method handles the sorting of the horse details according to the horse ID. It retrieves the list of horses from the data model and converts it to an array for sorting purposes. After, sort the array using the custom implementation of the selection sort algorithm.

```
private void updateTable() {
    if (horses != null) {
        ObservableList<Horse> horseList = FXCollections.observableArrayList(horses);
        tableView.setItems(horseList);
    }
}

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    dataModel = DataModel.getInstance();
    // Set the table items to the data model's list of horses
    tableView.setItems(dataModel.getHorses());
    colId.setCellValueFactory(new PropertyValueFactory<>("horseID"));
    colName.setCellValueFactory(new PropertyValueFactory<>("horseName"));
    colJockey.setCellValueFactory(new PropertyValueFactory<>("jockeyName"));
    colAge.setCellValueFactory(new PropertyValueFactory<>("age"));
    colBreed.setCellValueFactory(new PropertyValueFactory<>("breed"));
    colRecord.setCellValueFactory(new PropertyValueFactory<>("wins"));
    colGroup.setCellValueFactory(new PropertyValueFactory<>("group"));
}
```

Method to go back to the main page.

```
@FXML
void goToMain(ActionEvent event) throws IOException {
    FXMLLoader loader = new FXMLLoader(getClass().getResource("hello-view.fxml"));
    Parent root = loader.load();
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
```

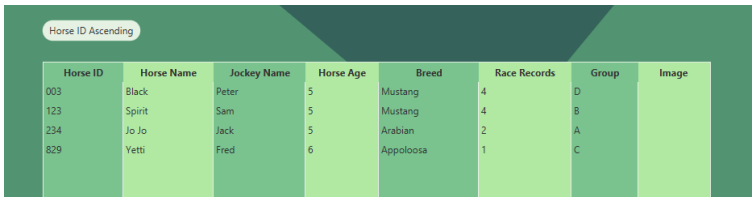
```
    Scene scene = new Scene(root);
    stage.setScene(scene);
    stage.show();
}
```

Sorting Algorithm.

```
@FXML
void sortTable(ActionEvent event) {
    // Get the list of horses
    List<Horse> horseList = dataModel.getHorses();

    // Convert the list to an array
    Horse[] horsesArray = horseList.toArray(new Horse[0]);

    // Sort the array using custom selection sort
    CustomSortView.selectionSort(horsesArray);

    // Clear the table and add the sorted horses back
    tableView.getItems().clear();
    tableView.getItems().addAll(horsesArray);
}
```

Before Sorting

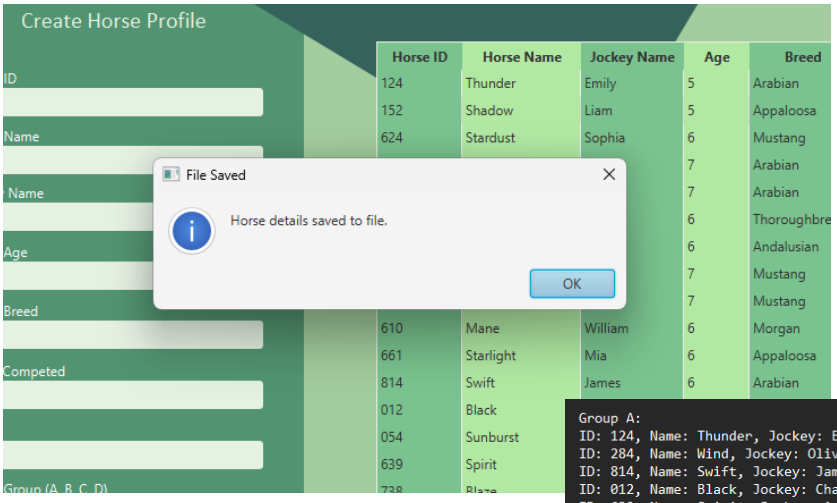
After Sorting



V. Save the Horse Details into the Text File

This method horse details to a text file. It organizes the data by grouping horses according to their allocated groups. This function efficiently handles file writing. Furthermore, it ensures the saved information is readable and structured well.

```
// Method to save horse details to a text file
public void saveToFile(String filename) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {
        for (String group : groupCount.keySet()) {
            writer.write("Group " + group + ":\n");
            for (Horse horse : horses) {
                if (horse.getGroup().equals(group)) {
                    writer.write(horse.toString() + "\n");
                }
            }
            writer.write("\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```




```
Group A:
ID: 124, Name: Thunder, Jockey: Emily, Age: 5, Wins: 5, Compete: 10, Breed: Arabian, Group: A
ID: 284, Name: Wind, Jockey: Olivia, Age: 7, Wins: 8, Compete: 20, Breed: Arabian, Group: A
ID: 814, Name: Swift, Jockey: James, Age: 6, Wins: 4, Compete: 11, Breed: Arabian, Group: A
ID: 012, Name: Black, Jockey: Charlotte, Age: 6, Wins: 12, Compete: 20, Breed: Mustang, Group: A
ID: 639, Name: Spirit, Jockey: Amelia, Age: 6, Wins: 4, Compete: 20, Breed: Mustang, Group: A

Group B:
ID: 152, Name: Shadow, Jockey: Liam, Age: 5, Wins: 3, Compete: 7, Breed: Appaloosa, Group: B
ID: 927, Name: Firefly, Jockey: Ethan, Age: 6, Wins: 5, Compete: 18, Breed: Thoroughbred, Group: B
ID: 661, Name: Starlight, Jockey: Mia, Age: 6, Wins: 10, Compete: 22, Breed: Appaloosa, Group: B
ID: 054, Name: Sunburst, Jockey: Alexander, Age: 7, Wins: 4, Compete: 10, Breed: Mustang, Group: B
ID: 220, Name: Wildfire, Jockey: Harper, Age: 3, Wins: 2, Compete: 10, Breed: Appaloosa, Group: B

Group C:
ID: 624, Name: Stardust, Jockey: Sophia, Age: 6, Wins: 5, Compete: 15, Breed: Mustang, Group: C
ID: 777, Name: Moon, Jockey: Mason, Age: 7, Wins: 7, Compete: 21, Breed: Mustang, Group: C
ID: 610, Name: Mane, Jockey: William, Age: 6, Wins: 8, Compete: 15, Breed: Morgan, Group: C
ID: 738, Name: Blaze, Jockey: Benjamin, Age: 4, Wins: 0, Compete: 4, Breed: Appaloosa, Group: C
ID: 501, Name: Aurora, Jockey: Lucas, Age: 7, Wins: 8, Compete: 25, Breed: Arabian, Group: C

Group D:
ID: 372, Name: Rider, Jockey: Noah, Age: 7, Wins: 4, Compete: 17, Breed: Arabian, Group: D
ID: 391, Name: Mystic, Jockey: Ava, Age: 6, Wins: 11, Compete: 20, Breed: Andalusian, Group: D
ID: 224, Name: Brave, Jockey: Mason, Age: 7, Wins: 9, Compete: 11, Breed: Mustang, Group: D
ID: 901, Name: Velvet, Jockey: Evelyn, Age: 5, Wins: 7, Compete: 15, Breed: Mustang, Group: D
ID: 200, Name: Dream, Jockey: Oliver, Age: 4, Wins: 1, Compete: 15, Breed: Thoroughbred, Group: D
```

VI. Selecting Four Horses Randomly

This functionality selects random horses from each group. (A, B, C, D) Then store all the selected horse's information in a hashmap for future purposes.

```
public Map<String, Horse> selectRandomHorses() {
    Map<String, Horse> randomHorses = new HashMap<>();
    Random random = new Random();
    for (String group : groupCount.keySet()) {
        List<Horse> horsesInGroup = new ArrayList<>();
        // Collect horses in the current group
        for (Horse horse : horses) {
            if (horse.getGroup().equals(group)) {
                horsesInGroup.add(horse);
            }
        }
        // Randomly select a horse from the group
        if (!horsesInGroup.isEmpty()) {
            int randomIndex = random.nextInt(horsesInGroup.size());
            randomHorses.put(group, horsesInGroup.get(randomIndex));
        }
    }
    return randomHorses;
}
```

VII. Select Winning Horses

This function organizes the list of winning horses that were selected before based on their race time. It uses a custom-created selection sort algorithm to organize the horses in ascending order of race time. It then displays the top 3 horses in a table-based structure along with horse details.

Custom Selection sort algorithm for sort race time.

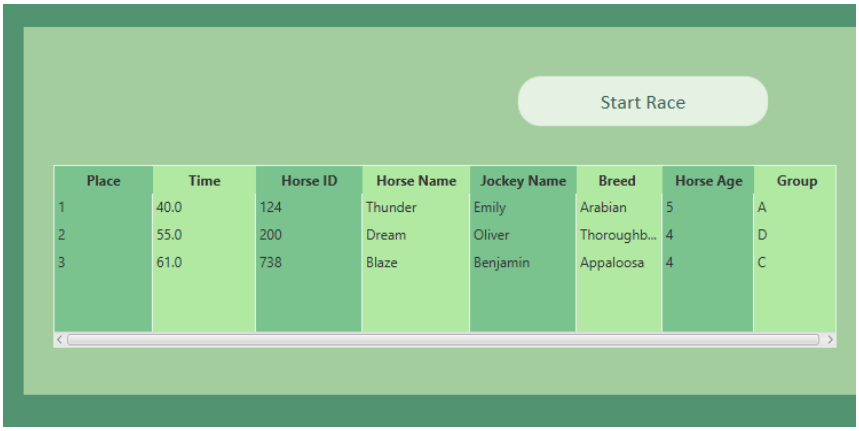
```
public class CustomSort {
    public static void selectionSort(Horse[] horses) {
        int n = horses.length;
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (horses[j].getRaceTime() < horses[minIndex].getRaceTime())
{
                    minIndex = j;
                }
            }
            // Swap horses[i] with horses[minIndex]
            Horse temp = horses[minIndex];
            horses[minIndex] = horses[i];
            horses[i] = temp;
        }
    }
}
```

Assign to race begin button.

```
@FXML
void raceBegin(ActionEvent event) {
    if (tableView.getItems().isEmpty()) {
        // Show message if horses are not selected for finals yet
        System.out.println("Horses not selected for finals yet!");
        return;
    }
    // Sort the selected horses based on their race times
    Horse[] horsesArray = tableView.getItems().toArray(new Horse[0]);
    CustomSort.selectionSort(horsesArray);
    // Display the sorted results
    displayRaceResults(horsesArray);
    // Visualize the race times using the bar chart
    visualizeRaceTimes(horsesArray);
}
```

Display the winning horse details

```
private void displayRaceResults(Horse[] sortedHorses) {
    // Clear the table before adding new results
    tableViewWinHorse.getItems().clear();

    // Add the first 3 horses to the winning horse's table
    for (int i = 0; i < Math.min(3, sortedHorses.length); i++) {
        sortedHorses[i].setPlace(i + 1);
        tableViewWinHorse.getItems().add(sortedHorses[i]);
    }
}
```

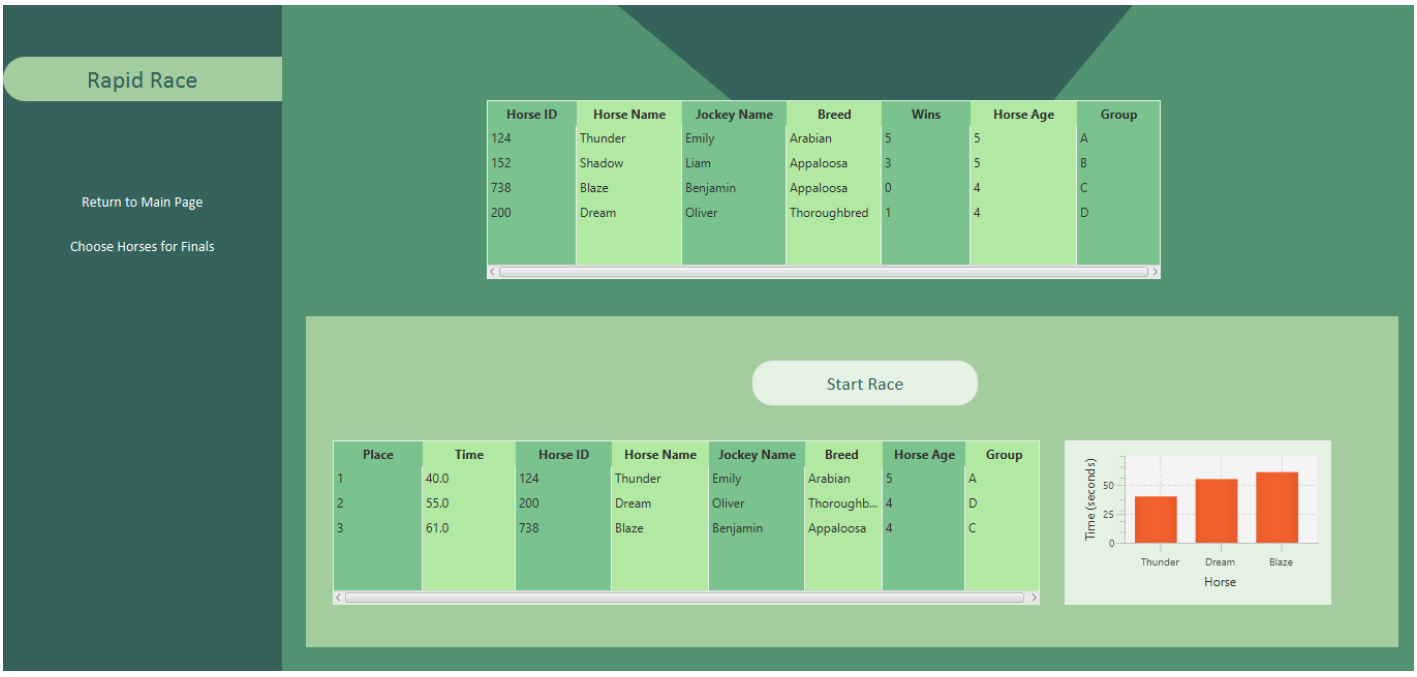


VIII. Visualize Winning Horses

This code visualizes the racing times of the top three horses. This function uses a bar chart to represent the winning horse's time. It iterates through the sorted list of horses and adds their names and race times to the chart series.

```
private void visualizeRaceTimes(Horse[] sortedHorses) {
    raceTimeChart.getData().clear(); // Clear old data

    XYChart.Series<String, Number> series = new XYChart.Series<>();
    for (int i = 0; i < Math.min(3, sortedHorses.length); i++) {
        Horse horse = sortedHorses[i];
        series.getData().add(new XYChart.Data<>(horse.getHorseName(),
horse.getRaceTime()));
    }

    raceTimeChart.getData().add(series);
}
```

View of the race interface



IX. GUI fxml codes

Main page

```
<AnchorPane maxHeight="-Infinity" maxWidth="-Infinity" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="660.0" prefWidth="1315.0" style="-fx-background-
color: #a3cd9e;" xmlns="http://javafx.com/javafx/21"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="org.example.javacw.MainController">
   <children>
      <AnchorPane layoutY="-5.0" prefHeight="506.0" prefWidth="207.0" style="-fx-
background-color: #35635b;" AnchorPane.bottomAnchor="-1.0" AnchorPane.leftAnchor="0.0"
AnchorPane.topAnchor="-5.0">
         <children>
            <Button fx:id="btnRaceStart" layoutX="26.0" layoutY="469.0"
mnemonicParsing="false" onAction="#goToHorseRace" prefHeight="43.0" prefWidth="218.0"
styleClass="button2" stylesheets="@fullpackstyling.css" text="Start Race"
textFill="#35635b">
               <font>
                  <Font name="Calibri" size="18.0" />
               </font>
            </Button>
            <VBox layoutY="214.0" prefHeight="240.0" prefWidth="270.0"
AnchorPane.bottomAnchor="212.0" AnchorPane.leftAnchor="0.0"
AnchorPane.topAnchor="214.0">
               <children>
                  <Button fx:id="btnUpdate" mnemonicParsing="false"
onAction="#updateHorse" prefHeight="43.0" prefWidth="313.0" styleClass="button1"
stylesheets="@fullpackstyling.css" text="Edit Horse Details" textFill="WHITE">
                     <font>
                        <Font name="Calibri" size="14.0" />
                     </font>
                  </Button>
                  <Button fx:id="btnView" mnemonicParsing="false"
onAction="#goToViewDetails" prefHeight="43.0" prefWidth="295.0" styleClass="button1"
stylesheets="@fullpackstyling.css" text="View Info" textFill="WHITE">
                     <font>
                        <Font name="Calibri" size="14.0" />
                     </font>
                  </Button>
                  <Button fx:id="btnSaveFile" mnemonicParsing="false"
onAction="#saveToFile" prefHeight="43.0" prefWidth="289.0" styleClass="button1"
stylesheets="@fullpackstyling.css" text="Export to Text" textFill="WHITE">
                     <font>
                        <Font name="Calibri" size="14.0" />
                     </font>
                  </Button>
                  <Button fx:id="btnOpenText" mnemonicParsing="false"
onAction="#openTextFile" prefHeight="43.0" prefWidth="279.0" styleClass="button1"
stylesheets="@fullpackstyling.css" text="Open File" textFill="WHITE">
                     <font>
                        <Font name="Calibri" size="14.0" />
                     </font>
                  </Button>
               </children>
            </VBox>
            <Label layoutX="78.0" layoutY="117.0" text="Ride the Future"
textFill="WHITE">
               <font>
                  <Font name="Calibri" size="18.0" />
               </font>
```

```
            </Label>
            <Pane layoutY="57.0" prefHeight="53.0" prefWidth="270.0" style="-fx-
background-color: #a3cd9e; -fx-background-radius: 30 0 0 30;">
               <children>
                  <Label alignment="TOP_LEFT" layoutX="103.0" layoutY="13.0"
text="Rapid Race" textFill="WHITE">
                     <font>
                        <Font name="Calibri" size="24.0" />
                     </font>
                  </Label>
                  <ImageView fitHeight="43.0" fitWidth="48.0" layoutX="44.0"
layoutY="6.0" pickOnBounds="true" preserveRatio="true">
                     <image>
                        <Image url="@img/logoWhite.png" />
                     </image>
                  </ImageView>
               </children>
            </Pane>
         </children>
      </AnchorPane>
      <AnchorPane layoutX="270.0" prefHeight="660.0" prefWidth="337.0" style="-fx-
background-color: #529471;" AnchorPane.bottomAnchor="0.0" AnchorPane.topAnchor="0.0">
         <children>
            <Label layoutX="87.0" layoutY="38.0" text="Create Horse Profile"
textFill="#e5f1e3">
               <font>
                  <Font name="Calibri" size="20.0" />
               </font>
            </Label>
            <VBox layoutX="37.0" layoutY="92.0" prefHeight="477.0" prefWidth="264.0">
               <children>
                  <Label prefHeight="17.0" prefWidth="207.0" text="Horse ID"
textFill="WHITE" />
                  <TextField fx:id="textId" style="-fx-background-color: #e5f1e3;" />
                  <Pane prefHeight="10.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Horse Name"
textFill="WHITE" />
                  <TextField fx:id="textName" style="-fx-background-color: #e5f1e3;"
/>
                  <Pane prefHeight="9.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Jockey Name"
textFill="WHITE" />
                  <TextField fx:id="textJockey" style="-fx-background-color: #e5f1e3;"
/>
                  <Pane prefHeight="10.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Horse Age"
textFill="WHITE" />
                  <TextField fx:id="textAge" style="-fx-background-color: #e5f1e3;" />
                  <Pane prefHeight="10.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Horse Breed"
textFill="WHITE" />
                  <TextField fx:id="textBreed" style="-fx-background-color: #e5f1e3;"
/>
                  <Pane prefHeight="11.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Races Competed"
textFill="WHITE" />
                  <TextField fx:id="textCompete" prefHeight="25.0" prefWidth="75.0"
style="-fx-background-color: #e5f1e3;" />
                  <Pane prefHeight="11.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Wins"
textFill="WHITE" />
                  <TextField fx:id="textWin" prefHeight="25.0" prefWidth="75.0"
```

```
style="-fx-background-color: #e5f1e3;" />
                  <Pane prefHeight="9.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Horse Group (A, B,
C, D)" textFill="WHITE" />
                  <TextField fx:id="textGroup" prefHeight="25.0" prefWidth="75.0"
style="-fx-background-color: #e5f1e3;" />
                  <Pane prefHeight="9.0" prefWidth="218.0" />
                  <Label prefHeight="17.0" prefWidth="207.0" text="Upload Image"
textFill="WHITE" />
                  <Button fx:id="btnUploadImage" alignment="CENTER"
contentDisplay="BOTTOM" mnemonicParsing="false" nodeOrientation="LEFT_TO_RIGHT"
onAction="#uploadImage" prefHeight="24.0" prefWidth="265.0" style="-fx-background-
color: #e5f1e3;" text="Upload">
                     <font>
                        <Font size="11.0" />
                     </font>
                     <cursor>
                        <Cursor fx:constant="OPEN_HAND" />
                     </cursor>
                  </Button>
                  <Pane prefHeight="23.0" prefWidth="201.0">
                     <children>
                        <CheckBox fx:id="checkImage" layoutY="3.0"
mnemonicParsing="false" text="Image " textFill="WHITE" />
                     </children></Pane>
               </children>
            </VBox>
            <Button fx:id="btnSubmit" layoutY="582.0" mnemonicParsing="false"
onAction="#submit" prefHeight="43.0" prefWidth="337.0" styleClass="button5"
stylesheets="@fullpackstyling.css" text="Submit" textFill="WHITE">
               <font>
                  <Font name="Calibri" size="14.0" />
               </font>
            </Button>
            <Label fx:id="updateLabel" layoutX="114.0" layoutY="61.0" text="Details
Updatable" textFill="WHITE" visible="false">
               <font>
                  <Font size="14.0" />
               </font>
            </Label>
         </children></AnchorPane>
      <Pane layoutX="533.0" layoutY="-166.0" prefHeight="286.0" prefWidth="422.0"
rotate="31.0" style="-fx-background-color: #35635b; -fx-background-radius: 30;" />
      <Pane layoutX="731.0" layoutY="584.0" prefHeight="434.0" prefWidth="626.0"
rotate="-45.0" style="-fx-background-color: #35635b; -fx-background-radius: 30;" />
      <TableView fx:id="tableHorseDetails" layoutX="671.0" layoutY="68.0"
prefHeight="491.0" prefWidth="593.0" style="-fx-background-color: #e5f1e3;"
AnchorPane.leftAnchor="671.0">
         <columns>
            <TableColumn fx:id="colId" prefWidth="75.0" style="-fx-background-color:
#7ac38f;" text="Horse ID" />
            <TableColumn fx:id="colName" prefWidth="105.0" sortable="false" style="-
fx-background-color: #b1e9a3;" text="Horse Name" />
            <TableColumn fx:id="colJockey" prefWidth="91.0" sortable="false" style="-
fx-background-color: #7ac38f;" text="Jockey Name" />
            <TableColumn fx:id="colAge" prefWidth="58.0" sortable="false" style="-fx-
background-color: #b1e9a3;" text="Age" />
            <TableColumn fx:id="colBreed" prefWidth="94.0" sortable="false" style="-
fx-background-color: #7ac38f;" text="Breed" />
            <TableColumn fx:id="colRecords" prefWidth="95.0" sortable="false" style="-
fx-background-color: #b1e9a3;" text="Race Records" />
            <TableColumn fx:id="colGroup" prefWidth="74.0" sortable="false" style="-
```

```
fx-background-color: #7ac38f;" text="Group" />
         </columns>
      </TableView>
      <Button fx:id="btnRemove" layoutX="683.0" layoutY="584.0"
mnemonicParsing="false" onAction="#removeHorse" prefHeight="34.0" prefWidth="154.0"
styleClass="button2" stylesheets="@fullpackstyling.css" text="Delete Selected Horse"
textFill="#35635b" AnchorPane.leftAnchor="683.0">
         <font>
            <Font name="Calibri" size="14.0" />
         </font>
      </Button>
   </children>
</AnchorPane>
```

View Horse Details page fxml ☑

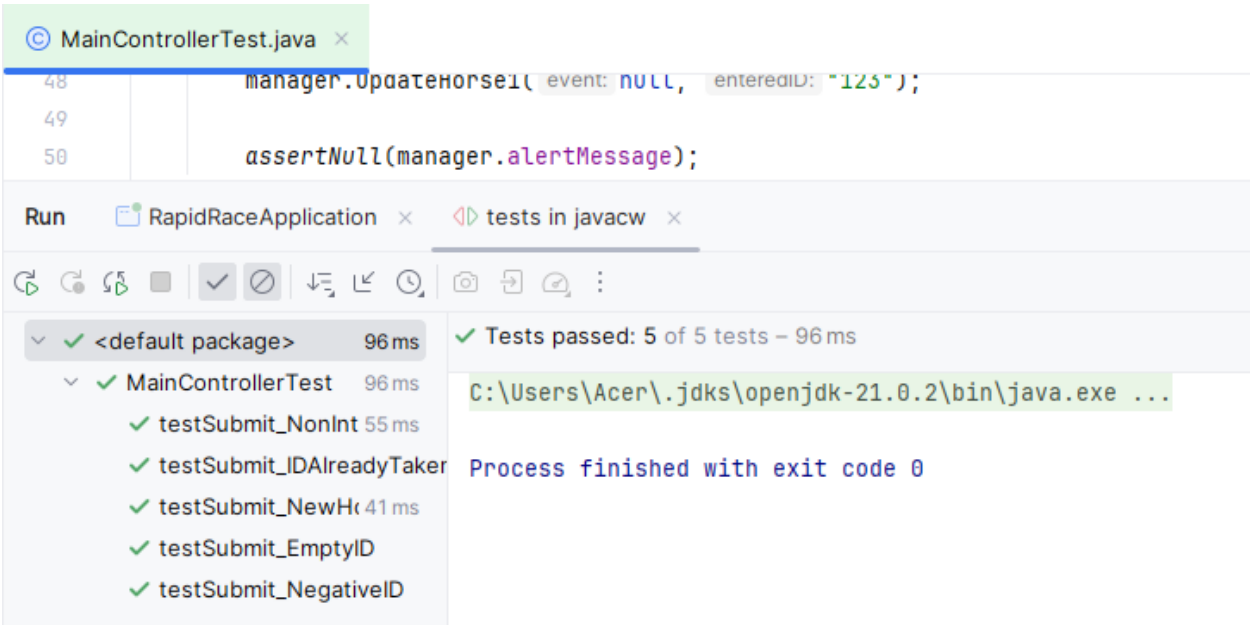
Horse Race View fxml ☑

4. J-units, Test plan, and test cases

I. J-units

```
package org.example.javacw;
import org.junit.Test;
import java.awt.*;
import java.awt.event.ActionEvent;
import static org.junit.Assert.*;

public class MainControllerTest {
    @Test
    public void testSubmit_EmptyID() {
        HorseManager manager = new HorseManager();
        manager.textId.setText(""); // Empty ID

        manager.submit(null);

        assertEquals("Horse ID cannot be empty.", manager.alertMessage);
    }

    @Test
    public void testSubmit_NonIntegerID() {
        HorseManager manager = new HorseManager();
        manager.textId.setText("abc"); // Non-integer ID

        manager.submit(null);

        assertEquals("Horse ID must be a positive integer.", manager.alertMessage);
    }

    @Test
    public void testSubmit_NegativeID() {
        HorseManager manager = new HorseManager();
        manager.textId.setText("-123"); // Negative ID

        manager.submit(null);

        assertEquals("Horse ID must be a positive integer.", manager.alertMessage);
    }

    @Test
    public void testSubmit_IDAlreadyTaken_UpdateInProgress() {
        HorseManager manager = new HorseManager();
        manager.textId.setText("123"); // ID already taken
        manager.updateLabel.setVisible(true); // Update in progress

        // Mock behavior of updateHorse1 method
        manager.updateHorse1(null, "123");

        assertNull(manager.alertMessage);
    }
    @Test
    public void testSubmit_NewHorse_NoUpdateInProgress() {
        HorseManager manager = new HorseManager();
        manager.textId.setText("123"); // New ID
        manager.updateLabel.setVisible(false); // No update in progress
        // Mock behavior of addNewHorse method
        manager.addNewHorse();
```

```
        assertNull(manager.alertMessage);
    }

    // Inner class to simulate HorseManager
    class HorseManager {
        // Mock alertMessage for testing purposes
        String alertMessage;

        // Mock text fields and labels
        TextField textId = new TextField();
        Label updateLabel = new Label();

        public void submit(ActionEvent event) {
            String enteredID = textId.getText().trim();

            // Check if the ID is empty
            if (enteredID.isEmpty()) {
                showAlert("Horse ID cannot be empty.");
                return;
            }

            // Check if the entered ID is a positive integer
            try {
                int id = Integer.parseInt(enteredID);
                if (id <= 0) {
                    throw new NumberFormatException();
                }
            } catch (NumberFormatException e) {
                showAlert("Horse ID must be a positive integer.");
                return;}

            // Check if the ID is already taken
            if (isIDTaken(enteredID)) {
      // If the update operation is in progress, proceed with updating horse details
                if (updateLabel.isVisible()) {
                    updateHorse1(event, enteredID);
                } else {
                    showAlert("Horse ID is already taken.");
                }
                return;
            }

       // If the update button is not pressed and ID is not taken, add a new horse
            if (!updateLabel.isVisible()) {
                addNewHorse();
            }
        }
        private void showAlert(String message) {
            // Set alertMessage for testing purposes
            this.alertMessage = message;
        }
        private boolean isIDTaken(String id) {
            return false;
        }
        private void updateHorse1(ActionEvent event, String enteredID) {

        }
        private void addNewHorse() {

        }
    }
}
```



J units test results. Tests passed

Above Junit test related to the 'MainController' class. Specifically, it tests the 'submit' method. The 'submit' method is special because it handles user input related to horse IDs or updating or adding horse details accordingly.

Each test case code tests,

- Submitting empty horse ID
- Submitting non-integer horse ID
- Submitting a negative integer for horse ID
- Submitting horse ID already taken with an update in progress
- Submitting horse ID already taken with no update in progress

The expected behavior of the 'submit' functionality is asserted using,

- 'assertEquals' statement
- 'assertNull' statement

II. Test Plan and Test Cases

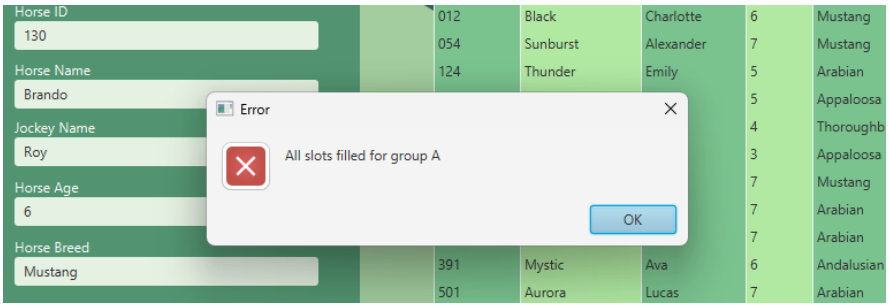
Test case 1: Add a new horse with valid inputs

- Inputs: 124, Thunder, Emily, 5, Arabian, 5, A
- Expected Output: Horse added successfully, Text fields cleared.
- Actual Output: Horse added successfully, Text fields cleared.

Horse ID	Horse Name	Jockey Name	Age	Breed	Race Records	Group
012	Black	Charlotte	6	Mustang	12	A
054	Sunburst	Alexander	7	Mustang	4	B
124	Thunder	Emily	5	Arabian	5	A
152	Shadow	Liam	5	Appaloosa	3	B
200	Dream	Oliver	4	Thoroughbred	1	D
220	Wildfire	Harper	3	Appaloosa	2	B
224	Brave	Mason	7	Mustang	9	D
284	Wind	Olivia	7	Arabian	8	A
372	Rider	Noah	7	Arabian	4	D
391	Mystic	Ava	6	Andalusian	11	D
501	Aurora	Lucas	7	Arabian	8	C
610	Mane	William	6	Morgan	8	C
624	Stardust	Sophia	6	Mustang	5	C
639	Spirit	Amelia	6	Mustang	4	A
661	Starlight	Mia	6	Appaloosa	10	B
738	Blaze	Benjamin	4	Appaloosa	0	C
777	Moon	Mason	7	Mustang	7	C
814	Swift	James	6	Arabian	4	A
901	Velvet	Evelyn	5	Mustang	7	D

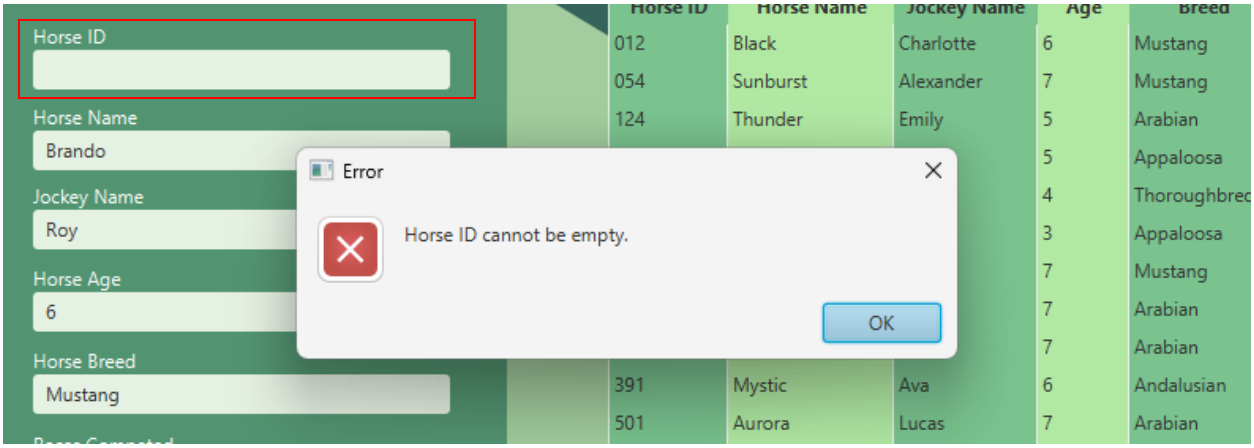
Test case 2: Add a new horse- All Slots Filled

- Inputs: 130, Brando, Roy, 7, Mustang, 5, A
- Expected Output: Alert displayed: "All slots filled for group Group A”
- Actual Output: Alert displayed: "All slots filled for group Group A”



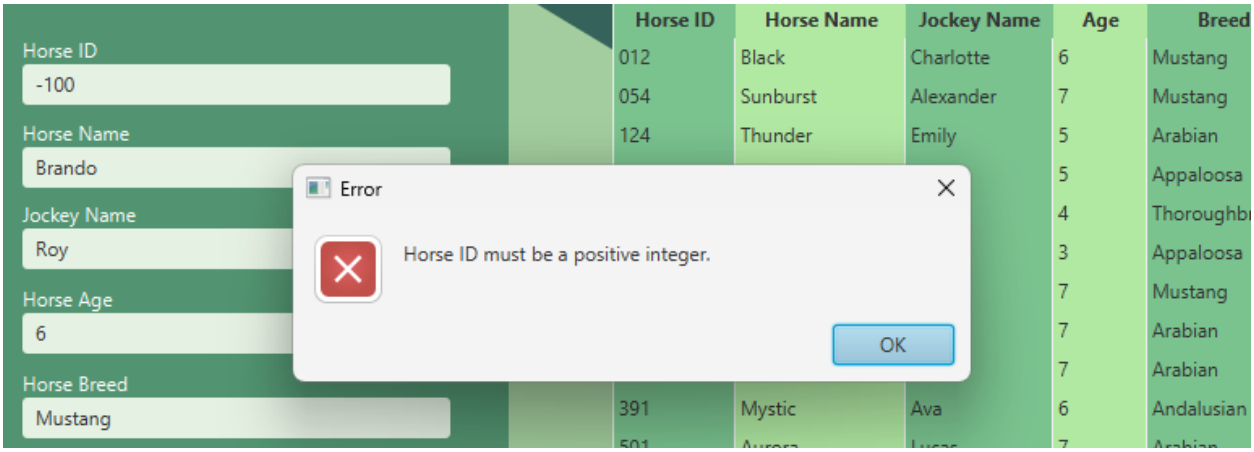
Test case 3: Add a new horse- Empty Horse ID

- Inputs: Brando, Roy, 7, Mustang, 5, A
- Expected Output: Alert displayed: "Horse ID can not be empty."
- Actual Output: Alert displayed: "Horse ID can not be empty."



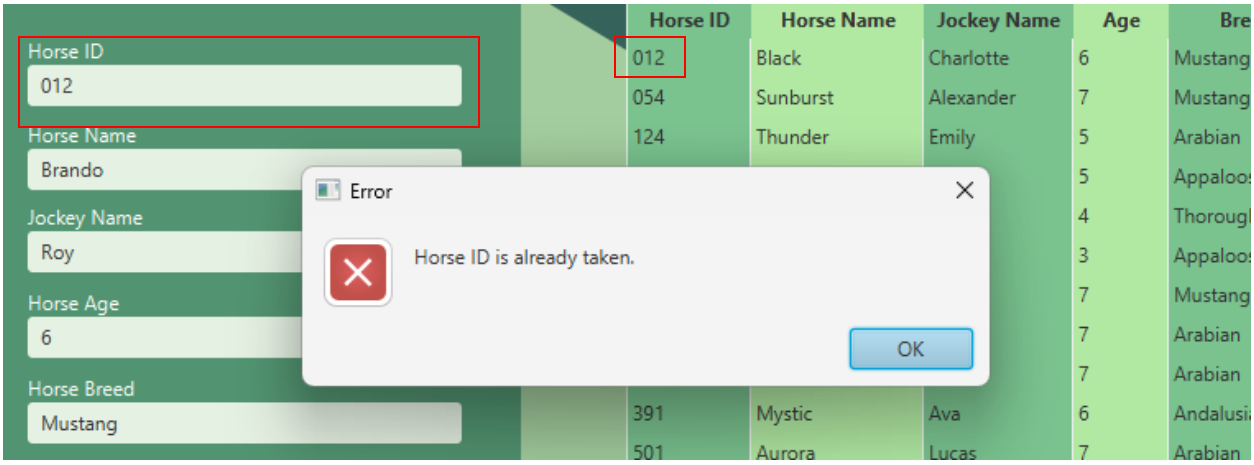
Test case 4: Add a new horse- Horse ID incorrect data type or negative.

- Inputs: -100, Brando, Roy, 7, Mustang, 5, A
- Expected Output: Alert displayed: "Horse ID must be a positive integer"
- Actual Output: Alert displayed: "Horse ID must be a positive integer"



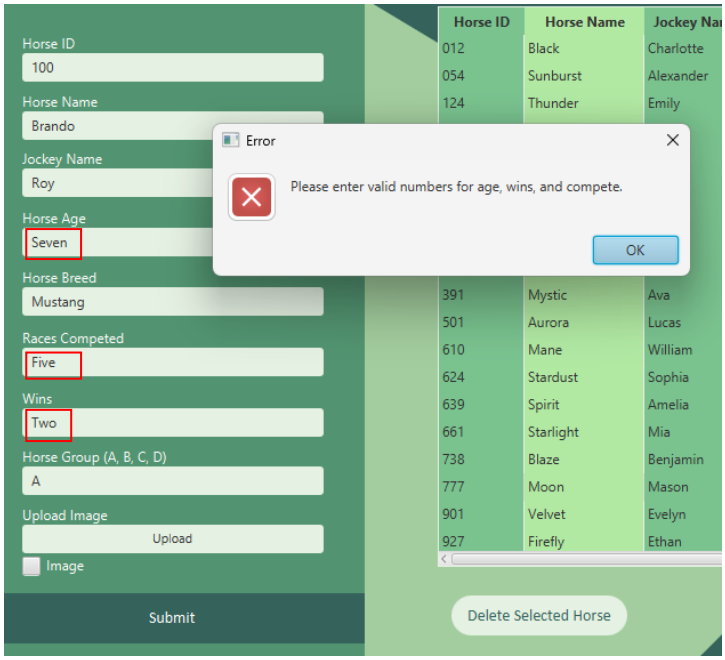
Test case 5: Add a new horse- Horse ID is already taken.

- Inputs: 012, Brando, Roy, 7, Mustang, 5, A
- Expected Output: Alert displayed: "Horse ID is already taken”
- Actual Output: Alert displayed: "Horse ID is already taken”



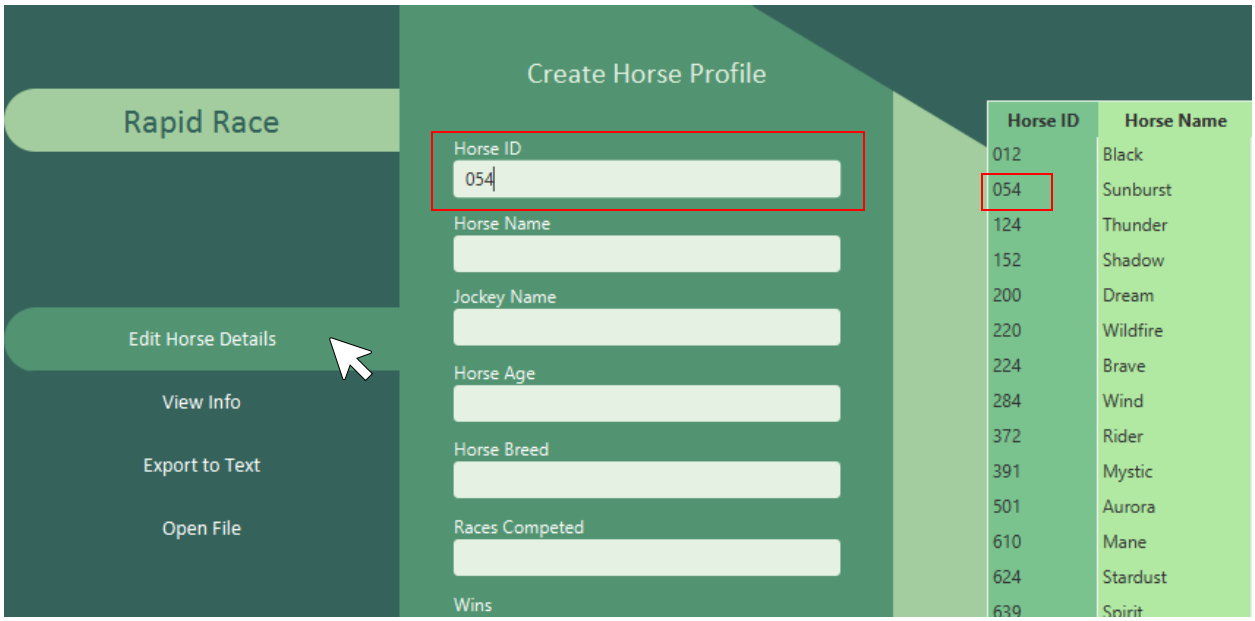
Test case 6: Add a new horse- Invalid Inputs.

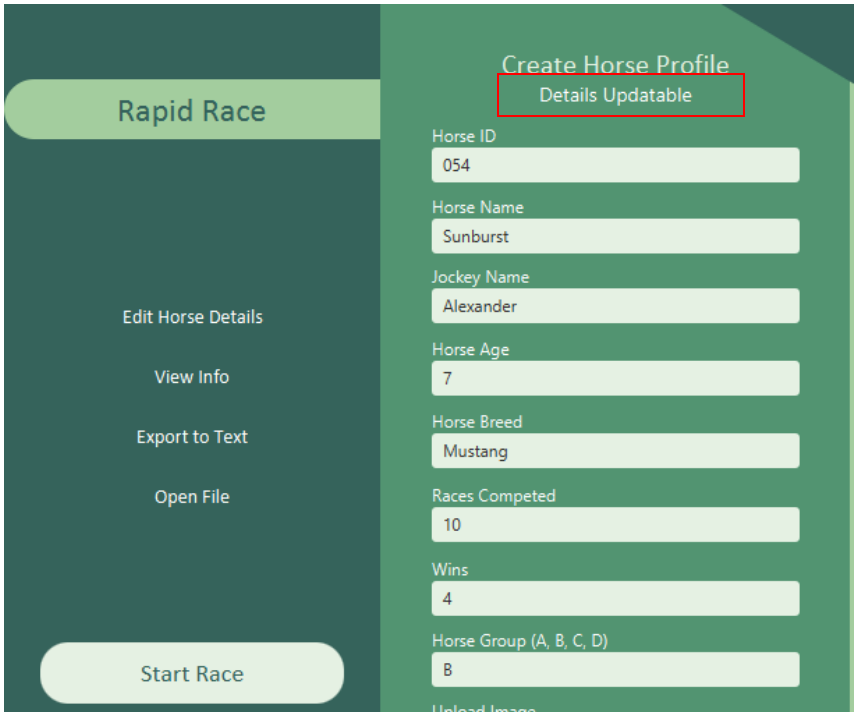
- Inputs: 100, Brando, Roy, **Seven**, Mustang, **Five, Two**, A.
- Expected Output: Alert displayed: "Please enter valid numbers for age, wins, and compete”
- Actual Output: Alert displayed: "Please enter valid numbers for age, wins, and compete”



Test case 7: Update horse details- Enter existing horse ID.

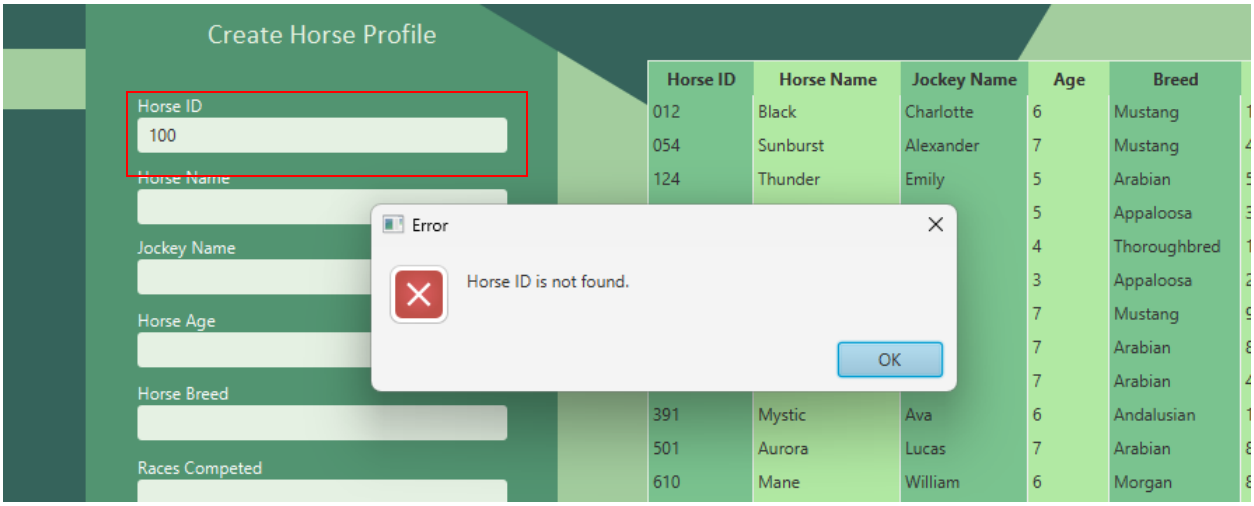
- Inputs: 054
- Expected Output: Fill all the text fields with the details that need to be updated.
- Actual Output: Alert displayed: Fill all the text fields with the details that need to be updated.





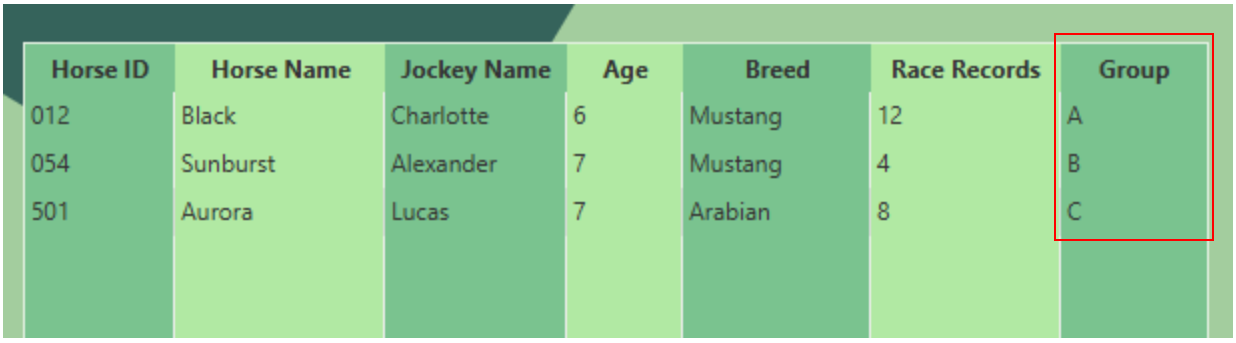
Test case 8: Update horse details- Enter invalid horse ID.

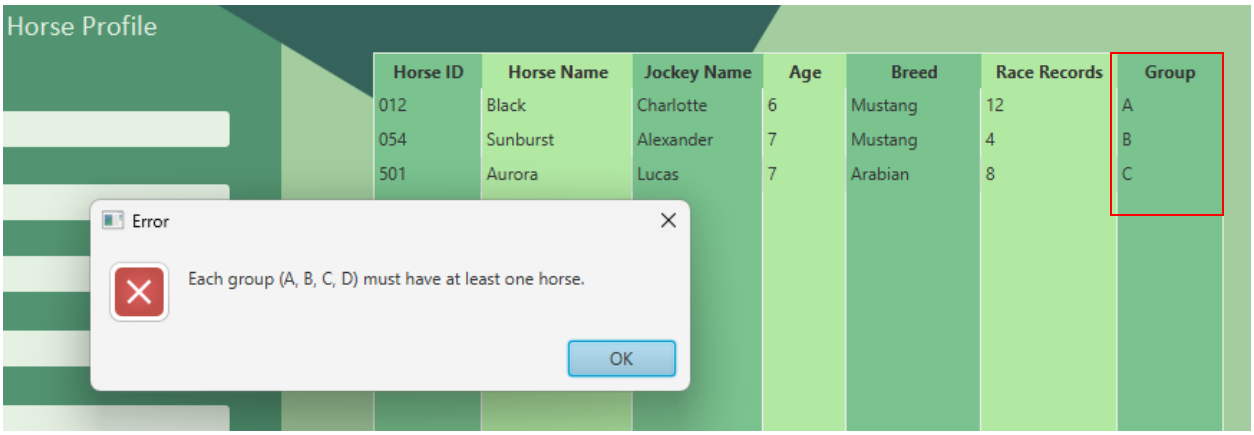
- Inputs: 100
- Expected Output: Alert displayed: "Horse ID is not found”
- Actual Output: Alert displayed: "Horse ID is not found”



Test case 9: Start Race- No at least one horse for each group.

- Expected Output: Alert displayed: "Each group (A, B, C, D) must have at least one horse”
- Actual Output: Alert displayed: "Each group (A, B, C, D) must have at least one horse”



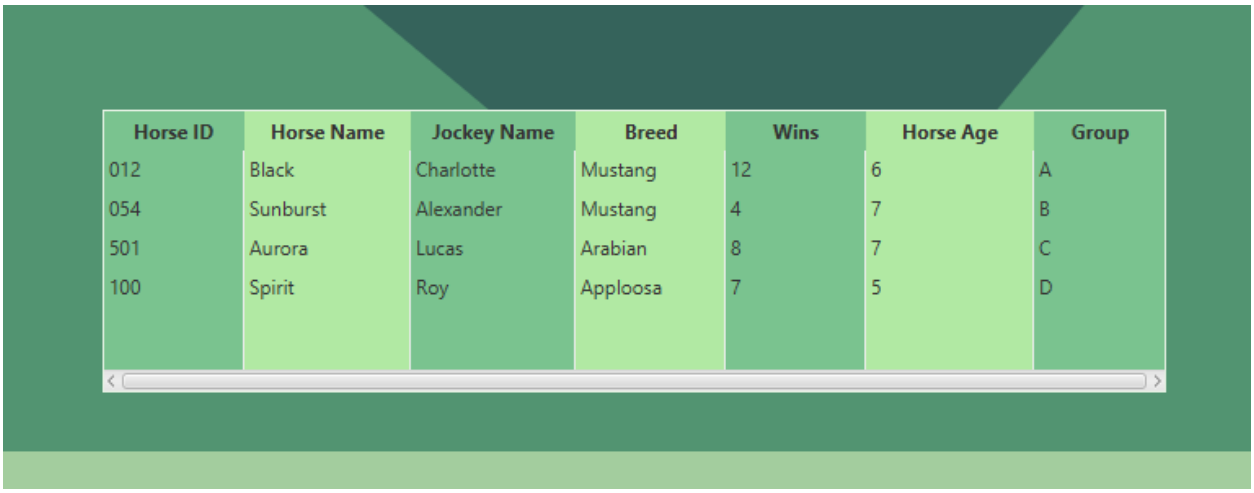


Test case 10: Start Race- All set correctly.

- Expected Output: Display Race Interface in new seane.
- Actual Output: Display Race Interface in new seane.

Test case 11: Select 4 random horses for finals.

- Expected Output: Fill the table accordingly with one horse for every group.
- Actual Output: Fill the table accordingly with one horse for every group.



Horse ID	Horse Name	Jockey Name	Breed	Wins	Horse Age	Group
012	Black	Charlotte	Mustang	12	6	A
054	Sunburst	Alexander	Mustang	4	7	B
501	Aurora	Lucas	Arabian	8	7	C
100	Spirit	Roy	Apploosa	7	5	D

Test case 12: Display winning horses and visualize them.

- Expected Output: Display the first, second, and third place horses with the time they spend, the bar chart visualized race time along with the horse name.
- Actual Output: Display the first, second, and third place horses with the time they spend, the bar chart visualized race time along with the horse name.

Place	Time	Horse ID	Horse Name	Jockey Name	Breed	Horse Age	Group
1	16.0	012	Black	Charlotte	Mustang	6	A
2	55.0	100	Spirit	Roy	Apploosa	5	D
3	60.0	054	Sunburst	Alexander	Mustang	7	B

5. Code Quality, Robustness and Maintainability

I. Readability

- Clear variables and function names like horseId, btnBackToMain, and selectHorseRace indicate their roles in the program.
- Proper indentation improves readability by showing the hierarchy and logic structure clearly.
- Comments are used to explain complex logic to improve understanding and provide value for providing context.

II. Reliability and correctness (Robustness)

- The presence of unit testing helps to verify the correctness of individual components of the code. The 'MainControllerTest' tests cover various scenarios such as ID checks, code behaviors under different conditions, etc.
- Error handling mechanisms help to handle unexpected behaviors of the code. For instance in the 'submit' method, if the entered horse ID is empty or not a valid integer, the system alerts the user about the issue. The code contains many numbers of these kinds of error-handling techniques. (Sierra, 2003)
- The application contains consistent behavior providing users with a reliable way to organize data. Sorting methods implemented in the system can be taken as examples.
- The code incorporates input validation techniques to ensure that only correct data is processed. This ensures the reliability of the data processing.

III. Efficiency and performance

- The code uses proper data structures to store and manipulate data efficiently. 'ObservableList' and 'TableView' are used to manage and display data in a tabular format.
- The graphical user interface elements are implemented efficiently to ensure flawless visualization and performance.
- The system contains efficient algorithms like custom-created selection sort algorithms for efficient sorting purposes.

IV. Maintainability

- Splitting code into reusable parts with explicit interfaces improves maintainability. This allows for simpler understanding, modification, and testing.
- For example 'HorseManager' class encapsulates functionality related to horse management.

- A well-organized structure, with clear logical grouping allows maintain the code sufficiently.
- The test-driven approach enhances the maintainability of the code by providing a safety platform for refactoring and enhancements.

V. Usage of OOP Concepts

1. Encapsulation

- The data encapsulates the data within classes such as ‘Horse’, ‘DataModel’, ‘HorseManager’, etc.
- ‘private’, and ‘public’ access modifiers used to access class members, ensuring data integrity and security.

2. Inheritance

- In the application codebase doesn’t have direct inheritance. However, inheritance can be seen when extending the ‘Horse’ class to create specialized horse types. (W3schools, 2020)

3. Polymorphism

- This concept is demonstrated through method overriding.
- For instance, ‘CustomSort’ and ‘CustomSortView’ override methods inherited from the parent class.

4. Abstraction

- Abstraction is used to hide the complexity of implementation and provide a simple form of interface to interact with objects. Hence this application is a small system, the codebase doesn’t have a direct abstraction concept. (geeksforgeeks, 2023)
- Even though the ‘DataModel’ class abstracts away data storage techniques, allowing interactive data without knowing internal details.

VI. SOLID Principles

i. S - Single Responsibility Principles (SRP)

- In the code each class has a distinct responsibility. For instance ‘Horse’, ‘DataModel’, ‘CustomSort’, etc all focus on specific functionality.
- This promotes better organization and improves code maintainability.

ii. O - Open-Closed Principle (OCP)

- In the code some functionalities allow for code extension without modification. For example ‘sortTable’ method in the ‘ViewHorseContrller’ class, extends the sorting technique without modifying the code. (Oloruntoba, 2021)

iii. L- Liskov Substitution Principle (LSP)

- Derived classes can be substitutable for their base type. Even though the code does not directly demonstrate the inheritance relationship, it follows the principle of substitutability.
- ‘Horse’ class properties are defined in a way that is compatible with derived classes.

iv. I - Interface Segregation Principle (ISP)

- While this method is commonly used for larger interfaces with many methods to break down interfaces into smaller parts. (Shukla, 2022)
- In the code ‘HorseManager’ class, the ‘submit’ method calls different methods based on specific conditions.

v. D - Dependency Inversion Principle (DIP)

- The code demonstrates this principle by relying on abstractions rather than concrete implementations.
- In the code the ‘submit’ method depends on abstract concepts like action execution and input validation.

6. Conclusion

In conclusion, the ‘Rapid Race’ application program demonstrates a well-developed solution for horse data management. The codebase has an admirable level of organization, with well-defined classes. The development of the application has been completed, providing a user-friendly interface for users to manage horse racing events.

Overall, it combines usability, functionality, and visual interface to deliver a user-engaging experience. Further, this implementation of JavaFX for horse racing management showcases effective data management and code quality.

Looking ahead, future enhancements include support for databases to store data permanently and add horse images as profile pictures. Furthermore, enhances the visuals to increase the user attraction.

Overall, ‘Rapid Race’ is a promising tool for racing devotees and industry professionals, with the potential to modernize and enhance the horse racing experience.

7. References

Amir, C., 2020. *JavaFX Scene Builder ,* s.l.: s.n.

geeksforgeeks, 2023. *geeksforgeeks.* [Online]
Available at: https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/
[Accessed April 2024].

Oloruntoba, S., 2021. *DigitalOcean.* [Online]
Available at: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design
[Accessed April 2024].

Shukla, S., 2022. *Medium.* [Online]
Available at: https://shirsh94.medium.com/top-100-flutter-interview-questions-efc4a96eb22c
[Accessed 18 April 2024].

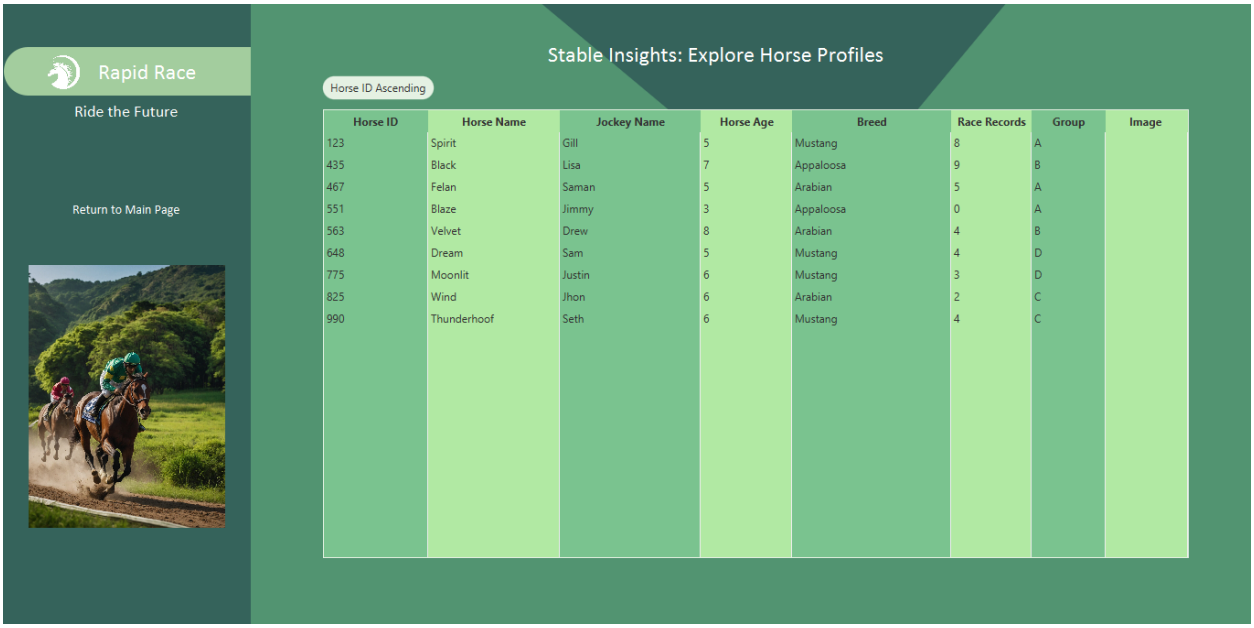
Sierra, K., 2003. *Head First Java.* Michigan: O'Reilly.

W3schools, 2020. *W3schools.* [Online]
Available at: https://www.w3schools.com/java/java_oop.asp
[Accessed April 2024].

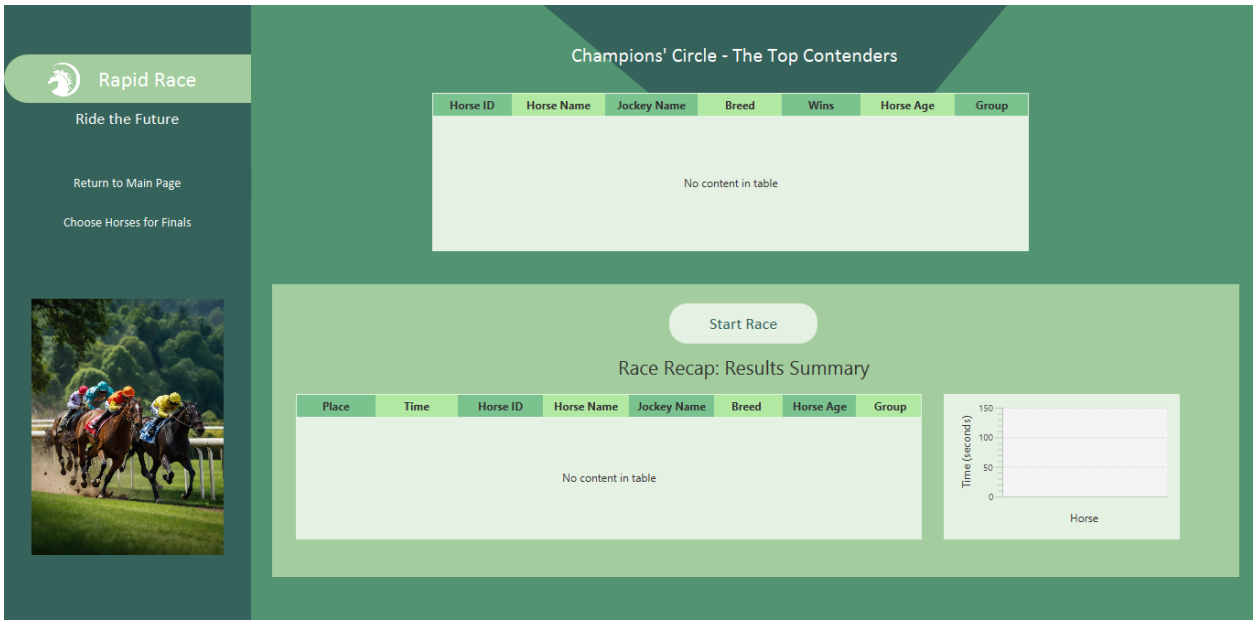
8. Appendices

User Interfaces

Main page GUI



View Horse Details GUI



Horse Race GUI

Horse Race GUI with Race Results

- End of Report -