OOP coursework

Initialising a local GIT repository to work on (Part 3):

This Section of the report focuses on some of the requirements which from Part 3 and how I meet them. The rest of Part 3 will be shown in other parts of the report (eg. sensible commit messages) and at Part 3 section at the end of the report.

After creating a local folder named “HorseRaceSimulator”, and adding folders “Part1” and “Part2” to it and after copying the existing .java files “Horse” and “Race” to the Part 1 folder, I initialised Git on the folder “HorseRaceSimulator”.

Initialising git:

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git init
Initialized empty Git repository in C:/Vasanth/Uni/Comp sci/HorseRaceSimulator/.git/
```

Listing contents of the folders:

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>dir /a
 Volume in drive C is Windows
 Volume Serial Number is A2A9-1DC0

 Directory of C:\Vasanth\Uni\Comp sci\HorseRaceSimulator

20/03/2024  04:39 PM    <DIR>          .
20/03/2024  04:32 PM    <DIR>          ..
20/03/2024  04:40 PM    <DIR>          .git
20/03/2024  04:22 PM    <DIR>          Part1
20/03/2024  04:22 PM    <DIR>          Part2
               0 File(s)              0 bytes
               5 Dir(s)  15,559,000,064 bytes free
```

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1>dir /a
 Volume in drive C is Windows
 Volume Serial Number is A2A9-1DC0

 Directory of C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1

20/03/2024  04:22 PM    <DIR>          .
20/03/2024  04:39 PM    <DIR>          ..
20/03/2024  04:22 PM             1,098 Horse.java
20/03/2024  04:22 PM             6,263 Race.java
               2 File(s)          7,361 bytes
               2 Dir(s)  15,550,427,136 bytes free
```

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2>dir /a
 Volume in drive C is Windows
 Volume Serial Number is A2A9-1DC0

 Directory of C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2

20/03/2024  04:22 PM    <DIR>          .
20/03/2024  04:39 PM    <DIR>          ..
               0 File(s)              0 bytes
               2 Dir(s)  15,550,246,912 bytes free

C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2>
```

Committing the changes of adding the files to the folder with appropriate commit message:

Git status at the start:

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Part1/

nothing added to commit but untracked files present (use "git add" to track)
```

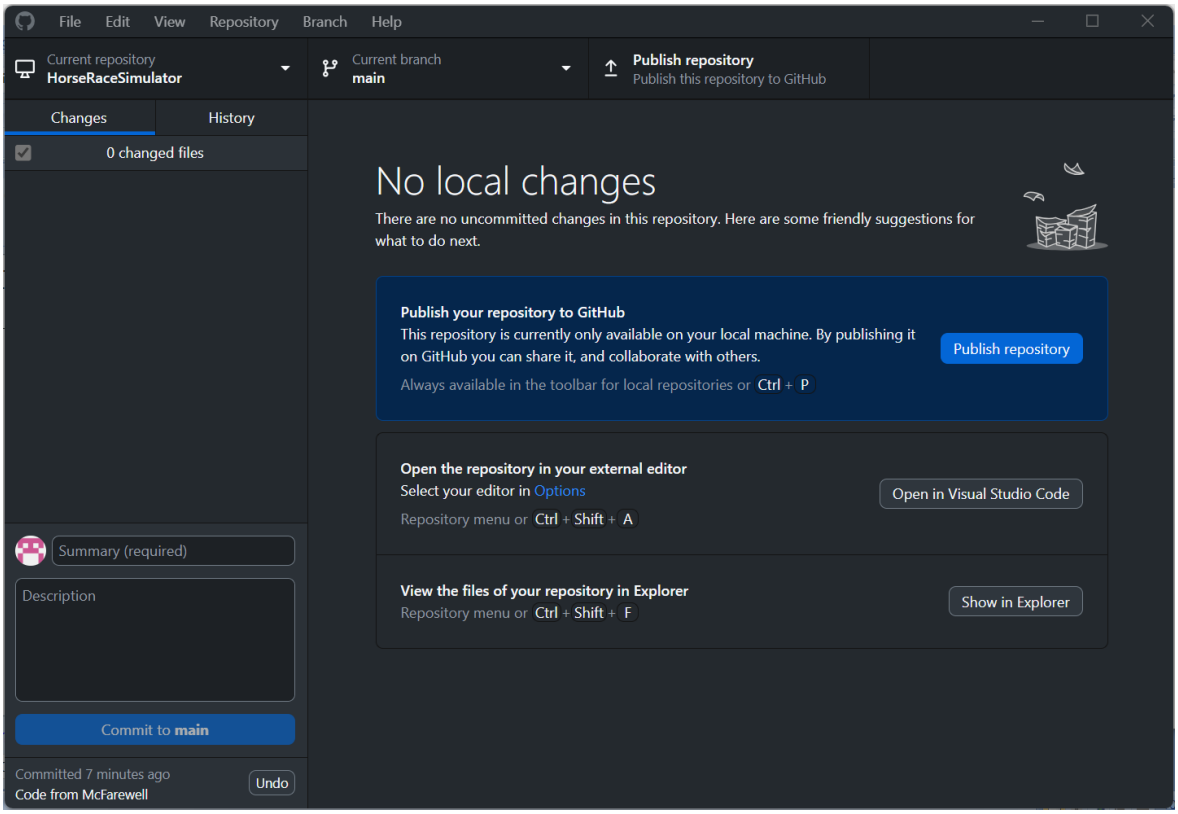
Committing the files:

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>git add Part1 Part2

C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   Part1/Horse.java
        new file:   Part1/Race.java


C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git commit Part1 -m "Code from McFarewell"
[main (root-commit) 906fdd8] Code from McFarewell
 2 files changed, 298 insertions(+)
 create mode 100644 Part1/Horse.java
 create mode 100644 Part1/Race.java
```

Git status after:

```
C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>git status
On branch main
nothing to commit, working tree clean
```

Adding the local repository to GitHub:



The rest of part 3 requirements will be illustrated at later sections of the report.

Part 1: A textual racing simulator

Section 1: the Horse class

A) Naming the class "Horse":

It is already named Horse from McFarwell's code, so I will not change it.

B) Making the required fields for the horse class:

I made the fields that were required, which were:

- The name of the horse – String.
- A single (Unicode) character used to represent the horse – char.
- The distance travelled by the horse as a whole number – int.
- A boolean indicating whether or not the horse has fallen – boolean.
- The confidence rating of the horse, represented as a decimal number between 0 and 1 – double.

Here is the screenshot of the code:

```
//Fields of class Horse

String name;
char icon;
int distance;
boolean fallen;
double confidence;
```

C) Making a Constructor method:

I followed the requirement to make a constructor method with the same signature as

`public Horse(char horseSymbol, String horseName, double horseConfidence)`

So I made one accordingly, here is a screenshot:

```
//Constructor of class Horse
/**
 * Constructor for objects of class Horse
 */
public Horse(char _icon, String _name, double _confidence)
{
    name=_name;//
    icon=_icon;//
    confidence=_confidence; // should be checked if valid or not before calling this method
    distance=0;
    fallen=false;
}
```

As pointed out in the comments, input validation should be done prior to calling this method.

Testing:

I made some other methods such as printout and testConstructor to test the constructor method Horse. Here are the screen shots:

```
/**
 * Method to test the public methods
 */
private static void printout (Horse horse){
    System.out.println("name: "+horse.name+" | icon: "+horse.icon+" | fallen: "+horse.fallen+" | distance: "+horse.distance+" | confidence: "+horse.confidence);
}

/**
 * Method to test the constructor method Horse
 */
private static void testConstructor(){
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    printout(test);
    // (expected:) name: testName | icon: A | fallen: false | distance: 0 | confidence: 0.5
}

/**
 * main method, currently used to test code
 */
Run | Debug
public static void main(String[] args) {
    testConstructor();
}
```

Here is the output:

```
name: testName | icon: A | fallen: false | distance: 0 | confidence: 0.5
```

So I believe the function operates as it should. (there was no need to test boundary/ erroneous data as the input needs to be validated outside the function).

D) Making the other public methods:

Fall():

```
/**
 * Method to make fallen= true
 */
public void fall()
{
    fallen=true;
    return;
}
```

Testing Fall():

```
/**
 * Method to test the fall() method
 */
private static void testFall(){
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    System.out.println(x:"Before: ");
    printout(test);
    System.out.println(x:"After: ");
    test.fall();
    printout(test);
    // (expected:) name: testName | icon: A | fallen: true | distance: 0 | confidence: 0.5
}
```

Output:

```
Before:
name: testName | icon: A | fallen: false | distance: 0 | confidence: 0.5
After:
name: testName | icon: A | fallen: true | distance: 0 | confidence: 0.5
```

Works as expected.

getConfidence():

```
/**
 * Method to get the confidence of the object
 * @return the confidence
 */
public double getConfidence()
{
    return this.confidence;
}
```

Testing for all the get methods are done together later.

getDistanceTravelled():

```java
/**
 * Method to get the distance travelled by far
 * @return the distance attribute of the object
 */
public int getDistanceTravelled()
{
    return this.distance;
}
```

getName():

```java
/**
 * Method to get the name of the horse
 * @return the name attribute of the object
 */
public String getName()
{
    return this.name;
}
```

getSymbol():

```java
/**
 * Method to return the icon/symbol of the horse
 * @return the icon attribute of the horse
 */
public char getSymbol()
{
    return this.icon;
}
```

Testing the get methods:

```java
Run | Debug
public static void main(String[] args) {
    // testing the get methods
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    System.out.println(test.getConfidence()); // expected: 0.5
    System.out.println(test.getDistanceTravelled()); //expected: 0
    System.out.println(test.getName());//expected: testName
    System.out.println(test.getSymbol());//expected: A

}
```

Output:

```
0.5
0
testName
A
```

This output was satisfactory as all the methods work as expected.

goBackToStart():

```java
/**
 * Method to set the distance to 0
 */
public void goBackToStart()
{
    this.distance = 0;
    return;
}
```

The testing of this method will be done after/along with the moveForward method.

hasFallen():

```java
/**
 * Method to see if a horse has fallen or not
 * @return returns the fallen attribute
 */
public boolean hasFallen()
{
    return this.fallen;
}
```

Testing:

```java
/**
 * main method, currently used to test code
 */
Run | Debug
public static void main(String[] args) {
    // testing the hasfallen method
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    System.out.println(test.hasFallen());// expected: false
    test.fall();
    System.out.println(test.hasFallen());// expected: true
}
```

Output:

```
false
true
```

The output was satisfactory as it was what the method was expected to do.

moveFoward():

```java
/**
 * Method to increment the distance moved by the horse by 1
 */
public void moveForward()
{
    this.distance+=1;
    return;
}
```

Testing:

```java
/**
 * main method, currently used to test code
 */
Run | Debug
public static void main(String[] args) {
    // testing the moveForward and goBackToStart methods
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    System.out.println(test.getDistanceTravelled());// expected: 0
    test.moveForward();
    System.out.println(test.getDistanceTravelled());// expected: 1
    for (int i=0;i<10;i++){test.moveForward();}
    System.out.println(test.getDistanceTravelled());// expected: 11

    test.goBackToStart();
    System.out.println(test.getDistanceTravelled());// expected: 0
}
```

Output:

```
0
1
11
0
```

The output is as expected of the method so we can conclude that it works as intended.

setConfidence():

```java
/**
 * Sets the confidence of the horse to the given value. Input needs to be validated before calling this method
 * @param newConfidence the new value for the confidence attribute
 */
public void setConfidence(double newConfidence)
{
    this.confidence=newConfidence;
    return;
}
```

The testing for this method is done along with the setSymbol() method

setSymbol():

```java
/**
 * Sets the icon of the horse to the given value, input needs to be validated before this method is called
 * @param newSymbol the new icon of the horse
 */
public void setSymbol(char newSymbol)
{
    this.icon=newSymbol;
}
```

Testing:

```java
/**
 * main method, currently used to test code
 */
Run | Debug
public static void main(String[] args) {
    // testing the setConfidence and setSymbol methods
    Horse test = new Horse(_icon:'A', _name:"testName", _confidence:0.5);
    printout(test);
    test.setConfidence(newConfidence:0.99);
    test.setSymbol(newSymbol:'@');
    printout(test);
    // the icon and confidence attributes should change to '@' and 0.99 respectively
}
```

The printout method here is the same as the one used to test the constructor method, see the report on that to see its code.

Output:

```
name: testName | icon: A | fallen: false | distance: 0 | confidence: 0.5
name: testName | icon: @ | fallen: false | distance: 0 | confidence: 0.99
```

The output is as expected so the methods work as intended.

E) Encapsulation:

To ensure that I encapsulated the fields of the class Horse so that it can only be acceptable values, I made all the fields/ attributes of the class private so that then they can only be accessed via the accessor/ mutator methods. I will also identify which of the methods from part D are accessor and mutator methods.

```java
// making the fields private for encapsulation
private String name;
private char icon;
private int distance;
private boolean fallen;
private double confidence;
```

Accessor Methods:

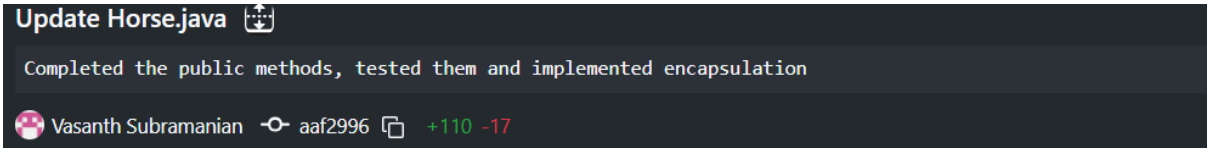
- `getConfidence()` [accesses the field confidence]
- `getDistanceTraveled()` [accesses the field distance]
- `getName()` [accesses the field name]

- `getSymbol()` [accesses the field icon]

Mutator Methods:

- `fall()` [modifies the field fallen]
- `goBackToStart()` [modifies the field distance]
- `moveForward()` [modifies the field distance]
- `setConfidence()` [modifies the field confidence]
- `setSymbol()` [modifies the field icon]

This marks the end of section 1 of part 1, so I will commit this to Git with a suitable message:



Section 2: the Race class

A) Changing the startRace() method to display winner:

To achieve this I introduced a new local variable to the method startRace, type Horse to store the winning horse. It is initialised to null, and upon race completion gets a value of the horse which won the race. Here is the screenshot of my implementation:

```java
public void startRace() {
    // Declare a local variable to store the winner
    Horse winner = null;

    // Reset all the lanes (all horses not fallen and back to 0).
    lane1Horse.goBackToStart();
    lane2Horse.goBackToStart();
    lane3Horse.goBackToStart();

    // Continue the race until a horse wins
    while (winner == null) {
        // Move each horse
        moveHorse(lane1Horse);
        moveHorse(lane2Horse);
        moveHorse(lane3Horse);

        // Print the race positions
        printRace();

        // Check if any of the horses has won the race
        if (raceWonBy(lane1Horse)) {
            winner = lane1Horse;
        } else if (raceWonBy(lane2Horse)) {
            winner = lane2Horse;
        } else if (raceWonBy(lane3Horse)) {
            winner = lane3Horse;
        }

        // Wait for 100 milliseconds
        try {
            TimeUnit.MILLISECONDS.sleep(timeout:100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    // Display the name of the winning horse upon race completion
    System.out.println("And the winner is " + winner.getName());
}
```

And here is screenshots of testing/ running it:

Code:

```java
/**
 * main Method of the Race class, currently used to test the race methods
 */
Run | Debug
public static void main(String[] args) {
    Race testRace = new Race(distance:10);
    testRace.addHorse(new Horse(_icon:'A',_name:"HorseOne",_confidence:1),laneNumber:1);
    testRace.addHorse(new Horse(_icon:'B',_name:"HorseTwo",_confidence:0.5),laneNumber:2);
    testRace.addHorse(new Horse(_icon:'C',_name:"HoresThree",_confidence:0.25),laneNumber:3);
    testRace.startRace();
}
```

Outputs:

```
=============
|    ?      |
|           B|
|    ?      |
=============
And the winner is HorseTwo
```

```
=============
|           A|
|     B     |
|  C        |
=============
And the winner is HorseOne
```

```
=============
|   ?       |
|           B|
|     C     |
=============
And the winner is HorseTwo
```

Many other tests were done, and it passed all the tests and displayed the winners correctly.

B) Improving the Race class:

To improve the class, first we need to spot issues with the existing class and any areas of improvements to focus on. Here are some I could find:

1. **Continuous Printing and Clearing Terminal**: continuously printing the race positions without clearing the terminal can clutter the output. Clearing the terminal before printing each race iteration would provide a cleaner display.

2. **Handling of Fallen Horses**: If all horses fall during the race, the race will never end. Implementing a mechanism to handle this scenario, such as checking if all horses are fallen or setting a maximum number of iterations or providing a way to manually end the race, would prevent an infinite loop.

3. **Race Length and Horse Attributes**: These values are predetermined and do not change during the runtime of the program right now. Allowing the user to enter them and making them customisable would add more realism to the simulation.

4. **Randomness in Horse Movement**: The **moveHorse()** method currently uses **Math.random()** to determine whether a horse moves forward, or falls based on its confidence level. This randomness can lead to unpredictable races and may not accurately reflect a horse's behavior. Implementing a more sophisticated algorithm that considers factors such as horse skill, track conditions, and competitors' performances could improve realism.

5. **Additional Horse statistics:** Giving the user other Horse statistics in real time like the time taken to complete the race, or average speed, or expected place etc. will be a good improvement to make on the current version.

6. **Handling of Tie Races**: Currently, the race stops as soon as one horse reaches the end, declaring it the winner. In a real race scenario, there could be tie races where multiple horses reach the finish line simultaneously. Implementing logic to handle tie race would improve the simulation's accuracy.

7. **Input Validation**: Currently, there is no validation in the **addHorse()** method to ensure that the lane number provided is valid (1, 2, or 3). Adding input validation would prevent adding horses to nonexistent lanes and provide better error handling. This can be either done in the Horse class or the Race class.
8. **Error Handling**: The code lacks proper error handling mechanisms. For example, if a horse with 0 confidence is added to the race, it will never move. Implementing error handling to notify users of invalid inputs or unexpected behaviour would improve the robustness of the simulation.
9. **Scalability**: The **Race** class is designed for a fixed number of three lanes. Enhancing the class to support variable numbers of lanes would increase its scalability and flexibility, allowing for races with more or fewer horses. Also, currently only one race occurs at once, we can perhaps simulate a “Championship” by making multiple races occur at the same time and track results to determine the winner.
10. **Performance Optimization**: The current implementation uses a simple loop to repeatedly move horses forward until one wins. For large race lengths or a large number of races, this approach may be inefficient. Optimizing the algorithm or introducing concurrency (e.g., using threads) could improve performance.

Implementing these changes (where possible):

1. Continuous Printing and Clearing Terminal:

I implemented a new method **clearTerminal()** to clear the screen of the terminal so that the display of the race looks nicer. Here is the code:

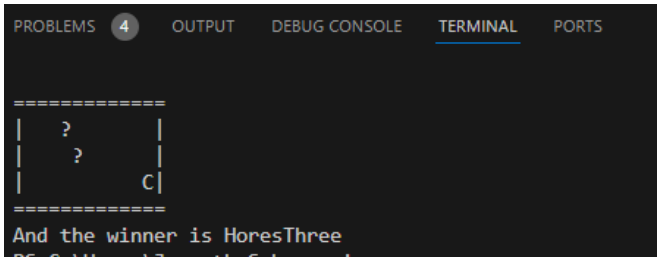
```java
/**
 * Clears the terminal window.
 */
private void clearTerminal() {
    try {
        // Clearing terminal based on the operating system
        if (System.getProperty(key:"os.name").contains(s:"Windows")) {
            new ProcessBuilder(...command:"cmd", "/c", "cls").inheritIO().start().waitFor();
        } else {
            new ProcessBuilder(...command:"bash", "-c", "clear").inheritIO().start().waitFor();
        }
    } catch (IOException | InterruptedException e) {
        // Handle exceptions
        System.out.println(x:"Error: Failed to clear terminal.");
    }
}
```

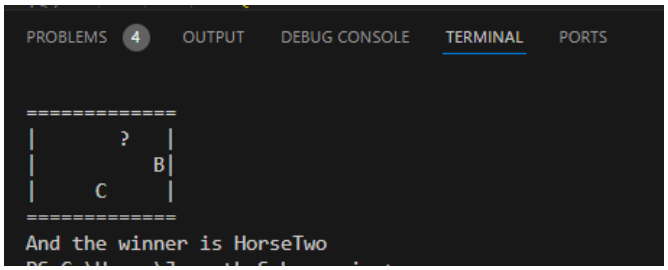
Changing the **startRace()** method by adding a line clearTerminal(); at the start of the whole loop:

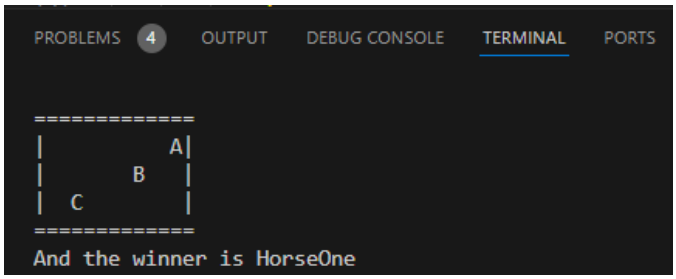
```java
public void startRace() {
    // Reset all the lanes (all horses not fallen and back to 0).
    lane1Horse.goBackToStart();
    lane2Horse.goBackToStart();
    lane3Horse.goBackToStart();

    // Continue the race until a horse wins
    while (winner == null) {
        //Clearing terminal
        clearTerminal();

        // Move each horse
        moveHorse(lane1Horse);
        moveHorse(lane2Horse);
        moveHorse(lane3Horse);

        // Print the race positions
        printRace();

        // Check if any of the horses has won the race
        if (raceWonBy(lane1Horse)) {
            winner = lane1Horse;
        } else if (raceWonBy(lane2Horse)) {
            winner = lane2Horse;
        } else if (raceWonBy(lane3Horse)) {
            winner = lane3Horse;
        }
```

The rest of the method is not shown here but it has not been changed form part 1

It running:







As you can see, only one race is printed at the terminal at the same time. Makes the output more readable.

2. Handling of Fallen Horses:

To make sure that we are not stuck in a dead lock when all the horses fall, we need to make sure we can handle it when it happens. I made a new method return type boolean called allHorsesFallen() that can be used in the startRace method to solve this issue.

```java
/**
 * Determines if all horses have fallen.
 *
 * @return true if all horses have fallen, false otherwise
 */
private boolean allHorsesFallen() {
    return lane1Horse.hasFallen() && lane2Horse.hasFallen() && lane3Horse.hasFallen();
}
```

Changed startRace method:

```java
public void startRace() {
    // Declare a local variable to store the winner
    Horse winner = null;

    // Reset all the lanes (all horses not fallen and back to 0).
    lane1Horse.goBackToStart();
    lane2Horse.goBackToStart();
    lane3Horse.goBackToStart();

    // Continue the race until a horse wins / all horses fall
    while (winner == null && !allHorsesFallen()) {
        //Clearing terminal
        clearTerminal();

        // Move each horse
        moveHorse(lane1Horse);
        moveHorse(lane2Horse);
        moveHorse(lane3Horse);

        // Print the race positions
        printRace();

        // Check if any of the horses has won the race
        if (raceWonBy(lane1Horse)) {
            winner = lane1Horse;
        } else if (raceWonBy(lane2Horse)) {
            winner = lane2Horse;
        } else if (raceWonBy(lane3Horse)) {
            winner = lane3Horse;
        }

        // Wait for 100 milliseconds
        try {
            TimeUnit.MILLISECONDS.sleep(timeout:100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    // Finding winner when all horses fall by distance travelled
    if (allHorsesFallen()) {
        winner=lane1Horse;
        if (lane2Horse.getDistanceTravelled() > winner.getDistanceTravelled()) {
            winner = lane2Horse;
        }

        if (lane3Horse.getDistanceTravelled() > winner.getDistanceTravelled()) {
            winner = lane3Horse;
        }
    }

    // Display the name of the winning horse upon race completion
    if (allHorsesFallen()) {
        System.out.println("All the horses fell! the winner with the longest distance is: "+ winner.getName());
    }
    else System.out.println("And the winner is " + winner.getName());
}
```

The changes made are the conditions of the while loop, and how the winner is chosen when all the horses fall.

Here is the testing code for this improvement: (made confidence 1 and length of race longer to get more falls)

```java
/**
 * main Method of the Race class, currently used to test the race methods
 */
Run | Debug
public static void main(String[] args) {
    Race testRace = new Race(distance:20);
    testRace.addHorse(new Horse(_icon:'A',_name:"HorseOne",_confidence:1),laneNumber:1);
    testRace.addHorse(new Horse(_icon:'B',_name:"HorseTwo",_confidence:1),laneNumber:2);
    testRace.addHorse(new Horse(_icon:'C',_name:"HoresThree",_confidence:1),laneNumber:3);
    testRace.startRace();
}
```

Output:

```
======================
|   ?                |
|    ?               |
| ?                  |
======================
All the horses fell! the winner with the longest distance is: HorseTwo
```

```
 ======================
 |    ?               |
 |     ?              |
 |         ?          |
 ======================
 All the horses fell! the winner with the longest distance is: HoresThree
```

```
======================
|       ?            |
|    ?               |
|   ?                |
======================
All the horses fell! the winner with the longest distance is: HorseOne
```

This improvement performs as expected.

3. Race Length and Horse Attributes:

To make these customisable, I changed the main method in the class, which was previously used for testing, to have it take user input to make the race and the horses. Here is the new main method:

```java
/**
 * main Method of the Race class, Takes user input to create and start a race
 */
Run | Debug
public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    // getting race length
    System.out.print(s:"Enter the race length: ");
    int raceLength = scanner.nextInt();
    Race testRace = new Race(raceLength);

    // adding the horses from input to the race
    for (int i = 1; i <= 3; i++) {
        System.out.print("Enter name for horse " + i + ": ");
        String horseName = scanner.next();
        System.out.print("Enter symbol for horse " + i + ": ");
        char horseSymbol = scanner.next().charAt(index:0);
        System.out.print("Enter confidence for horse " + i + ": ");
        double horseConfidence = scanner.nextDouble();

        testRace.addHorse(new Horse(horseSymbol, horseName, horseConfidence), i);
    }

    // Start the race
    testRace.startRace();
    scanner.close();
}
```

Output:

```
Enter the race length: 20
Enter name for horse 1: Alex
Enter symbol for horse 1: A
Enter confidence for horse 1: 0.7
Enter name for horse 2: Bob
Enter symbol for horse 2: B
Enter confidence for horse 2: 0.6
Enter name for horse 3: Charlotte
Enter symbol for horse 3: C
Enter confidence for horse 3: 0.8
```

```
======================
|         ?          |
|          ?         |
|  ?                 |
======================
All the horses fell! the winner with the longest distance is: Bob
```

```
Enter the race length: 5
Enter name for horse 1: X
Enter symbol for horse 1: £
Enter confidence for horse 1: 0.9
Enter name for horse 2: Y
Enter symbol for horse 2: $
Enter confidence for horse 2: 0.8
Enter name for horse 3: Z
Enter symbol for horse 3: &
Enter confidence for horse 3: 0.85
```

```
========
| ?    |
| ?    |
|     &|
========
And the winner is Z
```

```
Enter the race length: 50
Enter name for horse 1: SpongeBob
Enter symbol for horse 1: #
Enter confidence for horse 1: 0.8
Enter name for horse 2: Patrick
Enter symbol for horse 2: *
Enter confidence for horse 2: 0.5
Enter name for horse 3: Squidward
Enter symbol for horse 3: @
Enter confidence for horse 3: 0.6
```

```
|?                                                |
|               ?                                 |
|       ?                                         |
All the horses fell! the winner with the longest distance is: Patrick
```

4. Randomness in Horse Movement:

From the testing I have done throughout this section, I feels like the horses fall too often, and movement algorithm feels a little clunky. So, to improve this I changed the moveHorse() method. Here is the new moveHorse() method:

```java
/**
 * Randomly make a horse move forward or fall depending
 * on its confidence rating
 * A fallen horse cannot move
 *
 * @param theHorse the horse to be moved
 */
private void moveHorse(Horse theHorse) {
    // If the horse has fallen
    if (theHorse.hasFallen()) {
        return;
    }
    //If it has already finished
    if(theHorse.getDistanceTravelled()==raceLength){
        return;
    }

    // Calculate the probability of movement based on confidence
    double confidenceFactor = 0.2 + 0.8 * theHorse.getConfidence();

    // Calculate additional randomness based on track conditions or other factors, can be implemented better in the future
    double additionalRandomness = Math.random() * 0.2 - 0.1; // Adjust range as needed

    // Combined movement probability
    double movementProbability = confidenceFactor + additionalRandomness;

    // Generate a random number to determine movement
    double randomValue = Math.random();

    // Move the horse forward if random value falls within the movement probability
    if (randomValue < movementProbability) {
        theHorse.moveForward();
    } else {
        // Generate a random number to determine if the horse falls
        double fallProbability = 0.12 * theHorse.getConfidence() * theHorse.getConfidence();
        if (Math.random() < fallProbability) {
            theHorse.fall();
        }
    }
}
```

Testing:

```
Enter the race length: 50
Enter name for horse 1: SpongeBob
Enter symbol for horse 1: #
Enter confidence for horse 1: 0.8
Enter name for horse 2: Patrick
Enter symbol for horse 2: *
Enter confidence for horse 2: 0.5
Enter name for horse 3: Squidward
Enter symbol for horse 3: @
Enter confidence for horse 3: 0.65
```

```
==================================================
|        ?                                       |
|                                      *         |
|                                             @|
==================================================
And the winner is Squidward
```

As you can see this same test when done in the previous method resulted in all horses falling (even with higher confidence values), so this model seems good. I was tested a few times to adjust the values in the algorithm, which I will not show here.

5. Additional Statistics & 6. Handling ties:

I implemented these two changes at the same time to the Race class. Firstly, I added a new private attribute to the class Race which was a Map:

```
private Map<Horse,Double> results= new HashMap<>();
```

This holds the Horse and the time taken for the horse to complete which is stored as a double. The Map will be sorted in the future as a LinkedHashMap but for now we will just use this to provide the real time statistics for the horses.

To do this I changed the printRace method to be this:

```
private void printRace()
{
    System.out.print(c:'\u000C');  //clear the terminal window

    multiplePrint(aChar:'=',raceLength+3); //top edge of track
    System.out.println();

    printLane(lane1Horse);
    System.out.print(" "+lane1Horse.getName()+" Confidence: "+lane1Horse.getConfidence()+" ");
    if(lane1Horse.hasFallen()) System.out.print(s:"Fallen ");
    else if(results.containsKey(lane1Horse)) System.out.print("Finished, time: "+Math.round(results.get(lane1Horse)*100.0)/100.0);
    System.out.println();

    printLane(lane2Horse);
    System.out.print(" "+lane2Horse.getName()+" Confidence: "+lane2Horse.getConfidence()+" ");
    if(lane2Horse.hasFallen()) System.out.print(s:"Fallen");
    else if(results.containsKey(lane2Horse)) System.out.print("Finished, time: "+Math.round(results.get(lane2Horse)*100.0)/100.0);
    System.out.println();

    printLane(lane3Horse);
    System.out.print(" "+lane3Horse.getName()+" Confidence: "+lane3Horse.getConfidence()+" ");
    if(lane3Horse.hasFallen()) System.out.print(s:"Fallen");
    else if(results.containsKey(lane3Horse)) System.out.print("Finished, time: "+Math.round(results.get(lane3Horse)*100.0)/100.0);
    System.out.println();;

    multiplePrint(aChar:'=',raceLength+3); //bottom edge of track
    System.out.println();
}
```

This now prints the name of the horse, its confidence value, and if it finishes the time it took to finish and if I fall it prints “Fallen” all next to the lane of the horse displayed.

Output of this method:

```
======================
|                    A| Alice Confidence: 0.8 Finished, time: 2.1
|                    B| Bob Confidence: 0.7 Finished, time: 2.4
|                   C  | Carrie Confidence: 0.5
======================
```

As you can see once a horse finishes the race it shows their time, before it only shows their name and confidence.

```
========================================
|                                ?      | SpongeBob Confidence: 0.9 Fallen
|                                     @| Squidward Confidence: 0.9 Finished, time: 4.0
|                                     *| Patrick Confidence: 0.4 Finished, time: 6.7
========================================
```

As you can see it also displays “Fallen” if the horse on that lane has fallen.

Now to handle ties, I used the new system of storing the time the horse's finish to give their placements. Here is how I tracked the time in the startRace class, and how I put the horses in the map if the had fallen or finished:

```java
public void startRace() {
    // Reset all the lanes (all horses not fallen and back to 0).
    lane1Horse.goBackToStart();
    lane2Horse.goBackToStart();
    lane3Horse.goBackToStart();

    double time = 0.0; // Initialize time

    double fall= -1.0;

    // Continue the race until all horses finish or fall
    while (!allHorsesFinishedOrFallen()) {
        // Clear the terminal
        clearTerminal();

        //intial fallen values
        boolean fallen1 = lane1Horse.hasFallen();
        boolean fallen2 = lane2Horse.hasFallen();
        boolean fallen3 = lane3Horse.hasFallen();

        //initial "has finished" values
        boolean hasFinished1 = raceWonBy(lane1Horse);
        boolean hasFinished2 = raceWonBy(lane2Horse);
        boolean hasFinished3 = raceWonBy(lane3Horse);

        // Move each horse
        moveHorse(lane1Horse);
        moveHorse(lane2Horse);
        moveHorse(lane3Horse);

        // Print the race positions and horse confidence
        printRace();

        // putting to map if the horses just fell
        if(fallen1==false && lane1Horse.hasFallen()==true){
            results.put(lane1Horse,fall--);
        }
        if(fallen2==false && lane2Horse.hasFallen()==true){
            results.put(lane2Horse,fall--);
        }
        if(fallen3==false && lane3Horse.hasFallen()==true){
            results.put(lane3Horse,fall--);
        }

        //putting to map if horses just finished
        if(hasFinished1==false && raceWonBy(lane1Horse)==true){
            results.put(lane1Horse,time);
        }
        if(hasFinished2==false && raceWonBy(lane2Horse)==true){
            results.put(lane2Horse,time);
        }
        if(hasFinished3==false && raceWonBy(lane3Horse)==true){
            results.put(lane3Horse,time);
        }

        // Increment time by 0.1 seconds
        time += 0.1;

        // Wait for 100 milliseconds
        try {
            TimeUnit.MILLISECONDS.sleep(timeout:100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    clearTerminal();
    printRace();

    results=sortMap(results);
    System.out.println(x:"Results: ");
    for (Map.Entry<Horse, Double> entry : results.entrySet()) {
        if(entry.getValue()>0)
        System.out.println(entry.getKey().getName() + ": " + Math.round(entry.getValue()*100.0)/100.0);
        else System.out.println(entry.getKey().getName() + ": DNF");
    }

}
```

As you can see I implemented some other helper methods to achieve this, like allHorsesFinishedOrFallen() and sortMap(). Here are their implementations:

```java
/**
 * Check if all horses have finished the race or fallen.
 *
 * @return true if all horses have finished or fallen, false otherwise
 */
private boolean allHorsesFinishedOrFallen() {
    boolean horse1=lane1Horse.getDistanceTravelled() == raceLength || lane1Horse.hasFallen();
    boolean horse2=lane2Horse.getDistanceTravelled() == raceLength || lane2Horse.hasFallen();
    boolean horse3=lane3Horse.getDistanceTravelled() == raceLength || lane3Horse.hasFallen();
    return horse1 && horse2 && horse3;
}
```

```java
/**
 * Sort the given map to order the competitors
 *
 * @param map the map we are going to sort, which has the data on the competitors
 * @return returns the sorted map
 */
public static Map<Horse, Double> sortMap(Map<Horse, Double> map) {
    // Convert Map to List of Map.Entry objects
    List<Map.Entry<Horse, Double>> list = new ArrayList<>(map.entrySet());

    // Sort the list based on values
    Collections.sort(list, new Comparator<Map.Entry<Horse, Double>>() {
        @Override
        public int compare(Map.Entry<Horse, Double> o1, Map.Entry<Horse, Double> o2) {
            double value1 = o1.getValue();
            double value2 = o2.getValue();

            // Sort normal values in ascending order
            if (value1 >= 0 && value2 >= 0) {
                return Double.compare(value1, value2);
            }
            // Sort negative values in ascending order
            else if (value1 < 0 && value2 < 0) {
                return Double.compare(value1, value2);
            }
            // Place negative values at the end
            else {
                return value1 >= 0 ? -1 : 1;
            }
        }
    });

    // Create a new LinkedHashMap to preserve the insertion order
    Map<Horse, Double> sortedMap = new LinkedHashMap<>();
    for (Map.Entry<Horse, Double> entry : list) {
        sortedMap.put(entry.getKey(), entry.getValue());
    }

    return sortedMap;
}
```

The negative values mentioned here are the values put in the map if the Horse has fallen. We rank the time taken in ascending order for the horses that finished first, then at the end we put the horses that did not finish, in the order of how far they travelled before falling.

Output:

```
=======================================
|                              ?      | SpongeBob Confidence: 0.9 Fallen
|                                    @| Squidward Confidence: 0.9 Finished, time: 4.0
|                                    *| Patrick Confidence: 0.4 Finished, time: 6.7
=======================================
Results:
Squidward: 4.0
Patrick: 6.7
SpongeBob: DNF
```

```
=============================================================================================
|      ?                                                                      | Goku Confidence: 0.9 Fallen
|                                        ?                                    | Vegeta Confidence: 0.8 Fallen
|                                                                           F| Freiza Confidence: 0.85 Finished, time: 10.5
=============================================================================================
Results:
Freiza: 10.5
Vegeta: DNF
Goku: DNF
```

Many more tests were run to see if the sorting was working as intentional, which I will not show here.

7. Input Validation & 8. Error Handling:

The only user Input that I take is in the main method, where I ask the user to give the specifications for the race that is to be simulated. So, I added input validation and Error handling using exceptions, and while loops that ask the user for a valid value until it gets one. Here is the new main method:

```java
/**
 * main Method of the Race class, Takes user input to create and start a race
 */
Run | Debug
public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);

    // Getting race length with input validation
    int raceLength;
    while (true) {
        try {
            System.out.print(s:"Enter the race length: ");
            raceLength = scanner.nextInt();
            if (raceLength <= 0) {
                throw new IllegalArgumentException(s:"Race length must be a positive integer.");
            }
            break; // Break out of the loop if input is valid
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid input. Please enter a valid integer.");
            scanner.nextLine(); // Clear the input buffer
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }

    Race testRace = new Race(raceLength);
```

```java
    // Adding the horses from input to the race
    for (int i = 1; i <= 3; i++) {
        String horseName;
        char horseSymbol;
        double horseConfidence;

        // Getting horse name
        System.out.print("Enter name for horse " + i + ": ");
        horseName = scanner.next();

        // Getting horse symbol
        System.out.print("Enter symbol for horse " + i + ": ");
        horseSymbol = scanner.next().charAt(index:0);

        // Getting horse confidence with input validation
        while (true) {
            try {
                System.out.print("Enter confidence for horse " + i + ": ");
                horseConfidence = scanner.nextDouble();
                if (horseConfidence < 0 || horseConfidence > 1) {
                    throw new IllegalArgumentException(s:"Confidence must be a number between 0 and 1.");
                }
                break; // Break out of the loop if input is valid
            } catch (InputMismatchException e) {
                System.out.println(x:"Invalid input. Please enter a valid number.");
                scanner.nextLine(); // Clear the input buffer
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }

        // Adding horse to the race
        testRace.addHorse(new Horse(horseSymbol, horseName, horseConfidence), i);
    }

    // Start the race
    testRace.startRace();
    scanner.close();
}
```

Here is the output:

```
Enter the race length: abcd
Invalid input. Please enter a valid integer.
Enter the race length: -12
Race length must be a positive integer.
Enter the race length: 20
Enter name for horse 1: abcd
Enter symbol for horse 1: a
Enter confidence for horse 1: 3.2
Confidence must be a number between 0 and 1.
Enter confidence for horse 1: 0.1
Enter name for horse 2: hdj
Enter symbol for horse 2: kajd
Enter confidence for horse 2: 0.9
Enter name for horse 3: saj
Enter symbol for horse 3: i
Enter confidence for horse 3: -0.1
Confidence must be a number between 0 and 1.
Enter confidence for horse 3: 0.3
```

```
======================
|                    a| abcd Confidence: 0.1 Finished, time: 4.1
|  ?                   | hdj Confidence: 0.9 Fallen
|                    i| saj Confidence: 0.3 Finished, time: 3.1
======================
Results:
saj: 3.1
abcd: 4.1
hdj: DNF
```

It works as intended. In the future we could be stricter on some parameters such as the race length, as in testing on my monitor anything over 200 the lines overflow and the display looks illegible.

9. **Scaling the simulation to more lanes/races:**

Firstly, I implemented the race having multiple lanes. To do this I got rid of the attributes lane1Horse, lane2Horse and lane3Horse and instead had a new attribute horses which was an List of Horse objects, along with another new attribute which was the number of lanes:

```
private List<Horse> horses = new ArrayList<>();
private final int numberOfLanes;
```

This meant that many methods had to be changed to work with this implementation, so I changed the following methods:

Race //the constructor method

```
/**
 * Constructor for objects of class Race
 * Initially there are no horses in the lanes
 *
 * @param distance the length of the racetrack (in metres/yards...)
 * @param lanes the number of lanes in the race
 */
public Race(int distance,int lanes)
{
    // initialise instance variables
    raceLength = distance;
    numberOfLanes=lanes;
}
```

addHorse method:

```java
/**
 * Adds a horse to the race
 *
 * @param theHorse the horse to be added to the race
 */
public void addHorse(Horse theHorse) {
    if (horses.size() < numberOfLanes) {
        horses.add(theHorse);
    } else {
        System.out.println(x:"Cannot add more horses. Maximum number of lanes reached.");
    }
}
```

startRace method:

```java
/**
 * Start the race
 * The horse are brought to the start and
 * then repeatedly moved forward until the
 * race is finished
 */

public void startRace() {
    // Reset all horses to the start
    for (Horse horse : horses) {
        horse.goBackToStart();
    }

    double time = 0.0; // Initialize time

    double fall = -1.0;

    // Continue the race until all horses finish or fall
    while (!allHorsesFinishedOrFallen()) {
        // Clear the terminal
        clearTerminal();

        // Move each horse
        for (Horse horse : horses) {
            moveHorse(horse);
            if (horse.hasFallen() && !results.containsKey(horse)) {
                results.put(horse, fall--);
            }
            if (raceWonBy(horse) && !results.containsKey(horse)) {
                results.put(horse, time);
            }
        }

        // Print the race positions and horse confidence
        printRace();

        // Increment time by 0.1 seconds
        time += 0.1;
```

printRace method:

```java
/**
 * Print the race on the terminal
 */
private void printRace()
{
    clearTerminal();

    multiplePrint(aChar:'=',raceLength+3); //top edge of track
    System.out.println();

    for (Horse horse : horses) {
        printLane(horse);
        System.out.print(" " + horse.getName() + " Confidence: " + horse.getConfidence() + " ");
        if (horse.hasFallen()) System.out.print(s:"Fallen");
        else if (results.containsKey(horse)) System.out.print("Finished, time: " + Math.round(results.get(horse) * 100.0) / 100.0);
        System.out.println();
    }

    multiplePrint(aChar:'=',raceLength+3); //bottom edge of track
    System.out.println();
}
```

The hasAllHorsesFallen method was deleted as it became obsolete.

main method:

```java
/**
 * main Method of the Race class, Takes user input to create and start a race
 */
Run | Debug
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Getting race length with input validation
    int raceLength;
    while (true) {
        try {
            System.out.print(s:"Enter the race length: ");
            raceLength = scanner.nextInt();
            if (raceLength <= 0) {
                throw new IllegalArgumentException(s:"Race length must be a positive integer.");
            }
            break; // Break out of the loop if input is valid
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid input. Please enter a valid integer.");
            scanner.nextLine(); // Clear the input buffer
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }

    // Getting the number of horses participating in the race
    int numberOfHorses;
    while (true) {
        try {
            System.out.print(s:"Enter the number of horses: ");
            numberOfHorses = scanner.nextInt();
            if (numberOfHorses <= 0) {
                throw new IllegalArgumentException(s:"Number of horses must be a positive integer.");
            }
            break; // Break out of the loop if input is valid
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid input. Please enter a valid integer.");
            scanner.nextLine(); // Clear the input buffer
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
```

```
    Race testRace = new Race(raceLength, numberOfHorses);

    // Adding the horses from input to the race
    for (int i = 0; i < numberOfHorses; i++) {
        String horseName;
        char horseSymbol;
        double horseConfidence;

        // Getting horse name
        System.out.print("Enter name for horse " + (i + 1) + ": ");
        horseName = scanner.next();

        // Getting horse symbol
        System.out.print("Enter symbol for horse " + (i + 1) + ": ");
        horseSymbol = scanner.next().charAt(index:0);

        // Getting horse confidence with input validation
        while (true) {
            try {
                System.out.print("Enter confidence for horse " + (i + 1) + ": ");
                horseConfidence = scanner.nextDouble();
                if (horseConfidence < 0 || horseConfidence > 1) {
                    throw new IllegalArgumentException(s:"Confidence must be a number between 0 and 1.");
                }
                break; // Break out of the loop if input is valid
            } catch (InputMismatchException e) {
                System.out.println(x:"Invalid input. Please enter a valid number.");
                scanner.nextLine(); // Clear the input buffer
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }

        // Adding horse to the race
        testRace.addHorse(new Horse(horseSymbol, horseName, horseConfidence));
    }

    // Start the race
    testRace.startRace();
    scanner.close();
}
```

AllHorsesFinishedOrFallen method:

```
/**
 * Check if all horses have finished the race or fallen.
 *
 * @return true if all horses have finished or fallen, false otherwise
 */
private boolean allHorsesFinishedOrFallen() {
    for (Horse horse : horses) {
        if (horse.getDistanceTravelled() == raceLength || horse.hasFallen()) {
            continue;
        }
        return false;
    }
    return true;
}
```

All these methods were changed to incorporate this implementation.

Testing and output:

```
Enter the race length: 20
Enter the number of horses: 5
Enter name for horse 1: A
Enter symbol for horse 1: A
Enter confidence for horse 1: 0.9
Enter name for horse 2: B
Enter symbol for horse 2: B
Enter confidence for horse 2: 0.8
Enter name for horse 3: C
Enter symbol for horse 3: C
Enter confidence for horse 3: 0.85
Enter name for horse 4: D
Enter symbol for horse 4: D
Enter confidence for horse 4: 0.5
Enter name for horse 5: E
Enter symbol for horse 5: E
Enter confidence for horse 5: 0.7
```

```
======================
|                    A| A Confidence: 0.9 Finished, time: 2.2
|                    B| B Confidence: 0.8 Finished, time: 2.0
|                    C| C Confidence: 0.85 Finished, time: 2.0
|                    D| D Confidence: 0.5 Finished, time: 2.4
| ?                   | E Confidence: 0.7 Fallen
======================
Results:
C: 2.0
B: 2.0
A: 2.2
D: 2.4
E: DNF
```

```
Enter the race length: 30
Enter the number of horses: 6
Enter name for horse 1: Jojo
Enter symbol for horse 1: J
Enter confidence for horse 1: 0.8
Enter name for horse 2: Gyro
Enter symbol for horse 2: G
Enter confidence for horse 2: 0.8
Enter name for horse 3: Sandman
Enter symbol for horse 3: S
Enter confidence for horse 3: 0.7
Enter name for horse 4: Dio
Enter symbol for horse 4: D
Enter confidence for horse 4: 0.9
Enter name for horse 5: Valentine
Enter symbol for horse 5: V
Enter confidence for horse 5: 0.8
Enter name for horse 6: Pocoloco
Enter symbol for horse 6: P
Enter confidence for horse 6: 0.6
```

```
================================
|                             J| Jojo Confidence: 0.8 Finished, time: 3.2
|              ?               | Gyro Confidence: 0.8 Fallen
|                             S| Sandman Confidence: 0.7 Finished, time: 3.4
|                             D| Dio Confidence: 0.9 Finished, time: 3.0
|                             V| Valentine Confidence: 0.8 Finished, time: 3.2
|                    ?         | Pocoloco Confidence: 0.6 Fallen
================================
Results:
Dio: 3.0
Valentine: 3.2
Jojo: 3.2
Sandman: 3.4
Pocoloco: DNF
Gyro: DNF
```

Now before adding the Championship simulation, I first decided to make a points Map to store the points the horses receive regarding their position in the race. So First I added the attribute to the class:

```
private Map<Horse, Integer> points = new HashMap<>();
```

Then I had to change the startRace method, and had to make 2 new methods; calculatePoints and PrintResults:

```
21   public class Race
64       public void startRace() {
79               // Move each horse
80               for (Horse horse : horses) {
81                   moveHorse(horse);
82                   if (horse.hasFallen() && !results.containsKey(horse)) {
83                       results.put(horse, fall--);
84                   }
85                   if (raceWonBy(horse) && !results.containsKey(horse)) {
86                       results.put(horse, time);
87                   }
88               }
89
90               // Print the race positions and horse confidence
91               printRace();
92
93               // Increment time by 0.1 seconds
94               time += 0.1;
95
96               // Wait for 100 milliseconds
97               try {
98                   TimeUnit.MILLISECONDS.sleep(timeout:100);
99               } catch (InterruptedException e) {
100                  e.printStackTrace();
101              }
102          }
103          clearTerminal();
104          printRace();
105
106          results = sortMap(results);
107          calculatePoints();
108          printResults();
109
110      }
```

Only the lines with the green highlights where changed/added, form line 103 to 108.

```java
/**
 * calculates the points earned by each horse once the race finishes, and puts them in the points map.
 */
private void calculatePoints() {
    int numberOfLanes = horses.size();
    int pointsEarned = numberOfLanes;

    // Sort the race results by time taken to finish
    List<Map.Entry<Horse, Double>> entries = new ArrayList<>(results.entrySet());
    Collections.sort(entries, Comparator.comparing(Map.Entry::getValue));

    // Track the previous time to detect ties in the race
    double previousTime =-0.1;

    // Iterate over the sorted entries to assign points to horses
    for (Map.Entry<Horse, Double> entry : entries) {
        // Get the horse and its finishing time
        Horse horse = entry.getKey();
        double time = entry.getValue();

        // Decrement points earned if there's a new position or a tie
        if (time >= 0 && time != previousTime) {
            pointsEarned--;
            previousTime = time;
        }

        // Assign points to the horse based on its finishing position
        // If the horse didn't finish, assign 0 points
        points.put(horse, time >= 0 ? pointsEarned : 0);
    }
}
```

```java
/**
 * Prints the race results including each horse's position and points earned.
 */
private void printResults() {

    // Sort the race results by time taken to finish
    List<Map.Entry<Horse, Double>> entries = new ArrayList<>(results.entrySet());
    Collections.sort(entries, Comparator.comparing(Map.Entry::getValue));

    // Track the position of horses in the race
    int position = 1;

    // Iterate over the sorted entries to print results
    for (Map.Entry<Horse, Double> entry : entries) {
        // Get the horse and its finishing time
        Horse horse = entry.getKey();
        double time = entry.getValue();

        // Print the horse's name, time taken, position, and points earned
        System.out.print(horse.getName() + ": ");
        if (time >= 0) {
            System.out.print("Time: " + Math.round(time * 100.0) / 100.0 + " seconds, ");
            System.out.print("Position: " + position + ", ");
            System.out.println("Points: " + points.get(horse));
        } else {
            // If the horse didn't finish, print DNF (Did Not Finish)
            System.out.print(s:"Position: DNF ");
            System.out.println("Points: " + points.get(horse));
        }

        // Increment the position for the next horse
        position++;
    }
}
```

Testing and output:

```
Enter the race length: 30
Enter the number of horses: 5
Enter name for horse 1: A
Enter symbol for horse 1: A
Enter confidence for horse 1: 0.8
Enter name for horse 2: B
Enter symbol for horse 2: B
Enter confidence for horse 2: 0.7
Enter name for horse 3: C
Enter symbol for horse 3: C
Enter confidence for horse 3: 0.6
Enter name for horse 4: D
Enter symbol for horse 4: D
Enter confidence for horse 4: 0.8
Enter name for horse 5: E
Enter symbol for horse 5: E
Enter confidence for horse 5: 0.8
```

```
==============================
|                            A| A Confidence: 0.8 Finished, time: 3.6
|                            B| B Confidence: 0.7 Finished, time: 3.2
|                            C| C Confidence: 0.6 Finished, time: 3.3
|                            D| D Confidence: 0.8 Finished, time: 3.4
|                            E| E Confidence: 0.8 Finished, time: 3.1
==============================
E: Time: 3.1 seconds, Position: 1, Points: 4
B: Time: 3.2 seconds, Position: 2, Points: 3
C: Time: 3.3 seconds, Position: 3, Points: 2
D: Time: 3.4 seconds, Position: 4, Points: 1
A: Time: 3.6 seconds, Position: 5, Points: 0
```

```
Enter the race length: 20
Enter the number of horses: 3
Enter name for horse 1: Lion
Enter symbol for horse 1: L
Enter confidence for horse 1: 0.6
Enter name for horse 2: Tiger
Enter symbol for horse 2: T
Enter confidence for horse 2: 0.55
Enter name for horse 3: Cheetah
Enter symbol for horse 3: C
Enter confidence for horse 3: 0.9
```

```
======================
|                   L| Lion Confidence: 0.6 Finished, time: 2.1
|                   T| Tiger Confidence: 0.55 Finished, time: 3.2
|                   C| Cheetah Confidence: 0.9 Finished, time: 1.9
======================
Cheetah: Time: 1.9 seconds, Position: 1, Points: 2
Lion: Time: 2.1 seconds, Position: 2, Points: 1
Tiger: Time: 3.2 seconds, Position: 3, Points: 0
```

```
Enter the race length: 100
Enter the number of horses: 2
Enter name for horse 1: Gon
Enter symbol for horse 1: G
Enter confidence for horse 1: 0.99
Enter name for horse 2: Killua
Enter symbol for horse 2: K
Enter confidence for horse 2: 0.99
```

```
================================================================================
|                                                                 ?             | Gon Confidence: 0.99 Fallen
|                   ?                                                           | Killua Confidence: 0.99 Fallen
================================================================================
Gon: Position: DNF Points: 0
Killua: Position: DNF Points: 0
```

Okay now with the points system in place, I made many new methods and edited some old ones to now implement the option to simulate Championships, which is multiple races that are held in succession, and the points from all the races are kept track to display the winners and the points

table at the end. I also implemented the characteristic of horses where if they fall their confidences decrease, and if they finish the race their confidences increase relative to the position they finish in (so first place gets a big confidence boost, last place gets little boost). Here are all the methods I changed/ added, and I will provide screenshots below.

In Race class:

- Added new attribute overallPoints which is a Map
- Changed startRace method
- Added getRaceLength method
- Added getNumberOfHorses method
- Added getHorseName method
- Added getHorseSymbol method
- Added getHorseConfidence method
- Added updateConfidence method
- Added printOverallResults method
- Changed calculatePoints method
- Changed main method
- Added runSingleRace method
- Added runChampionship method

In Horse class:

- Added reset method

Screenshots:

In Race class:

```
private Map<Horse, Integer> overallPoints = new HashMap<>();
```

```java
    /**
     * Start the race
     * The horse are brought to the start and
     * then repeatedly moved forward until the
     * race is finished
     */

    public void startRace() {
        // Reset all horses to the start
        for (Horse horse : horses) {
            horse.goBackToStart();
            horse.reset();
        }

        double time = 0.0; // Initialize time

        double fall = -1.0;
        int placement = numberOfLanes;

        // Continue the race until all horses finish or fall
        while (!allHorsesFinishedOrFallen()) {
            // Clear the terminal
            clearTerminal();

            // Move each horse
            for (Horse horse : horses) {
                moveHorse(horse);
                if (horse.hasFallen() && !results.containsKey(horse)) {
                    results.put(horse, fall--);
                    updateConfidence(horse, -1); // Decrease confidence if the horse falls
                }
                if (raceWonBy(horse) && !results.containsKey(horse)) {
                    results.put(horse, time);
                    updateConfidence(horse, placement); // Increase confidence if the horse wins
                    placement--; // Decrease placement for the next horse
                }
            }

            // Print the race positions and horse confidence
            printRace();

            // Increment time by 0.1 seconds
            time += 0.1;

            // Wait for 100 milliseconds
            try {
                TimeUnit.MILLISECONDS.sleep(timeout:100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        clearTerminal();
        printRace();

        results = sortMap(results);
        calculatePoints();
        printResults();
    }
```

```java
    /**
     * Helper method to get the race length from the user.
     * @param scanner The Scanner object for user input
     * @return The race length entered by the user
     */
    private static int getRaceLength(Scanner scanner) {
        int raceLength;
        while (true) {
            try {
                System.out.print(s:"Enter the race length: ");
                raceLength = scanner.nextInt();
                if (raceLength <= 0) {
                    throw new IllegalArgumentException(s:"Race length must be a positive integer.");
                }
                break; // Break out of the loop if input is valid
            } catch (InputMismatchException e) {
                System.out.println(x:"Invalid input. Please enter a valid integer.");
                scanner.nextLine(); // Clear the input buffer
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }
        return raceLength;
    }
```

```java
    /**
     * Helper method to get the number of horses participating from the user.
     * @param scanner The Scanner object for user input
     * @return The number of horses entered by the user
     */
    private static int getNumberOfHorses(Scanner scanner) {
        int numberOfHorses;
        while (true) {
            try {
                System.out.print(s:"Enter the number of horses: ");
                numberOfHorses = scanner.nextInt();
                if (numberOfHorses <= 0) {
                    throw new IllegalArgumentException(s:"Number of horses must be a positive integer.");
                }
                break; // Break out of the loop if input is valid
            } catch (InputMismatchException e) {
                System.out.println(x:"Invalid input. Please enter a valid integer.");
                scanner.nextLine(); // Clear the input buffer
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }
        return numberOfHorses;
    }
```

```java
/**
 * Helper method to get the name of a horse from the user.
 * @param scanner The Scanner object for user input
 * @param horseNumber The number of the horse for which the name is being entered
 * @return The name of the horse entered by the user
 */
private static String getHorseName(Scanner scanner, int horseNumber) {
    System.out.print("Enter name for horse " + horseNumber + ": ");
    return scanner.next();
}

/**
 * Helper method to get the symbol of a horse from the user.
 * @param scanner The Scanner object for user input
 * @param horseNumber The number of the horse for which the symbol is being entered
 * @return The symbol of the horse entered by the user
 */
private static char getHorseSymbol(Scanner scanner, int horseNumber) {
    System.out.print("Enter symbol for horse " + horseNumber + ": ");
    return scanner.next().charAt(index:0);
}
```

```java
/**
 * Helper method to get the confidence of a horse from the user.
 * @param scanner The Scanner object for user input
 * @param horseNumber The number of the horse for which the confidence is being entered
 * @return The confidence of the horse entered by the user
 */
private static double getHorseConfidence(Scanner scanner, int horseNumber) {
    double horseConfidence;
    while (true) {
        try {
            System.out.print("Enter confidence for horse " + horseNumber + ": ");
            horseConfidence = scanner.nextDouble();
            if (horseConfidence < 0 || horseConfidence > 1) {
                throw new IllegalArgumentException(s:"Confidence must be a number between 0 and 1.");
            }
            break; // Break out of the loop if input is valid
        } catch (InputMismatchException e) {
            System.out.println(x:"Invalid input. Please enter a valid number.");
            scanner.nextLine(); // Clear the input buffer
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
    return horseConfidence;
}
```

```java
/**
 * Update the confidence of a horse based on its performance in the current race.
 * Confidence increases if the horse finishes well, decreases if it falls.
 * @param horse The horse whose confidence needs to be updated
 * @param placement The placement of the horse in the race (-1 if the horse fell)
 */
public void updateConfidence(Horse horse, int placement) {
    double currentConfidence = horse.getConfidence();

    if (placement > 0) {
        // Increase confidence more for higher positions
        double increase = 0.2 / (numberOfLanes-placement+1); // Adjust this factor as needed
        horse.setConfidence(Math.min(currentConfidence + increase, b:0.99)); // capping max confidence at 0.99
    } else {
        // Decrease confidence
        double decrease = 0.2; // Adjust this factor as needed
        horse.setConfidence(Math.max(currentConfidence - decrease, b:0.1)); // capping min confidence at 0.1
    }
}

/**
 * Prints the overall results including each horse's position and points earned across all races.
 */
private void printOverallResults(Map<Horse, Integer> overallPoints) {
    System.out.println(x:"Overall Results:");
    // Sort the horses based on their overall points
    List<Horse> sortedHorses = new ArrayList<>(overallPoints.keySet());
    Collections.sort(sortedHorses, Comparator.comparingInt(overallPoints::get).reversed());

    // Print the overall position and points of each horse
    int position = 1;
    for (Horse horse : sortedHorses) {
        int pointsEarned = overallPoints.get(horse);
        System.out.println("Position " + position + ": " + horse.getName() + " - Points: " + pointsEarned);
        position++;
    }
}
```

```java
/**
 * Calculates the points earned by each horse once the race finishes, and updates the overall points earned by each horse in the championship.
 */
private void calculatePoints() {
    int numberOfLanes = horses.size();
    int pointsEarned = numberOfLanes;

    // Sort the race results by time taken to finish
    List<Map.Entry<Horse, Double>> entries = new ArrayList<>(results.entrySet());
    Collections.sort(entries, Comparator.comparing(Map.Entry::getValue));

    // Track the previous time to detect ties in the race
    double previousTime = -0.1;

    // Iterate over the sorted entries to assign points to horses
    for (Map.Entry<Horse, Double> entry : entries) {
        // Get the horse and its finishing time
        Horse horse = entry.getKey();
        double time = entry.getValue();

        // Decrement points earned if there's a new position or a tie
        if (time >= 0 && time != previousTime) {
            pointsEarned--;
            previousTime = time;
        }

        // Assign points to the horse based on its finishing position
        // If the horse didn't finish, assign 0 points
        points.put(horse, time >= 0 ? pointsEarned : 0);

        // Update overall points earned by the horse in the championship
        overallPoints.put(horse, overallPoints.getOrDefault(horse, defaultValue:0) + (time >= 0 ? pointsEarned : 0));
    }
}
```

```java
    /**
     * Main Method of the Race class, Takes user input to create and start a race
     */
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Getting user choice between Single Race and Championship
        int choice;
        while (true) {
            System.out.println(x:"Welcome to Horse Race Simulator!");
            System.out.println(x:"Choose an option:");
            System.out.println(x:"1. Single Race");
            System.out.println(x:"2. Championship");
            System.out.print(s:"Enter your choice (1 or 2): ");
            try {
                choice = scanner.nextInt();
                if (choice != 1 && choice != 2) {
                    throw new IllegalArgumentException(s:"Invalid choice. Please enter 1 or 2.");
                }
                break; // Break out of the loop if input is valid
            } catch (InputMismatchException e) {
                System.out.println(x:"Invalid input. Please enter a valid integer.");
                scanner.nextLine(); // Clear the input buffer
            } catch (IllegalArgumentException e) {
                System.out.println(e.getMessage());
            }
        }

        // Getting horse details
        List<Horse> horses = new ArrayList<>();
        int numberOfHorses= getNumberOfHorses(scanner);
        Race race = new Race(distance:0, numberOfHorses); // Create an instance of the Race class

        for (int i = 0; i < numberOfHorses; i++) {
            String horseName = getHorseName(scanner, i + 1);
            char horseSymbol = getHorseSymbol(scanner, i + 1);
            double horseConfidence = getHorseConfidence(scanner, i + 1);
            horses.add(new Horse(horseSymbol, horseName, horseConfidence));
        }

        if (choice == 1) {
            runSingleRace(scanner, horses); // Call the method for a single race
        } else if (choice == 2) {
            race.runChampionship(scanner, horses); // Call the method for a championship
        }

        scanner.close();
    }

    /**
     * Run a single race based on user input.
     * @param scanner The Scanner object for user input
     */
    private static void runSingleRace(Scanner scanner,List<Horse> horses) {
        // Getting race details from the user
        int raceLength = getRaceLength(scanner);
        Race singleRace = new Race(raceLength, horses.size());
        for(Horse horse:horses){
            singleRace.addHorse(horse);
        }

        // Start the single race
        singleRace.startRace();
    }
```

```java
/**
 * Run a single race based on user input.
 * @param scanner The Scanner object for user input
 */
private static void runSingleRace(Scanner scanner,List<Horse> horses) {
    // Getting race details from the user
    int raceLength = getRaceLength(scanner);
    Race singleRace = new Race(raceLength, horses.size());
    for(Horse horse:horses){
        singleRace.addHorse(horse);
    }

    // Start the single race
    singleRace.startRace();
}
```

```java
/**
 * Run a championship consisting of multiple races.
 * @param scanner The Scanner object for user input
 * @param horses The list of horses participating in the championship
 */
public void runChampionship(Scanner scanner, List<Horse> horses) {
    // Prompt for the number of races in the championship
    System.out.print(s:"Enter the number of races in the championship: ");
    int numRaces = scanner.nextInt();

    // Initialize overall points for each horse to 0
    Map<Horse, Integer> overallPoints = new HashMap<>();
    for (Horse horse : horses) {
        overallPoints.put(horse, value:0);
    }

    // For each race in the championship
    for (int i = 1; i <= numRaces; i++) {
        System.out.println("Race " + i + " of " + numRaces);

        int length = getRaceLength(scanner);
        // Transfer overall points data to the new Race object
        Race newRace = new Race(length, numberOfLanes);
        newRace.horses = horses; // Transfer horses
        newRace.overallPoints = overallPoints; // Transfer overallPoints data
        newRace.startRace();


        // Update overall points for each horse based on the current race results
        for (Map.Entry<Horse, Integer> entry : points.entrySet()) {
            Horse horse = entry.getKey();
            int pointsEarned = entry.getValue();
            overallPoints.put(horse, overallPoints.get(horse) + pointsEarned);
        }

        // Prompt to continue to the next race
        if (i < numRaces) {
            System.out.print(s:"Do you want to continue to the next race? (y/n): ");
            String input = scanner.next();
            if (!input.equalsIgnoreCase(anotherString:"y")) {
                break;
            }
        }
    }

    // Print overall results after all races are finished
    printOverallResults(overallPoints);
}
```

In Horse class:

```java
    /**
     * Method to make fallen= flase
     */
    public void reset()
    {
        fallen=false;
        return;
    }
```

Testing and output:

Single Race:

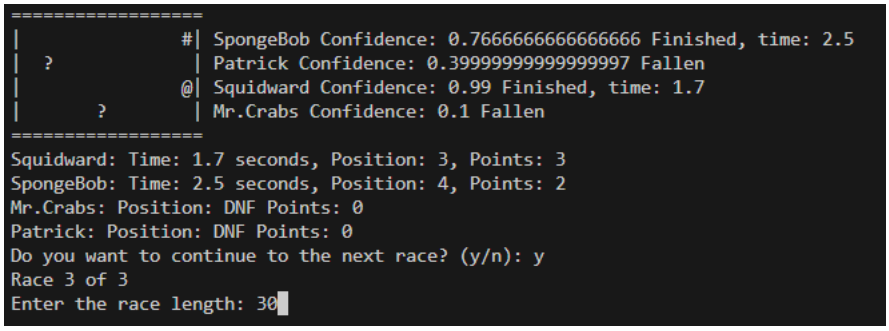
```
Welcome to Horse Race Simulator!
Choose an option:
1. Single Race
2. Championship
Enter your choice (1 or 2): 1
Enter the number of horses: A
Invalid input. Please enter a valid integer.
Enter the number of horses: 2
Enter name for horse 1: A
Enter symbol for horse 1: A
Enter confidence for horse 1: 0.5
Enter name for horse 2: B
Enter symbol for horse 2: B
Enter confidence for horse 2: 0.5
Enter the race length: 20
```

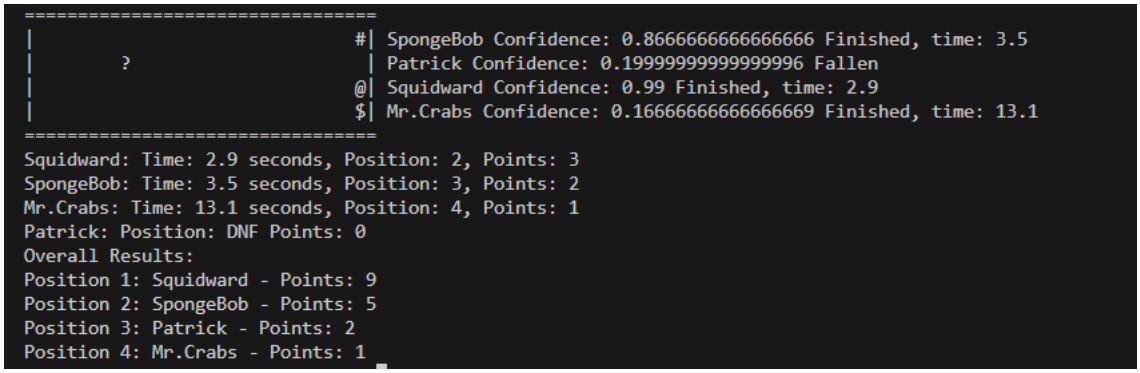
```
======================
|              ?     | A Confidence: 0.3 Fallen
|         ?          | B Confidence: 0.3 Fallen
======================
A: Position: DNF Points: 0
B: Position: DNF Points: 0
```

Championship:

```
Welcome to Horse Race Simulator!
Choose an option:
1. Single Race
2. Championship
Enter your choice (1 or 2): 2
Enter the number of horses: 4
Enter name for horse 1: SpongeBob
Enter symbol for horse 1: #
Enter confidence for horse 1: 0.6
Enter name for horse 2: Patrick
Enter symbol for horse 2: *
Enter confidence for horse 2: 0.5
Enter name for horse 3: Squidward
Enter symbol for horse 3: @
Enter confidence for horse 3: 0.6
Enter name for horse 4: Mr.Crabs
Enter symbol for horse 4: $
Enter confidence for horse 4: 0.5
Enter the number of races in the championship: 3
Race 1 of 3
Enter the race length: 20
```

```
======================
|                   #| SpongeBob Confidence: 0.6666666666666666 Finished, time: 3.4
|                   *| Patrick Confidence: 0.6 Finished, time: 3.2
|                   @| Squidward Confidence: 0.8 Finished, time: 2.8
|       ?             | Mr.Crabs Confidence: 0.3 Fallen
======================
Squidward: Time: 2.8 seconds, Position: 2, Points: 3
Patrick: Time: 3.2 seconds, Position: 3, Points: 2
SpongeBob: Time: 3.4 seconds, Position: 4, Points: 1
Mr.Crabs: Position: DNF Points: 0
Do you want to continue to the next race? (y/n): y
Race 2 of 3
Enter the race length: 15
```





Many other tests were conducted to see if all the methods worked properly, they were all passed.

This concludes the improvements section of the of the Race class, and also the Part 1 of this project.

Committing Changes to GIT

As I have completed the improvements In the Horse and Race classes before GUI, I am going to commit this as version 1.0.1 .

```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directo
        modified:   Part1/Horse.java
        modified:   Part1/Race.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Part1/Horse.class
        Part1/Race$1.class
        Part1/Race.class

no changes added to commit (use "git add" and/or "git commit -a")
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator>
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> cd Part1
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1> git add Horse.java
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1> git add Race.java
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1> git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Horse.java
        modified:   Race.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Horse.class
        Race$1.class
        Race.class

PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1> git commit -m "Implemented races with multiple lanes, results of ra
ces, simulating championships, and other small improvements"
[main d2214bf] Implemented races with multiple lanes, results of races, simulating championships, and other small improv
ements
 2 files changed, 531 insertions(+), 86 deletions(-)
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part1>
```

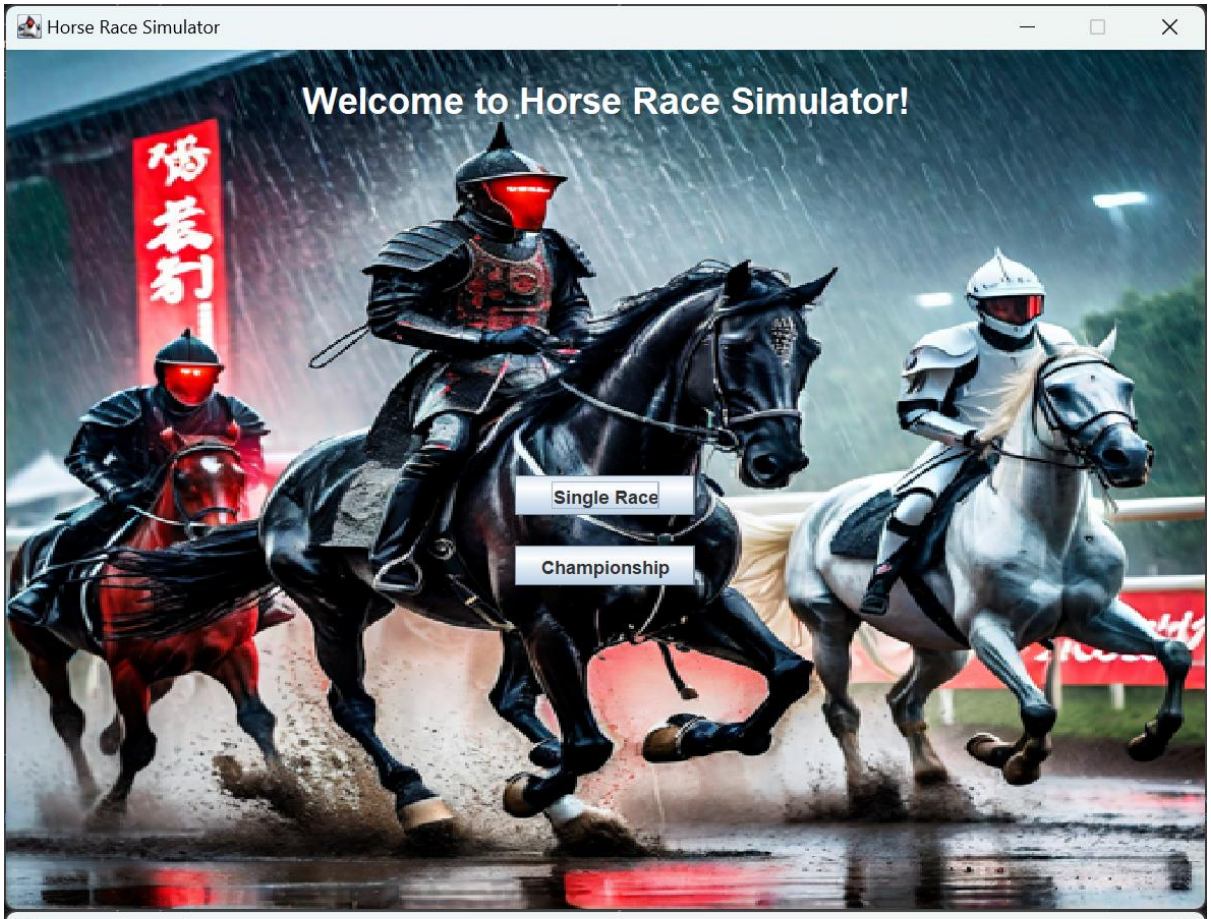
Files successfully committed in git, along with suitable message.

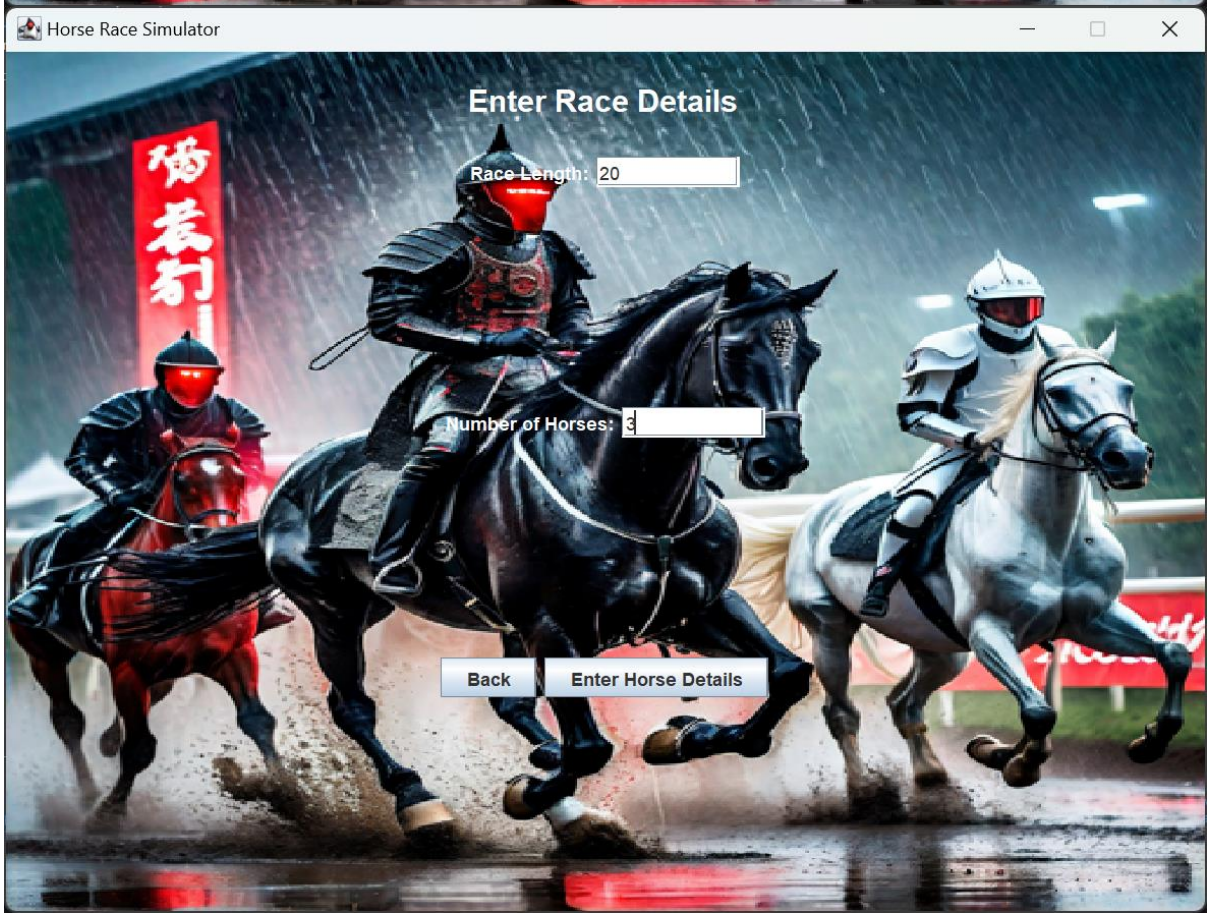
Now branching to develop GUI.

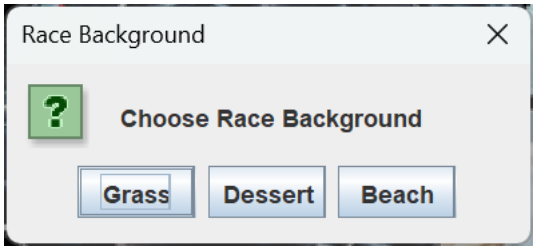
```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> cd Part2
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git branch gui-development
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git branch
  gui-development
* main
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git checkout "gui-development"
Switched to branch 'gui-development'
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git status
On branch gui-development
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../Part1/Horse.class
        ../Part1/Race$1.class
        ../Part1/Race.class
        ./

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add Horse.java
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add Race.java
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2>  git status
On branch gui-development
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Horse.java
        new file:   Race.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../Part1/Horse.class
        ../Part1/Race$1.class
        ../Part1/Race.class
        Horse.class
        Race$1.class
        Race.class

PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git commit -m "making new branch to develop GUI"
[gui-development 3fa466a] making new branch to develop GUI
 2 files changed, 836 insertions(+)
 create mode 100644 Part2/Horse.java
 create mode 100644 Part2/Race.java
```

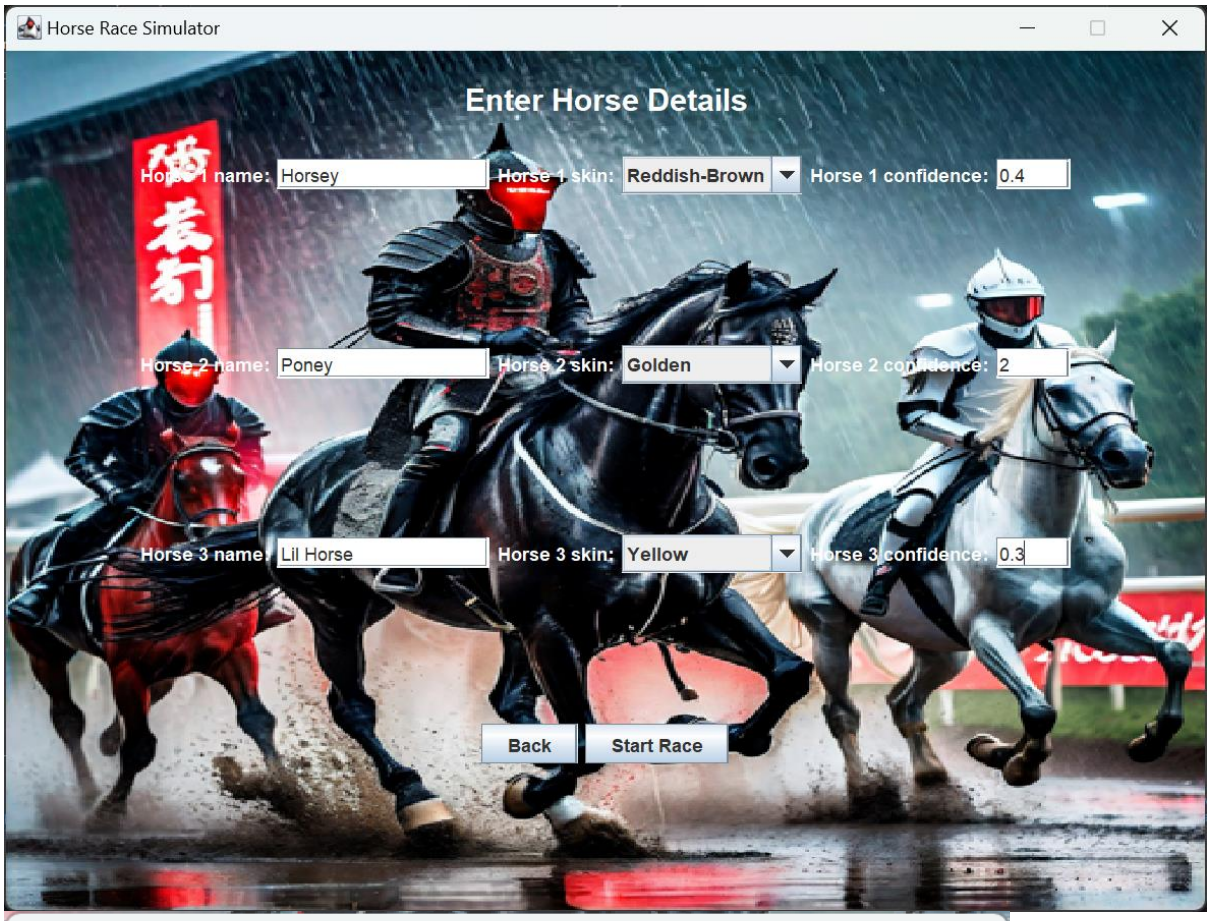
Part II – GUI

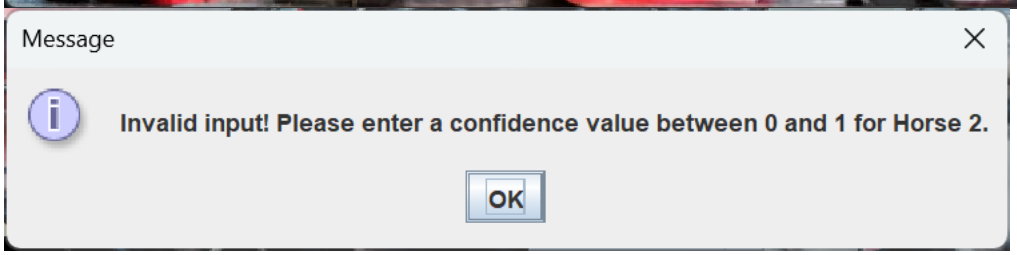
As it was not required, I did not document my journey of making the GUI. Here are some images of the final product:

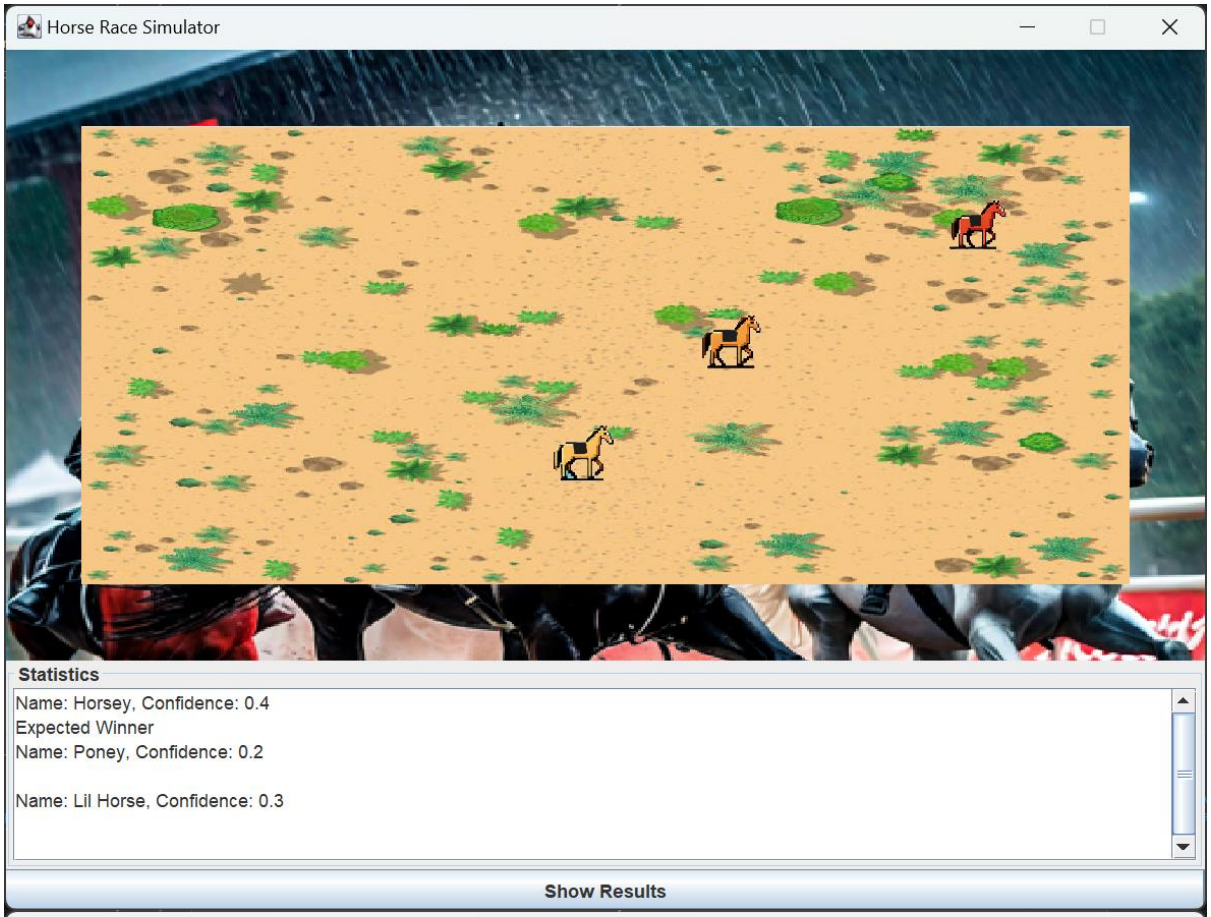

Horse Race Simulator
Welcome to Horse Race Simulator!
Single Race
Championship

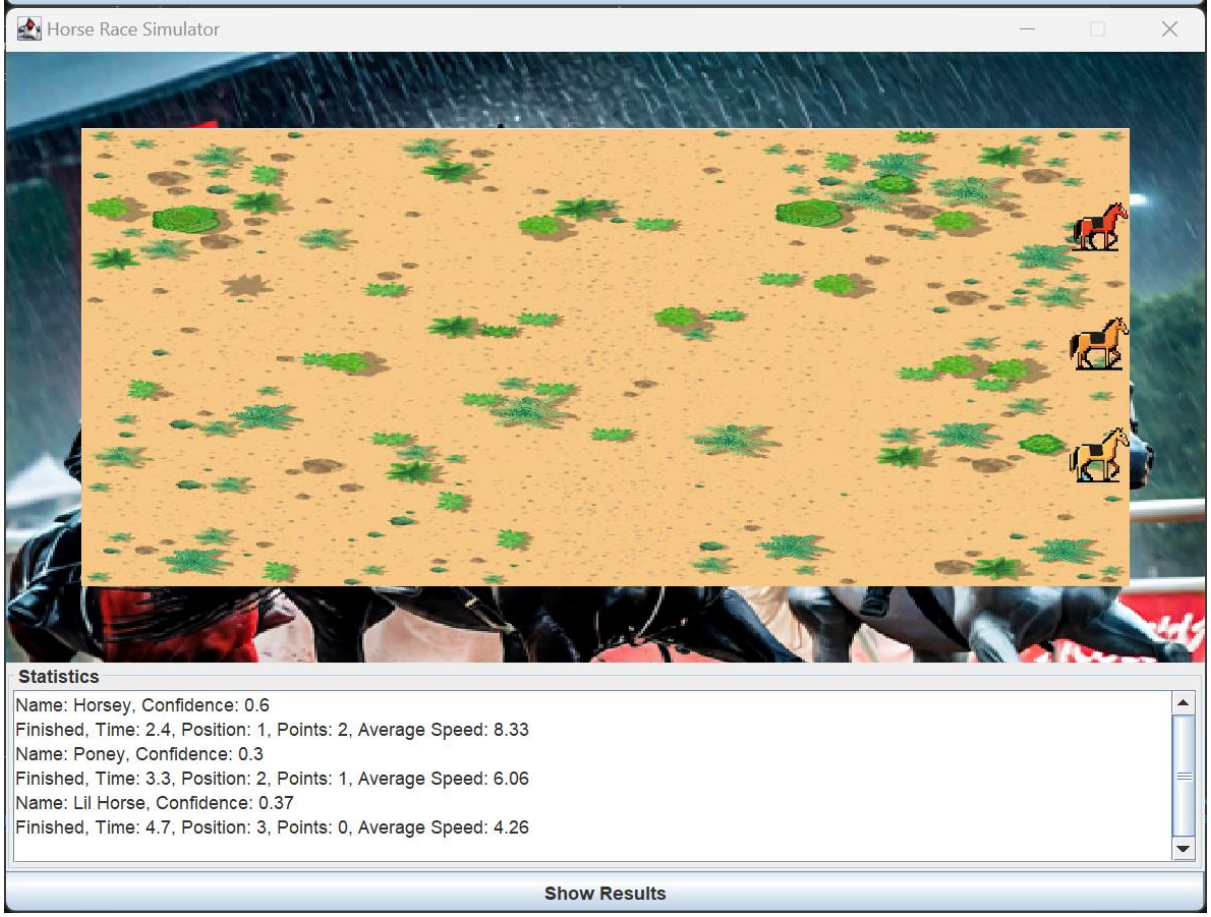

Horse Race Simulator
Enter Race Details
Race Length: 20
Number of Horses: 3
Back
Enter Horse Details

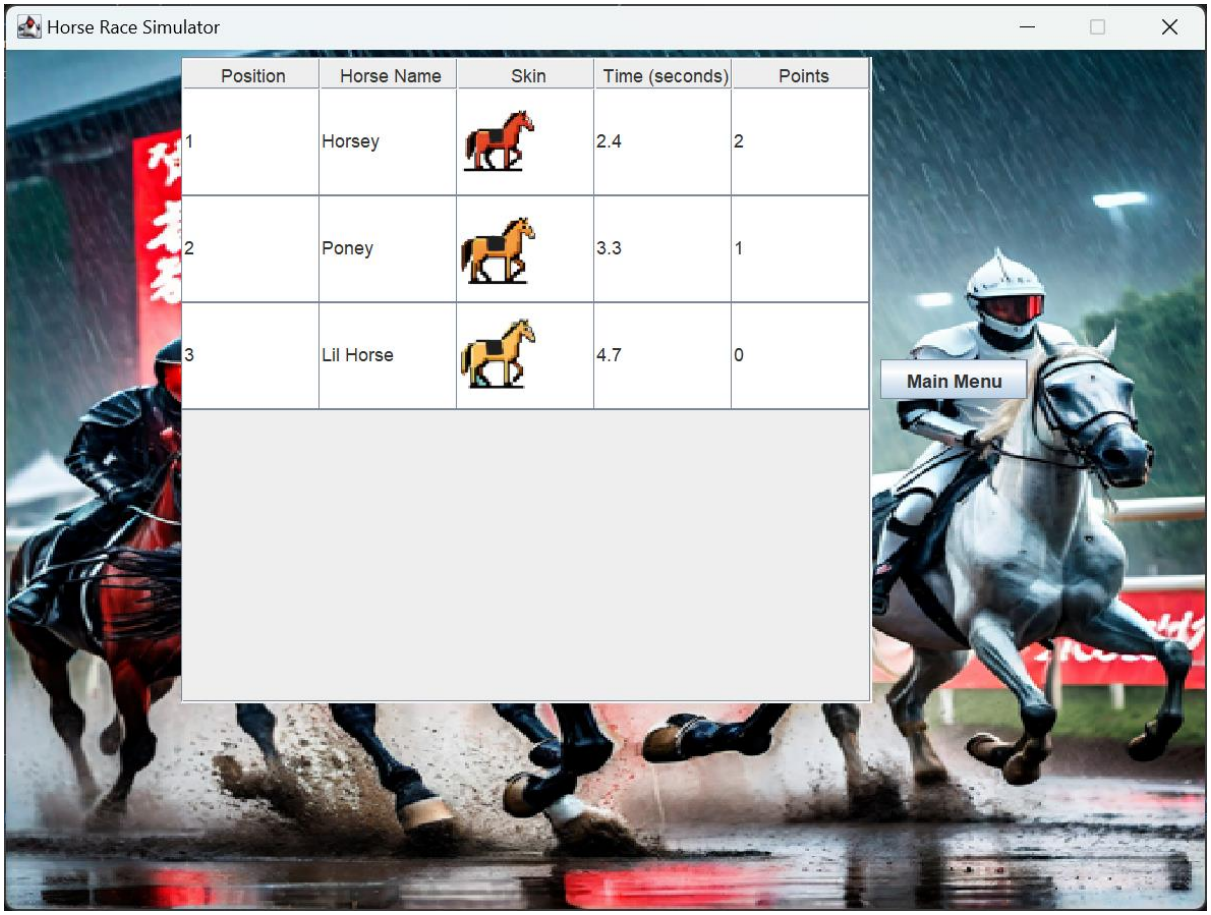

Race Background
Choose Race Background
Grass
Dessert
Beach


Horse Race Simulator
Enter Horse Details
Horse 1 name: Horsey
Horse 1 skin: Reddish-Brown
Horse 1 confidence: 0.4
Horse 2 name: Poney
Horse 2 skin: Golden
Horse 2 confidence: 2
Horse 3 name: Lil Horse
Horse 3 skin: Yellow
Horse 3 confidence: 0.3
Back
Start Race


Message
Invalid input! Please enter a confidence value between 0 and 1 for Horse 2.
OK


Horse Race Simulator
Statistics
Name: Horsey, Confidence: 0.4
Expected Winner
Name: Poney, Confidence: 0.2
Name: Lil Horse, Confidence: 0.3
Show Results


Horse Race Simulator
Statistics
Name: Horsey, Confidence: 0.6
Finished, Time: 2.4, Position: 1, Points: 2, Average Speed: 8.33
Name: Poney, Confidence: 0.3
Finished, Time: 3.3, Position: 2, Points: 1, Average Speed: 6.06
Name: Lil Horse, Confidence: 0.37
Finished, Time: 4.7, Position: 3, Points: 0, Average Speed: 4.26
Show Results



I did not show all the features of the GUI here, as there are many more error messages etc. In this option, and the whole other option of running a championship. I hope you have fun exploring that as well when marking.

Part3: Finishing parts

As you know I have been keeping my project up-to date on git, so once I finished developing the GUI I added it to the staging area and committed it. Here are some images of that:

```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git status
On branch gui-development
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Part1/Race.java
        modified:   Part2/Horse.java
        modified:   Part2/Race.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Part1/Horse.class
        Part1/Race$1.class
        Part1/Race.class
        Part2/BackgroundPanel.java
        Part2/Fallen.png
        Part2/Horse.class
        Part2/Horse1.png
        Part2/Horse2.png
        Part2/Horse3.png
        Part2/Horse4.png
        Part2/Horse5.png
        Part2/Horse6.png
        Part2/Horse7.png
        Part2/Horse8.png
        Part2/HorseBG.png
        Part2/HorseRaceGUI.java
        Part2/Race$1.class
        Part2/Race.class
        Part2/RaceBG1.png
        Part2/RaceBG2.png
        Part2/RaceBG3.png
        Part2/RaceDisplayPanel.java

no changes added to commit (use "git add" and/or "git commit -a")
```

Adding the files I missed on the Part1 Folder:

```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\part1> git add Horse.class
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\part1> git add Race.class
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\part1> git add Race$.class
fatal: pathspec 'Race$.class' did not match any files
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\part1> git add Race$1.class
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\part1> git commit -m "Adding the compiled versions"
[gui-development 541a6f4] Adding the compiled versions
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Part1/Horse.class
 create mode 100644 Part1/Race.class
```

Commiting Part 2:

```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> cd Part2
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add
Nothing specified, nothing added.
hint: Maybe you wanted to say 'git add .'?
hint: Turn this message off by running
hint: "git config advice.addEmptyPathspec false"
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2
fatal: pathspec 'Vasanth\Uni\Comp' did not match any files
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add "Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2"
warning: could not open directory 'Part2/Vasanth/Uni/Comp sci/HorseRaceSimulator/': No such file or directory
fatal: pathspec 'Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2' did not match any files
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git add .
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git status
On branch gui-development
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   BackgroundPanel.java
        new file:   Fallen.png
        new file:   Horse.class
        modified:   Horse.java
        new file:   Horse1.png
        new file:   Horse2.png
        new file:   Horse3.png
        new file:   Horse4.png
        new file:   Horse5.png
        new file:   Horse6.png
        new file:   Horse7.png
        new file:   Horse8.png
        new file:   HorseBG.png
        new file:   HorseRaceGUI.java
        new file:   Race$1.class
        new file:   Race.class
        modified:   Race.java
        new file:   RaceBG1.png
        new file:   RaceBG2.png
        new file:   RaceBG3.png
        new file:   RaceDisplayPanel.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ../Part1/Race.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        ../Part1/Race$1.class

PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator\Part2> git commit -m "Implemented a GUI, with customisable tracks and Horses, along with some statistics displayed."
[gui-development 0a2f340] Implemented a GUI, with customisable tracks and Horses, along with some statistics displayed.
 21 files changed, 1283 insertions(+), 7 deletions(-)
 create mode 100644 Part2/BackgroundPanel.java
 create mode 100644 Part2/Fallen.png
 create mode 100644 Part2/Horse.class
 create mode 100644 Part2/Horse1.png
 create mode 100644 Part2/Horse2.png
 create mode 100644 Part2/Horse3.png
 create mode 100644 Part2/Horse4.png
 create mode 100644 Part2/Horse5.png
 create mode 100644 Part2/Horse6.png
 create mode 100644 Part2/Horse7.png
 create mode 100644 Part2/Horse8.png
 create mode 100644 Part2/HorseBG.png
 create mode 100644 Part2/HorseRaceGUI.java
 create mode 100644 Part2/Race$1.class
 create mode 100644 Part2/Race.class
 create mode 100644 Part2/RaceBG1.png
 create mode 100644 Part2/RaceBG2.png
 create mode 100644 Part2/RaceBG3.png
 create mode 100644 Part2/RaceDisplayPanel.java
```

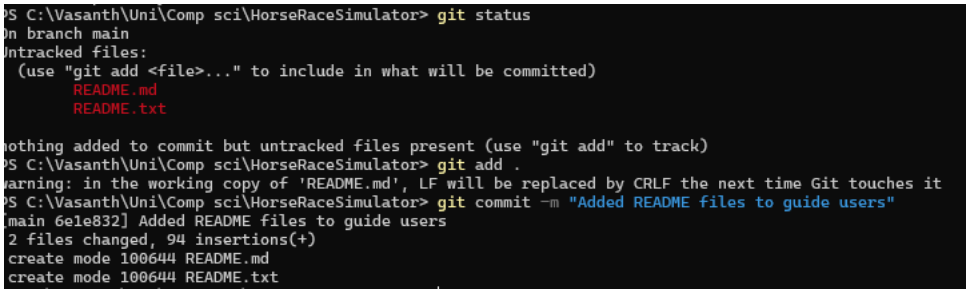
The git log so far:

```
commit 0a2f34043db79a44ca5fde522271fe2633879944 (HEAD -> gui-development)
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Thu Apr 25 19:11:29 2024 +0100

    Implemented a GUI, with customisable tracks and Horses, along with some statistics displayed.

commit 541a6f446ee9460cf1a7e5a2e6fde466c402d2a7
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Thu Apr 25 19:07:12 2024 +0100

    Adding the compiled versions

commit 3fa466a2d621a0f50caa8cd84a54285cdf201175
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Mon Apr 22 20:53:46 2024 +0100

    making new branch to develop GUI

commit d2214bf5b8fc0fea0476a6b63c6a83774c94ea92 (main)
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Mon Apr 22 20:41:08 2024 +0100

    Implemented races with multiple lanes, results of races, simulating championships, and other small improvements

commit aaf29963525e25d865a0bd6e22586f2611ec4dc0
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Tue Apr 2 21:15:16 2024 +0100

    Update Horse.java

    Completed the public methods, tested them and implemented encapsulation

commit 906fdd8a957f7c1c20a8221d5ca028dc2153cdc4
Author: Vasanth Subramanian <skjvasanth@gmail.com>
Date:   Wed Mar 20 16:54:20 2024 +0000

    Code from McFarewell
```

Merging the Branches:

```
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git status
On branch main
nothing to commit, working tree clean
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git merge gui-development
Updating d2214bf..df38b97
Fast-forward
 Part1/Horse.class           |  Bin 0 -> 2173 bytes
 Part1/Race$1.class          |  Bin 0 -> 1063 bytes
 Part1/Race.class            |  Bin 0 -> 7553 bytes
 Part1/Race.java             |    2 +-
 Part2/BackgroundPanel.java  |   27 ++
 Part2/Fallen.png            |  Bin 0 -> 271522 bytes
 Part2/Horse.class           |  Bin 0 -> 2173 bytes
 Part2/Horse.java            |  182 +++++++
 Part2/Horse1.png            |  Bin 0 -> 17083 bytes
 Part2/Horse2.png            |  Bin 0 -> 16642 bytes
 Part2/Horse3.png            |  Bin 0 -> 15884 bytes
 Part2/Horse4.png            |  Bin 0 -> 17797 bytes
 Part2/Horse5.png            |  Bin 0 -> 14840 bytes
 Part2/Horse6.png            |  Bin 0 -> 16423 bytes
 Part2/Horse7.png            |  Bin 0 -> 19804 bytes
 Part2/Horse8.png            |  Bin 0 -> 16469 bytes
 Part2/HorseBG.png           |  Bin 0 -> 517882 bytes
 Part2/HorseRaceGUI.java     | 1094 ++++++++++++++++++++++++++++++++++++++++++++
 Part2/Race$1.class          |  Bin 0 -> 1063 bytes
 Part2/Race.class            |  Bin 0 -> 7553 bytes
 Part2/Race.java             |  716 ++++++++++++++++++++++++++
 Part2/RaceBG1.png           |  Bin 0 -> 2119908 bytes
 Part2/RaceBG2.png           |  Bin 0 -> 456819 bytes
 Part2/RaceBG3.png           |  Bin 0 -> 241222 bytes
 Part2/RaceDisplayPanel.java |   71 +++
 25 files changed, 2091 insertions(+), 1 deletion(-)
 create mode 100644 Part1/Horse.class
 create mode 100644 Part1/Race$1.class
 create mode 100644 Part1/Race.class
 create mode 100644 Part2/BackgroundPanel.java
 create mode 100644 Part2/Fallen.png
 create mode 100644 Part2/Horse.class
 create mode 100644 Part2/Horse.java
 create mode 100644 Part2/Horse1.png
 create mode 100644 Part2/Horse2.png
 create mode 100644 Part2/Horse3.png
 create mode 100644 Part2/Horse4.png
 create mode 100644 Part2/Horse5.png
 create mode 100644 Part2/Horse6.png
 create mode 100644 Part2/Horse7.png
 create mode 100644 Part2/Horse8.png
 create mode 100644 Part2/HorseBG.png
 create mode 100644 Part2/HorseRaceGUI.java
 create mode 100644 Part2/Race$1.class
 create mode 100644 Part2/Race.class
 create mode 100644 Part2/Race.java
 create mode 100644 Part2/RaceBG1.png
 create mode 100644 Part2/RaceBG2.png
 create mode 100644 Part2/RaceBG3.png
 create mode 100644 Part2/RaceDisplayPanel.java
PS C:\Vasanth\Uni\Comp sci\HorseRaceSimulator> git commit -m "Merged gui-development branch with main branch"
On branch main
nothing to commit, working tree clean
```

Adding the README files



That concludes my report, Hope you had fun running my code! If I had more time, I would implement the betting system and make things more polished, but that's improvements for another time.